8-2017

Algorithms for automated assignment of solution-state and solid-state protein NMR spectra.

Andrey Smelter
University of Louisville

ALGORITHMS FOR AUTOMATED ASSIGNMENT OF SOLUTION-STATE AND SOLID-STATE PROTEIN NMR SPECTRA

By

Andrey Smelter
Specialist Degree, Perm State University, Russia, 2010
M.S., University of Louisville, U.S.A., 2013

A Dissertation
Submitted to the Faculty of the
School of Interdisciplinary and Graduate Studies of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy
in
Interdisciplinary Studies: Specialization in Bioinformatics

School of Interdisciplinary and Graduate Studies
University of Louisville
Louisville, Kentucky

August 2017

ALGORITHMS FOR AUTOMATED ASSIGNMENT OF SOLUTION-STATE AND SOLID-STATE PROTEIN NMR SPECTRA

By

Andrey Smelter
Specialist Degree, Perm State University, Russia, 2010
M.S., University of Louisville, U.S.A., 2013

A Dissertation Approved on

July 17, 2017

By the following Dissertation Committee

____
Hunter N.B. Moseley, Ph.D. Advisor

____
Eric C. Rouchka, D.Sc. Co-Advisor

____
Juw Won Park, Ph.D.

____
Richard J. Wittebort, Ph.D.

____
Guy Brock, Ph.D.

DEDICATION

This dissertation is dedicated to my parents,
all my family and friends
for their support and encouragement.

ACKNOWLEDGEMENTS

I would like to thank a number of people I owe a debt of gratitude and without whom this dissertation work could not have been possible.

I am especially grateful to my advisor Dr. Hunter Moseley for his excellent support and guidance during my doctoral journey. I am thankful for the opportunity to learn so much from him and work on the project that requires knowledge across multiple areas. His broad knowledge in multiple disciplines, passion for diligent research, and ability to work on many unrelated projects at the same time always inspires me.

I am also thankful to my co-advisor Dr. Eric Rouchka who guided me through the entire Bioinformatics Ph.D. program and was always helpful with all the obstacles I encountered on my way towards the completion of the program and always provided valuable advice. I am thankful to Dr. Rouchka for the opportunity to be in Bioinformatics Ph.D. program and work in the Bioinformatics Lab.

I would like to thank Dr. Juw Won Park, Dr. Richard Wittebort, and Dr. Guy Brock for being on my dissertation committee. With Dr. Park I worked on a RNA sequencing project and learned invaluable bioinformatics tools and techniques necessary to analyze next generation sequencing data. I was lucky to work with Dr. Wittebort on experimental protein NMR project and gained insights on the problem domain from the spectroscopist perspective. With Dr. Brock I took statistical methods for bioinformatics class where I learned necessary statistical background which I used extensively in my research project.

I would like to thank my current and former lab members who made the lab a friendly and enjoyable place: Xi Chen, Sen Yao, Joshua Mitchell, Eugene Hinderer, Morgan Astra, Andrew Nova, Dr. Abdallah Eteleeb, Ernur Saka, Mohammed Sayed, and many others. I would like to thank Dr. Robert Flight for valuable advice and especially for his passion for open research and open science.

Last but not least, I would like to thank my parents for their support, love, and opportunities they provided me in my life. I am thankful to all my family and friends that always encouraged me and helped me get through the program. I am especially thankful to Marina Malovichko whom I met in Louisville during my years at the University of Louisville. I am thankful for her love and kindness. She is extremely supportive and caring, and she made me a better person.

ABSTRACT

ALGORITHMS FOR AUTOMATED ASSIGNMENT OF SOLUTION-STATE AND SOLID-STATE PROTEIN NMR SPECTRA

Andrey Smelter

July 17, 2017

Protein nuclear magnetic resonance spectroscopy (Protein NMR) is an invaluable analytical technique for studying protein structure, function, and dynamics. There are two major types of NMR spectroscopy that are used for investigation of protein structure – solution-state and solid-state NMR. Solution-based NMR spectroscopy is typically applied to proteins of small and medium size that are soluble in water. Solid-state NMR spectroscopy is amenable for proteins that are insoluble in water.

In the vast majority NMR-based protein studies, the first step after experiment optimization is the assignment of protein resonances via the association of chemical shift values to specific atoms in a protein macromolecule. Depending on the quality of the spectra, a manual protein resonance assignment process often requires a considerable amount of time, from weeks to months-worth of effort even, by an experienced NMR spectroscopist.

The resonance assignment processes for solution-state and solid-state protein NMR studies are conceptually similar, but have distinct differences due to the utilization of different NMR experiments and to the use of different resonances for grouping peaks into spin systems.

Currently, there is a shortage of robust, effective software tools that can perform solid-state protein resonance assignment and there is no general software that can perform both solution-state and solid-state protein resonance assignment in a reliable, automated fashion. Hence, the motivation of this research is to design and implement algorithms and software tools that will automate the resonance assignment problem.

As a result of this research, several algorithms and software packages that aid several important steps in the protein resonance assignment process were developed. For example, the nmrstarlib software package can access and utilize data deposited in the NMR-STAR format; the core of this library is the lexical analyzer for NMR-STAR syntax that acts as a generator-based state-machine for token processing. The jpredapi software package provides an easy-to-use API to submit and retrieve results from secondary structure prediction server. The single peak list and pairwise peak list registration algorithms address the problem of multiple sources of variance within single peak list and between different peak lists and is capable of calculating the match tolerance values necessary for spin system grouping. The single peak list and pairwise peak list grouping algorithms are based on the well-known DBSCAN clustering algorithm and are designed to group peaks into spin systems within single peak list as well as between different peak lists.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Protein resonance assignment

The process of protein resonance assignment of peaks derived from protein NMR spectra is the first critical step for the vast majority of studies of protein structure and dynamics by NMR. In most of the cases, the assignment of protein resonances is performed manually and can take a significant amount of time, ranging from weeks to several months of work depending on the difficulty of the assignment problem and the quality of spectra.

1.1.1 Solution-state NMR sequential protein resonance manual assignment strategy overview

The first systematic approach to manually assign protein resonances and subsequently determine the protein 3-dimensional structure was proposed in the 1980s by Nobel prize winner Kurt Wüthrich and his research group [1]. This is sequential resonance assignment strategy that relies on two types of 2-dimensional nuclear magnetic resonance (2D NMR) experiments: correlated spectroscopy (COSY) and nuclear Overhausser effect spectroscopy (NOESY). In the first phase, the COSY experiment provides information about ^1H – ^1H through-bond (spin-spin) connectivities which can be used to identify amino acid spin systems [2]. Then, in the second phase, NOESY experiment is used to identify through-space (dipole-dipole) interactions of neighbor hydrogen atoms within 2-5 Å proximity from each other [3], linking neighboring spin systems together. Spin systems determined in the first stage can be assigned to specific residues in the protein sequence by linking to spin systems of its neighbors as determined in the second phase. This sequential approach was used during 1980s and allowed the assignment of proteins of up to ~10-15 kilodaltons (kD) (~80-120 residues). Development of ^{15}N-labeling methodologies during the late 1980s improved peak dispersion, enabling the assignment of larger protein molecules using the sequential assignment approach [4], [5].

1.1.2 Solution-state NMR triple resonance manual assignment strategy overview

In the early 1990s, advancements in ^{13}C and ^{15}N double labeling protein synthesis technologies led to the development of new strategies that use ^{1}H, ^{13}C, and ^{15}N magnetically active nuclei to design new 2D and 3D protein NMR experiments and assignments strategies [6], [7]. This triple-resonance strategy relies on set of NMR experiments that utilize through-bond (spin-spin) couplings to identify spin systems that belong to either single amino acid or dipeptide. Then the redundancy across multiple spectra is used to identify neighbor spin systems and link them together, which results in resonance assignments for the full protein chain.

In a typical triple resonance experiment, the backbone amide ^{1}H and ^{15}N resonance pair are used as common resonances across all spectra, i.e. they serve as root resonances for grouping peaks from multiple spectra into spin systems. This ^{1}H and ^{15}N double resonance is associated with one or more carbons which include backbone carbonyl ^{13}C, backbone ^{13}CA, or side-chain ^{13}C in order to assign backbone and side-chain resonances.

1.1.3 Solid-state NMR manual assignment strategy overview

In the early 2000s, researchers started to apply magic-angle spinning solid-state NMR (MAS SSNMR) to the problem of protein resonance assignment and structure determination. The first assignable spectral data were obtained around 2000 [8] and the first high-resolution structures of a peptide [9] and a protein [10] were obtained in 2002. Since that period, the field of MAS solid-state NMR has experienced rapid development.

Resonance assignment by solid-state NMR often requires uniform ^{13}C and ^{15}N double labeling of protein of interest. A typical assignment strategy uses experiments that utilize ^{13}C and ^{15}N resonances to group peaks from multiple spectra into spin systems. Depending on the chosen strategy, double or triple root resonances are used to create spin systems that associate ^{13}C and ^{15}N of residues i and $i - 1$. Although standard solid-state NMR strategies use ^{13}C and ^{15}N resonances, experiments that utilize ^{1}H-detection are being developed [11]–[13], which can improve the sensitivity of the spectrum and increase the number of experimental strategies to perform resonance assignment.

1.1.4 Automated protein resonance assignment overview

With the development of each new generation of NMR experiments, improved manual approaches for sequence site-specific protein resonance assignment would first develop, followed by the development of computational methodologies that would attempt to automate the manual assignment process. In the late 1980s and early 1990s, automated and semi-automated algorithms were developed to perform resonance assignment on homo-nuclear and then hetero-nuclear solution-state protein NMR spectra. Later, due to the advancements in solid-state protein NMR, the feasibility of automated protein resonance assignment was demonstrated in 2010. A more detailed discussion on automated assignment algorithms and methodologies is provided in the next chapter.

1.2 Motivation

With the advancements in sequencing technologies, genetic and protein sequence information became widely available with the ultimate goal to understand the function of gene-products, mostly protein biological function. However, a protein's biological function more directly depends on its 3-dimensional structure, which has spurred the continued development of methods for determining protein structure and related dynamics.

Two related methods for determining a protein's 3-dimensional structure and dynamics is solution-state and solid-state protein NMR. As of July 2017, these techniques contribute about 10 % (~10,300 solution-state NMR structures and ~100 solid-state NMR structures) of the structures deposited in Protein Data Bank [14]. But more importantly, NMR structure determination methods facilitate the study of classes of proteins not amenable to other structure determination techniques like x-ray crystallography and can observe and verify structural and dynamic characteristics that may not be detectable by x-ray crystallography. These reasons provide the motivation for the development of new computational methodologies that enable robust automated protein resonance assignment and subsequent structure determination, especially for the solid-state NMR technique.

The scope of this dissertation is focused on providing the survey of currently available automated resonance assignment approaches for both solution-state and solid-state protein NMR data and demonstrating results of new algorithms and software tools for implementing effective and robust automated protein resonance assignment.

1.3 Dissertation outline

Chapter 2 reviews the important biological applications of both solution-state and solid-state protein NMR. This chapter explains the problem of protein resonance assignment for solution-state and solid-state NMR from the algorithmic/computational point of view. Next it reviews currently available algorithms that are applied to the protein resonance assignment problem in both solution-state and solid-state protein NMR. Chapter 3 provides general design principles and the philosophy behind approach to the protein resonance assignment problem, along with the data structures and algorithms supporting this approach. Chapter 4 provides a description of the software package nmrstarlib which is designed to provide easy-to use access in order to utilize protein NMR data such as assigned chemical shifts and assigned experimental peak lists in the NMR-STAR format, especially publicly-available NMR-STAR formatted datasets in the Biological Magnetic Resonance Data Bank [15]. Chapter 5 describes the first critical step in resonance assignment algorithms, which is new single peak list registration and single peak list spin system grouping algorithms for peak lists that have multiple peaks per spin system, which are used to create initial local spin system groupings. Chapter 6 provides a description of the pairwise peak list registration and spin system grouping algorithms which globally merge spin system clusters from different peak lists, while detecting and correcting spin system overlap or spin system split in the initial spin system groupings. Chapter 7 is devoted to discussion and future directions of the whole analysis. Time and space complexity of the algorithms is discussed. Chapter 8 is devoted to project summary and conclusions.

CHAPTER 2
PROTEIN NMR AUTOMATED RESONANCE ASSIGNMENT BACKGROUND

2.1 Biology background

NMR spectroscopy is one of the essential analytical techniques that complements x-ray crystallography and electron microscopy in protein 3D structure determination. As of July 2017 NMR spectroscopy contributes 9% (10,404 structures out of 121,831 total structures available) of the protein 3D structures to the Protein Data Bank (PDB) [14].

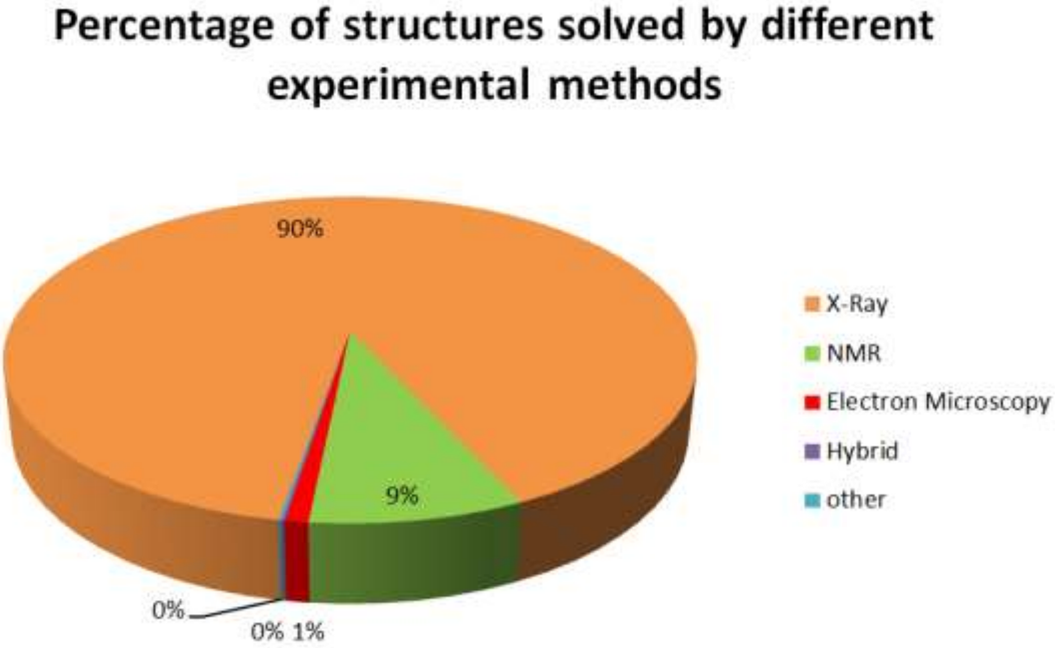


Figure 1. PDB statistics by experimental method used to determine 3D structure of proteins (as of July 2017).

Both solution-state and solid-state NMR spectroscopies contribute structures to the PDB. However, the number of structures determined by solution-state NMRis significantly larger than the number of structures determined by solid-state NMR spectroscopy. As of July 2017, the number of protein structures solved by solution-state NMR is ~10,400 and the number of structures solved by solid-state NMR is ~100 (**Figure 2**). The low numbers of solid-state NMR structures solved to date come from the challenges associated with obtaining good quality spectra for samples in the solid-state. But several advancements, which include improvements in spectrometer hardware, development of fast and ultra-fast magic-angle-spinning probes, and development of new experiments specific to solid-state NMR spectroscopy, are improving the resolution and overall quality of solid-state protein NMR spectra.

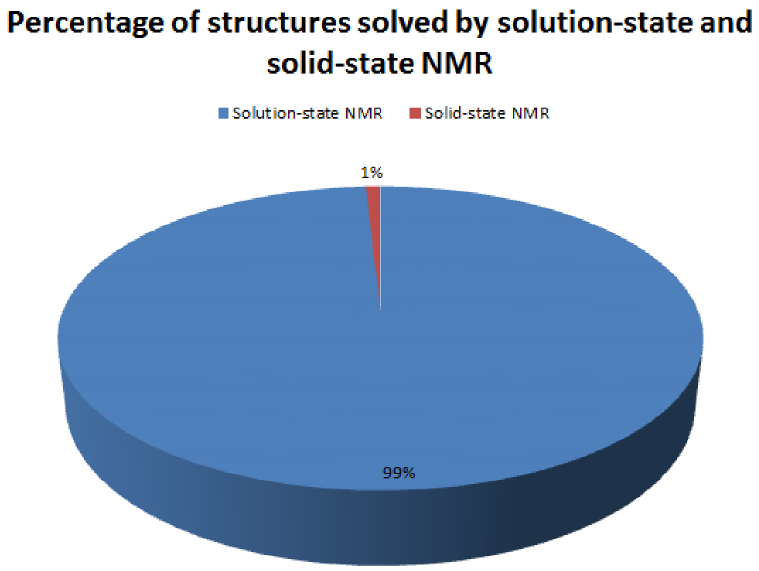


Figure 2. The percentage of structures determined by solution-state and solid-state NMR spectroscopy in PDB (as of July 2017).

There are several advantages that NMR spectroscopy provides in protein structure determination as compared to other methods: experiments are carried out in a native-like environment both in solution-state NMR and solid-state NMR; the ability to obtain unique information about protein dynamics; and there is no need to crystallize proteins into diffractable crystals. One of the big disadvantages of NMR spectroscopy is that structure determination is limited to relatively small proteins.

Also, NMR spectroscopy complements x-ray crystallography in the structure determination of membrane proteins, especially in cases where a given protein cannot be crystallized. In cases where membrane proteins cannot be solubilized for solution-state NMR investigations, proteins can by studied by solid-state NMR in the microcrystalline state. An estimated 20%-30% of all genes in most genomes encode membrane proteins [16]. Membrane proteins are one of the main protein classes besides fibrous proteins, globular proteins, and disordered proteins. Also, membrane proteins are directly associated with the membranes of a cell or organelle and have myriads of functions that are crucial to many fundamental biological processes of organisms [17]. Highlighting just a few, these functions include: transport of ions, metabolites, and larger molecules (proteins and RNA) across membranes; relaying signals between the internal and external environment of a cell; targeting enzymes to the specific locations in the cell; controlling the composition of the membrane bilayer; maintenance and organization of the shape of cells and organelles [18]; recognition and defense against invading pathogens; and maintenance of lipid energy supply [17]. Because of the important roles they play, malfunctioning membrane proteins can be causal agents in a large variety of diseases. For example, malfunctioning ion channels can cause neurological and cardiac diseases [19],

[20]. Color blindness is caused by nonfunctional photoreceptors [21]. Cystic fibrosis is caused by mutations that lead to the misfolding of a chloride transporter in the lung [22]. 3D structure and dynamics are needed in order to mechanistically understand how specific membrane proteins function in biological and disease processes and for structure-based (rational) drug design. Also, more than 50% of all current drug targets are membrane proteins [18], [23]. Unfortunately, only a relatively small number of these membrane protein 3D structures have been characterized. Membrane proteins account for less than 1% of the proteins with known 3D structure (~700 unique structures [24] out of ~121831 structure entities in the PDB [14]). Membrane proteins remain hard to study by traditional methods, because their structures depend on complex membrane environments [25].

Another class of proteins that are difficult to study by classical approaches is amyloid fibrils. Amyloid refers to the abnormal fibrous protein aggregates found in organs and tissues [26]. Medical interest in amyloid fibrils comes from their involvement in a variety of diseases such as Alzheimer’s disease, type II diabetes, Parkinson’s disease, and Huntington’s disease [27].

2.2 Description of the protein resonance assignment problem

2.2.1 Difference between solution-state and solid-state assignment strategies

Both solution-state and solid-state protein NMR resonance assignment strategies are conceptually very similar to each other. However, software tools and algorithms developed for solution-state NMR cannot be directly applied to the solid-state NMR peak lists due to the fact that solution-state and solid-state utilize different NMR experiments and, as a result, different resonances are used to organize peaks into spin systems. **Figure**

3 demonstrates differences between typical resonances used in grouping peaks across multiple different peak lists for solution-state (**Figure 3a**) and solid-state (**Figure 3b**) protein NMR, i.e. a typical solution-state NMR assignment strategy utilizes ^{1}H and ^{15}N resonances to organize peaks into spin systems versus a typical solid-state NMR assignment strategy that uses a combination of ^{13}CO, ^{15}N, and ^{13}CA resonances to assembly spin systems. However, in addition to ^{13}C and ^{15}N detection, new solid-state NMR experimental assignment strategies are being developed that utilize ^{1}H resonance detection [11]–[13]. **Table 1** and **Table 2** summarize experimental assignment strategies employed in solution-state and solid-state protein NMR respectively, color-coded according to categories described by **Figure 3a** and **Figure 3b**.

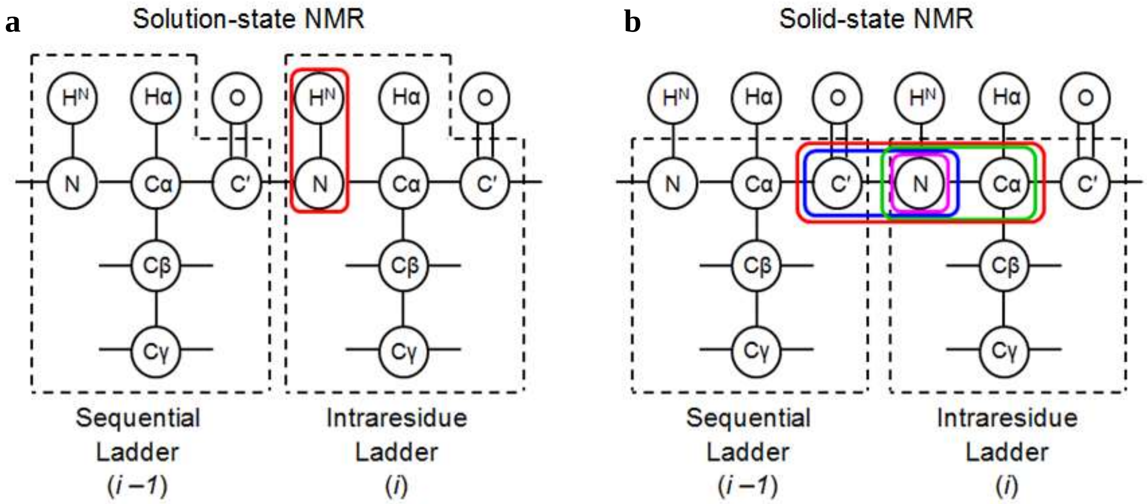


Figure 3. Standard dipeptide spin system definitions for protein resonance assignments in solution-state and solid-state NMR. Spin system root resonances are color coded: **a**) solution-state NMR assignment strategies based on ^{1}H and ^{15}N root definition found in all standard experiments used in spin system assembly; **b**) solid-state NMR is based on partial triple resonance root definition that utilizes ^{13}C and ^{15}N resonances and include one, two, or three resonances that are used in spin system assembly depending on the assignment strategy.

Table 1. Solution-state NMR experimental assignment strategies for protein resonance assignment.

Category II **($H_i–N_i$)**
$H_i–N_i$
$H_i–N_i–CO_{i-1}$
$H_i–N_i–(CA_i)–CO_i$
$H_i–N_i–(CA_{i-1})–CO_{i-1}$
$H_i–N_i–CA_i$
$H_i–N_i–CA_{i-1}$
$H_i–N_i–(CO_{i-1})–CA_{i-1}$
$H_i–N_i–(CO_{i-1})–CA_{i-1}CB_{i-1}$
$H_i–N_i–CA_{i-1}CB_{i-1}$
$H_i–N_i–CA_iCB_i$
$H_i–N_i–(CO_{i-1})–CA_{i-1}CB_{i-1}CG_{i-1}$
$H_i–N_i–(CO_{i-1}CA_{i-1}CB_{i-1}CG_{i-1})–HA_{i-1}HB_{i-1}HG_{i-1}$
$H_i–N_i–(CO_{i-1}CA_{i-1}CB_{i-1})–HA_{i-1}HB_{i-1}$

Table 2. Solid-state NMR experimental assignment strategies for protein resonance assignment.

Category I **(N_i)**	**Category IIa** **($CO_{i–1}–N_i$)**	**Category IIb** **($CA_i–N_i$)**	**Combined** **IIa and IIb**	**Category III** **($CO_{i–1}–N_i–CA_i$)**
$CA_i–N_i–CO_{i–1}$	$CO_{i–1}–N_i–CA_i$	$CA_i–N_i–CO_{i–1}$	$CA_i–N_i–CO_{i–1}$	$CA_i–N_i–CO_{i–1}–CX_{i–1}$
$N_i–CA_i–CX_i$	$CO_{i–1}–N_i–(CA_i)–CX_i$	$CA_i–N_i–(CO_{i–1})–CX_{i–1}$	$N_i–CO_{i–1}–CX_{i–1}$	$CO_{i–1}–N_i–CA_i–CX_i$
$N_i–CO_{i–1}–CX_{i–1}$	$N_i–CO_{i–1}–CX_{i-1}$	$N_i–CA_i–CX_i$	$N_i–CA_i–CX_i$	$CA_i–N_i–CO_{i–1}–CA_{i-1}$
$N_i–CO_{i–1}–CA_{i–1}$	$N_i–CO_{i–1}–CA_{i-1}$	$N_i–CA_i–CA_iCB_i$		$CO_{i–1}–N_i–CA_i–CB_i$
$N_i–CA_i–CA_iCB_i$	$CO_{i–1}–N_i–(CA_i)–CB_i$	$N_i–CA_i–CB_i$		$CO_{i–1}–N_i–CA_i–CA_iCB_i$
$N_i–CA_i–CB_i$	$CO_{i–1}–N_i–(CA_i)–CO_i$	$N_i–CA_i–CO_i$		$CO_{i–1}–N_i–CA_i–CO_i$
$N_i–CO_{i–1}–(CA_{i-1})–CA_{i-1}–CB_{i-1}$	$N_i–CO_{i–1}–(CA_{i-1})–CA_{i-1}–CB_{i-1}$	$CA_i–N_i–(CO_{i–1})–CA_{i-1}$		
$N_i–CO_{i–1}–(CA_{i-1})–CB_{i-1}$	$N_i–CO_{i–1}–(CA_{i-1})–CB_{i-1}$			
$N_i–CA_i–CO_i$				

2.2.2 Protein resonance assignment problem description

The protein resonance assignment problem can be represented as a bipartite graph: a graph whose vertices can be divided into two disjoint sets such that every edge connects a vertex in the first set to a vertex in the second set. One set is a collection of spin systems (SS) and the second ordered set represents the linear amino acid sequence (AA) of a protein. **Figure 4** demonstrates the general case of the protein resonance assignment problem as a bipartite graph.

The basic assignment problem is essentially the same mathematically for both solution-state NMR and solid-state NMR. Thus, a reliable common assignment strategy is implemented in the following basic steps: **1)** peak list registration – alignment of common dimensions between peak lists from different spectra; **2)** peak list quality assessment – evaluation of the quality of input peak lists; **3)** spin system grouping – grouping peaks from peak lists into spin systems using common root resonances; **4)** amino acid typing – classification of dipeptide spin systems by possible amino acid type using chemical shift values; **5)** linking – linking nearest-neighbor spin systems by matching sequential and intraresidue chemical shifts; **6)** mapping – mapping linked spin system segments uniquely to the primary sequence(s) of the protein; **7)** resonance assignment quality assessment – evaluation of the quality of the resulting resonance assignments.

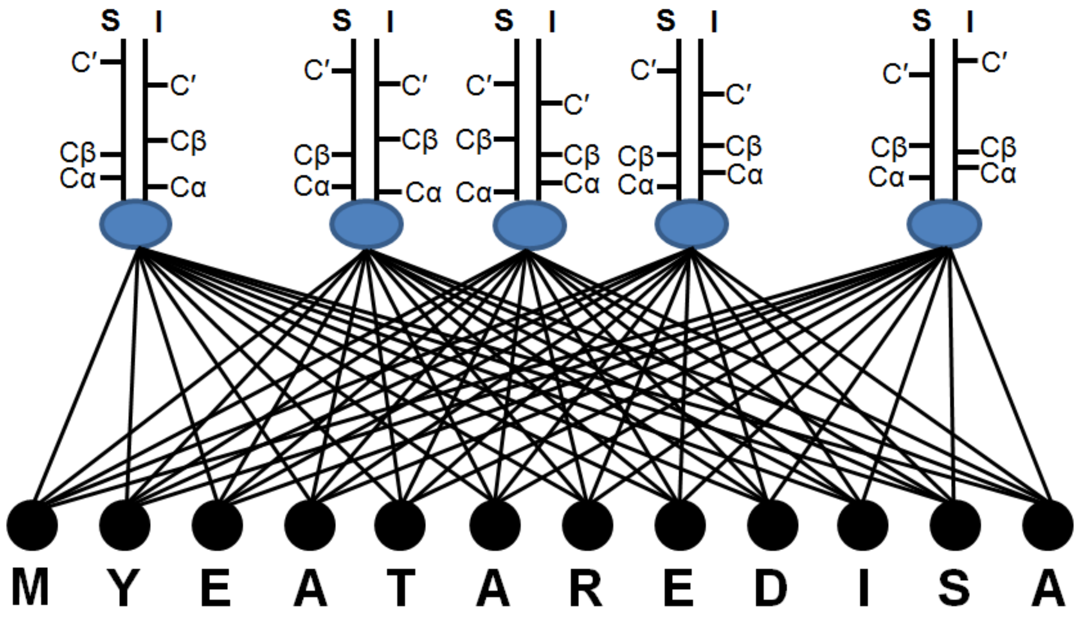


Figure 4. Bipartite graph representing the protein resonance assignment problem: black circles represent linear sequence of amino acids, where each letter is a single-letter amino acid code; blue ovals represent root resonances that were used to group peaks into spin systems; each spin systems has an intraresidue (I) and sequential (S) ladder associated with it; each ladder contains chemical shift values.

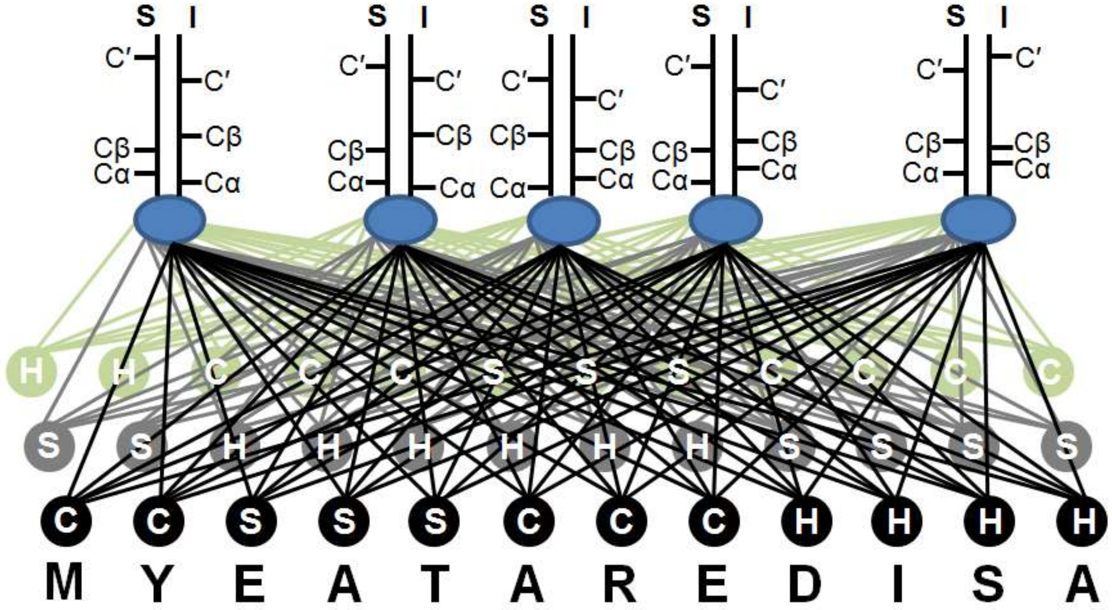


Figure 5. Multi-layered bipartite graph representing the protein resonance assignment problem with secondary structure information: black circles represent the linear sequence of amino acids, where each letter outside circle is a single-letter amino acid code; each letter inside circle designate secondary structure conformation (H – helix, S – strand, C – coil); blue ovals represent root resonances that were used to group peaks into spin systems; each spin systems has an intraresidue (I) and sequential (S) ladder associated with it; each ladder contains chemical shift values.

It is known that chemical shift values are secondary-structure-dependent [28]. Thus, the inclusion of secondary structure information transforms the bipartite representation into a multi-layered bipartite graph representation with additional layers of edges and nodes. **Figure 5** demonstrates the multi-layered bipartite graph representation where black edges between spin systems (blue ovals) and primary sequence (black circles) form the first layer bipartite graph, then gray edges between spin systems (blue ovals) and primary sequence (gray circles) form the second layer bipartite graph. Many layers are possible depending on the different secondary structure combinations the primary protein sequence can have.

In order to reduce the number of layers, prior information can be leveraged to predict secondary structure for specific parts of the primary sequence. The state of the art secondary structure prediction from sequence tools achieve very high prediction accuracy [29]. In addition, secondary structure information can be extracted from homologous protein structures generated by homology modeling tools.

The number of edges can be reduced by the prediction of the most probable amino acids for particular chemical shifts within spin systems (either root and/or ladder chemical shift values). **Figure 6** shows the protein resonance assignment problem where secondary structure information reduces the number of layers, and amino acid typing information reduces the number of edges. Leveraging redundancy in chemical shift values between intraresidue and sequential ladders spin systems can be linked together into a segment and then that segment can be mapped into a protein sequence.

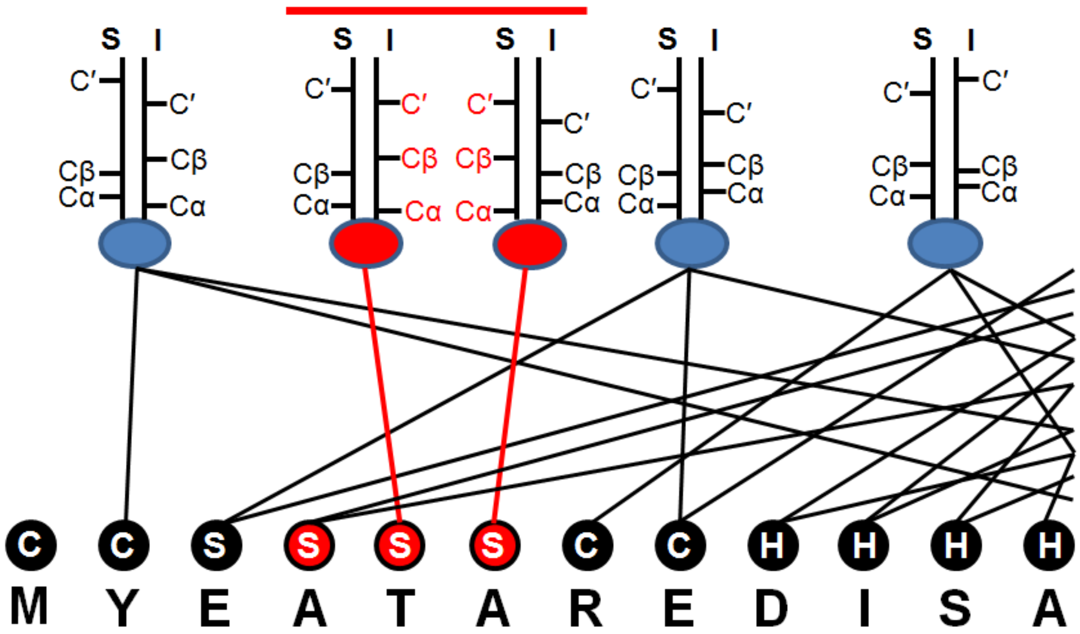


Figure 6. Secondary structure prediction information limits the number of layers; amino acid typing limits the number of edges between spin systems and primary amino acid sequence; red chemical shift values identify spin system linking; red edges represent spin system mapping into the amino acid sequence.

2.2.3 Example resonance assignment strategy using a set of solid-state NMR peak lists

Figure 7 illustrates a combined Category IIa and IIb assignment strategy that utilizes three different peak lists derived from solid-state NMR experiments. The 3D NCOCX peaks are composed of chemical shift values that belong to ^{15}N of residue i, ^{13}CO of residue $i-1$, and ^{13}CX of residue $i-1$ [30], with the resulting NCOCX peak list containing multiple peaks per spin system due to ^{13}CX (any carbon) dimension. The 3D CANCO peaks have chemical shift values from ^{13}CA and ^{15}N of residue i, and ^{13}CO of residue $i-1$ [31], with the resulting CANCO peak list containing a single peak per spin system. The 3D NCACX peaks contain chemical shift values from ^{15}N, ^{13}CA, and ^{13}CX of the same residue i [30], with the resulting NCACX peak list containing multiple peaks per spin system due to ^{13}CX dimension. The ^{15}N of residue i and ^{13}CO of residue $i-1$ chemical shift values can be used to group peaks into spin systems between the NCOCX and CANCO peak lists, then the ^{15}N and ^{13}CA of residue i chemical shift values can be used to group peaks into spin systems between the NCACX and CANCO peak lists. Together, both groupings form global spin systems across all three peak lists.

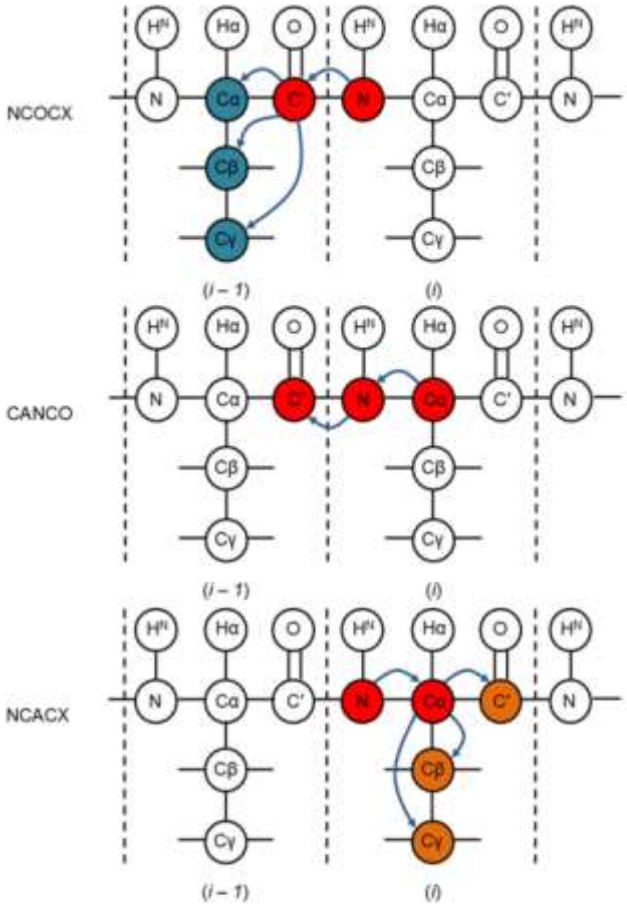
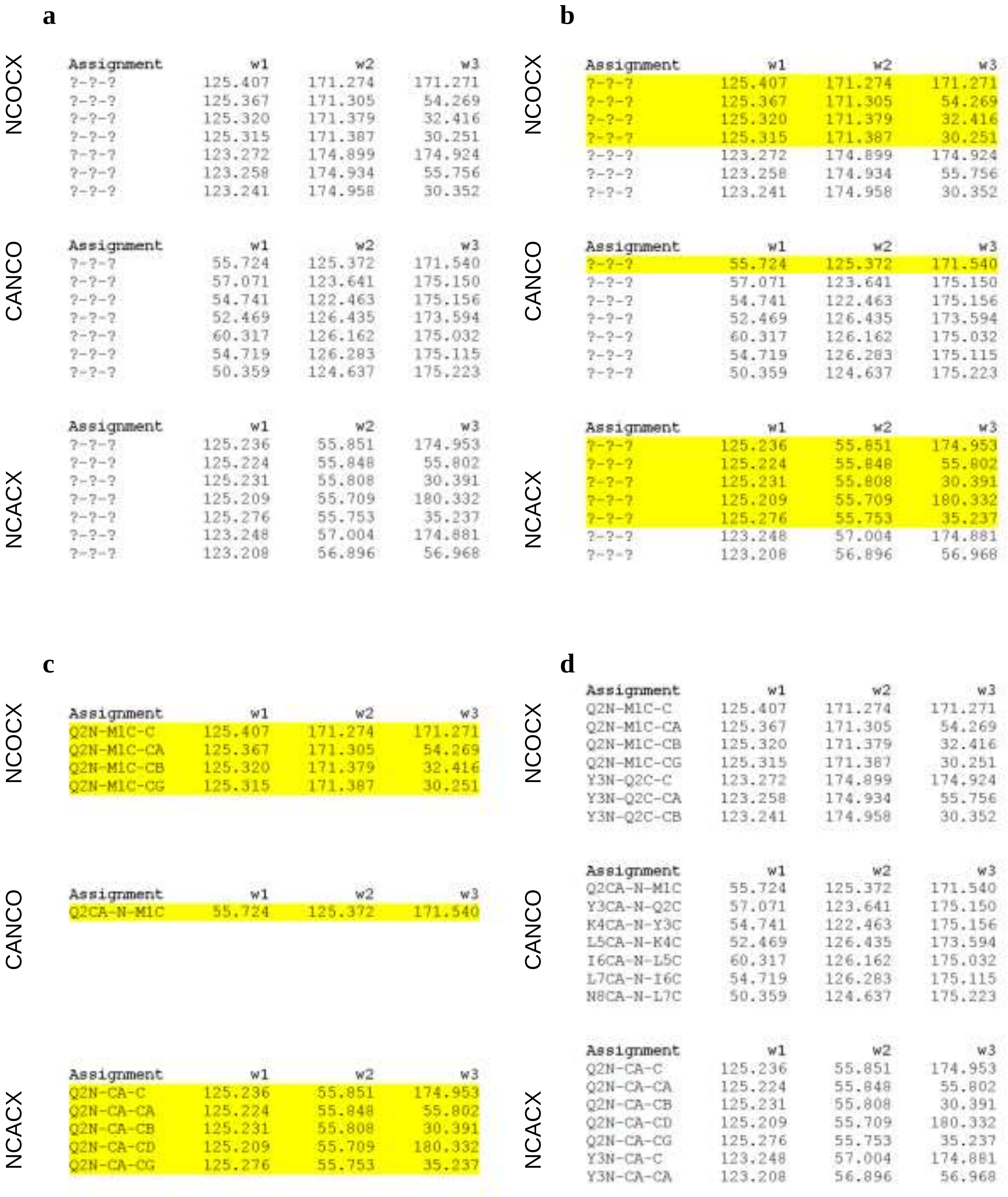


Figure 7. Example of solid-state NMR assignment strategy based on NCOCX, CANCO, and NCACX experiments.



a

NCOCX

Assignment	w1	w2	w3
?-?-?	125.407	171.274	171.271
?-?-?	125.367	171.305	54.269
?-?-?	125.320	171.379	32.416
?-?-?	125.315	171.387	30.251
?-?-?	123.272	174.899	174.924
?-?-?	123.258	174.934	55.756
?-?-?	123.241	174.958	30.352

CANCO

Assignment	w1	w2	w3
?-?-?	55.724	125.372	171.540
?-?-?	57.071	123.641	175.150
?-?-?	54.741	122.463	175.156
?-?-?	52.469	126.435	173.594
?-?-?	60.317	126.162	175.032
?-?-?	54.719	126.283	175.115
?-?-?	50.359	124.637	175.223

NCACX

Assignment	w1	w2	w3
?-?-?	125.236	55.851	174.953
?-?-?	125.224	55.848	55.802
?-?-?	125.231	55.808	30.391
?-?-?	125.209	55.709	180.332
?-?-?	125.276	55.753	35.237
?-?-?	123.248	57.004	174.881
?-?-?	123.208	56.896	56.968

b

NCOCX

Assignment	w1	w2	w3
?-?-?	125.407	171.274	171.271
?-?-?	125.367	171.305	54.269
?-?-?	125.320	171.379	32.416
?-?-?	125.315	171.387	30.251
?-?-?	123.272	174.899	174.924
?-?-?	123.258	174.934	55.756
?-?-?	123.241	174.958	30.352

CANCO

Assignment	w1	w2	w3
?-?-?	55.724	125.372	171.540
?-?-?	57.071	123.641	175.150
?-?-?	54.741	122.463	175.156
?-?-?	52.469	126.435	173.594
?-?-?	60.317	126.162	175.032
?-?-?	54.719	126.283	175.115
?-?-?	50.359	124.637	175.223

NCACX

Assignment	w1	w2	w3
?-?-?	125.236	55.851	174.953
?-?-?	125.224	55.848	55.802
?-?-?	125.231	55.808	30.391
?-?-?	125.209	55.709	180.332
?-?-?	125.276	55.753	35.237
?-?-?	123.248	57.004	174.881
?-?-?	123.208	56.896	56.968

c

NCOCX

Assignment	w1	w2	w3
Q2N-M1C-C	125.407	171.274	171.271
Q2N-M1C-CA	125.367	171.305	54.269
Q2N-M1C-CB	125.320	171.379	32.416
Q2N-M1C-CG	125.315	171.387	30.251

CANCO

Assignment	w1	w2	w3
Q2CA-N-M1C	55.724	125.372	171.540

NCACX

Assignment	w1	w2	w3
Q2N-CA-C	125.236	55.851	174.953
Q2N-CA-CA	125.224	55.848	55.802
Q2N-CA-CB	125.231	55.808	30.391
Q2N-CA-CD	125.209	55.709	180.332
Q2N-CA-CG	125.276	55.753	35.237

d

NCOCX

Assignment	w1	w2	w3
Q2N-M1C-C	125.407	171.274	171.271
Q2N-M1C-CA	125.367	171.305	54.269
Q2N-M1C-CB	125.320	171.379	32.416
Q2N-M1C-CG	125.315	171.387	30.251
Y3N-Q2C-C	123.272	174.899	174.924
Y3N-Q2C-CA	123.258	174.934	55.756
Y3N-Q2C-CB	123.241	174.958	30.352

CANCO

Assignment	w1	w2	w3
Q2CA-N-M1C	55.724	125.372	171.540
Y3CA-N-Q2C	57.071	123.641	175.150
K4CA-N-Y3C	54.741	122.463	175.156
L5CA-N-K4C	52.469	126.435	173.594
I6CA-N-L5C	60.317	126.162	175.032
L7CA-N-I6C	54.719	126.283	175.115
N8CA-N-L7C	50.359	124.637	175.223

NCACX

Assignment	w1	w2	w3
Q2N-CA-C	125.236	55.851	174.953
Q2N-CA-CA	125.224	55.848	55.802
Q2N-CA-CB	125.231	55.808	30.391
Q2N-CA-CD	125.209	55.709	180.332
Q2N-CA-CG	125.276	55.753	35.237
Y3N-CA-C	123.248	57.004	174.881
Y3N-CA-CA	123.208	56.896	56.968

Figure 8. NCACX, CANCO, and NCOCX peak lists during the assignment process: **a**) unassigned peak lists; **b**) peaks that belong to the same spin system within single peak list as well as across different peak lists are identified; **c**) peaks that belong to the same spin system are isolated, grouped, and assigned; **d**) completely assigned peak lists.

The identified spin systems are used to calculate the list of most probable amino acids for each of the spin system's ladders. Next, linking and mapping algorithms must be applied in order to uniquely assign the spin systems to the protein sequence.

Figure 8 demonstrates the assignment strategy illustrated in **Figure 7** in terms of peak lists: **Figure 8a** shows the unassigned NCOCX, CANCO, and NCACX peak lists. In **Figure 8b**, the groups of peaks within and across peaks lists are identified. **Figure 8c** shows isolated spin system group that has been typed and assigned. **Figure 8d** demonstrates completely assigned peak lists after each of the spin system groups are typed, linked, and uniquely mapped to the protein sequence.

2.3 Currently available automated assignment tools

2.3.1 Tools for automated assignment of solution-state NMR data

This section provides an overview of the automated protein resonance assignment software tools and algorithms for the solution-state NMR. **Table 3** shows a non-exhaustive list of solution-state NMR tools published in the last 20 years or so. There are several major computational methods and approaches that are employed to address the automated resonance assignment problem: Monte Carlo/simulated annealing methods, evolutionary algorithms, exhaustive search, best-first heuristic and tree search approaches.

Monte Carlo/simulated annealing methods [32], [33], [34], [35], [36] try to explore the landscape of all possible solutions and optimize the pseudo energy function in order to identify the global optimal resonance assignments. Genetic algorithm [37], [38], [39] approaches are related to Monte Carlo methods and try to identify the optimal resonance assignments through evolution of set of initial random individual solutions.

Table 3. Programs for automated resonance assignment of solution-state NMR data.

Year published	Program name	Core methodology	Grouping	Registration
1997	Buchler *et al* [32]	Monte Carlo / simulated annealing	Yes	No
1997	AUTOASSIGN [40], [41]	Heuristic best-first	Yes	Yes
1997	GARANT [37]	Genetic algorithm	Yes	No
1997	Li et al [42]	Heuristic best-first	Yes	No
1997	Lukin *et al* [33]	Monte Carlo / simulated annealing	Yes	No
1998	CAMRA [43]	Matching predicted shifts with observed spin systems	Yes	No
1998	PASTA [34]	Monte Carlo / simulated annealing	Yes	No
2000	MAPPER [44]	Exhaustive search	Yes	No
2000	TATAPRO [45]	Exhaustive search	Yes	No
2002	Andrec *et al* [46]	Exhaustive search	Yes	No
2003	Reed et al [35]	Monte Carlo / simulated annealing	Yes	No
2003	IBIS [47]	Heuristic best-first	Yes	No
2003	MONTE [36]	Monte Carlo / simulated annealing	Yes	No
2003	PACES [48]	Exhaustive search	Yes	No
2004	MARS [49]	Heuristic best-first	Yes	No
2005	CASA [50]	Depth-first tree search	Yes	No
2005	PISTACHIO [51]	Probabilistic identification of spin systems and their assignments	Yes	No
2007	CISA [52]	Connectivity graph	Yes	No
2007	GASA [53]	Connectivity graph	Yes	No
2008	MATCH [38]	Genetic algorithm	Yes	No
2009	IPASS [54]	Integer linear programming	Yes	No
2010	SAGA [55]	Depth-first tree search	Yes	No
2012	FLYA [39]	Genetic algorithm	Yes	Yes
2013	EZ-ASSIGN [56]	Exhaustive search	Yes	No

The optimal solution is deduced through multiple cycles of mutation and recombination. Global and local optimization schemas might be used to guide the resonance assignment to the global optimum. Heuristic best-first approaches [40], [41], [42], [47], [49] try to identify the set of initial best unambiguous (complete, non-overlapped) segments of spin systems and assign them first, then try to assign the more ambiguous (overlapped, incomplete) spin systems next.

2.3.2 Tools for automated assignment of solid-state NMR data

This section provides an overview of the automated protein resonance assignment software tools and approaches published recently that are designed specifically to handle the task of protein resonance assignment of peak lists derived from solid-state NMR experiments. The number of programs designed for solid-state NMR automated assignment is significantly smaller than the number of solution-state NMR automated assignment tools. **Table 4** shows a list of programs designed to perform automated protein resonance assignment of solid-state NMR data.

Table 4. Programs for automated resonance assignment of solid-state NMR data.

Year published	Program name	Core methodology	Grouping	Registration
2010	MC_ASSIGN1 [57]	Monte Carlo/ simulated annealing	Yes	No
2010	SASS [58]	Heuristic best-first	Yes	Yes
2013	ssFLYA [59]	Genetic algorithm	Yes	Yes
2014	GAMES_ASSIGN [60]	Genetic algorithm	Yes	No

In 2010, one of the first programs that demonstrated feasibility of automated protein resonance assignment on solid-state NMR peak lists was the MC_ASSIGN1[57]. The MC_ASSIGN1 algorithm uses protein sequence and a limited set of 2D peak lists, N(CA)CX and N(CO)CX, in order to generate sequential resonance assignment. The approach is based on a Monte Carlo/simulated annealing computational algorithm. The algorithm tries to assign each peak within N(CA)CX and N(CO)CX to every residue within protein sequence using global optimization score function. During the algorithm execution, this score function tries to maximize the number of “good connections” and minimize the number of “bad connections”, number of “edges”, and number of unused peaks. The MC_ASSIGN1 algorithm was tested against uniformly labeled HET-s(218–289) fibrils with known manual assignments.

Another early approach that demonstrated tractability of automated protein resonance assignment using solid-state NMR peak list was the SASS software [58]. The program also used a limited set of solid-state NMR experiments, 3D NCACX, 3D CAN(CO)CA, and 4D CANCOCX in order to produce the resonance assignments of 56 amino acids long GB1 protein with known manual assignments. The design of the program is similar to the solution NMR assignment package AutoAssign [40], [61]–[63]. It implements the prototype grouping, and typing algorithms, but uses linking and mapping algorithms from the AutoAssign. The prototype program was able to achieve 84.1% assignment of the ^{15}N, ^{13}CO, ^{13}CA, and ^{13}CB resonances with no errors. Both MC_ASSIGN1 and SASS programs represented a proof of concept software tools that demonstrate the tractability of the automated protein resonance assignment problem.

Later, in 2013, the algorithm called ssFLYA was developed within the automated resonance assignment and structure calculation program CYANA [59]. This approach was developed on the basis of FLYA algorithm for automated assignment of solution-state NMR peak lists within same CYANA software [39]. The ssFLYA algorithm is able to handle more standard 2D and 3D solid-state NMR peak lists such as 3D NCACB, 3D CAN(CO)CA, 3D CANCO, 3D NCACO, 3D NCACX, 3D NCOCA, 3D NCOCX, 2D NCO, and 2D NCA. The resonance assignment solutions are generated by comparing the set of measured peaks with known positions to the set of expected assigned peaks with unknown positions. The resonance assignment process relies on a global evolutionary optimization algorithm and local optimization routine that takes back and tries to reassign small parts of the generated assignment. Initial set of solutions is generated randomly. The recombination procedure is used to generate a new generation of resonance assignment solutions from the previous generation. The scoring function is used in order to select the best resonance assignment solutions and the solution that maximizes the scoring function is then reported as the final solution. The algorithm was applied to peak lists from four different proteins: microcrystals of ubiquitin and Ure2 prion C-terminal domain amyloids of HET-s(218–289) and α-synuclein.

In 2014, the algorithm called GAMES_ASSIGN (Genetic Algorithm using Maximum Entropy for Solid state NMR resonance ASSIGNments of proteins) was published. This algorithm uses standard solid-state NMR experiments such as 3D NCACX, 3D NCACO, 3D NCOCX, 3D NCOCA, 3D CONCA. The algorithm proceeds in three phases. In the first phase, spin systems are generated by pairing peaks one by one. In the second phase, resonance assignments are generated by pairing generated spin systems to the specific positions within protein sequence. Both the first and second phases are repeated a number of times in order to generate the set of candidate resonance assignment solutions. A genetic algorithm with mutation and recombination is applied at these phases in order to guide the creation best candidate solutions. Statistics are generated during the first two phases, including how many times a certain peak was assigned to a particular position within the protein sequence. In the third phase, the final consensus resonance assignments are generated utilizing the statistics information obtained in the first and second phases. Peak lists from three different proteins were used to evaluate the performance: GB1, ubiquitin, and CsmA.

CHAPTER 3
PROJECT DESIGN OVERVIEW

3.1 Overview

This chapter provides a high-level overview of the algorithms and data structures necessary to model and solve the protein resonance assignment problem. In order to model the overall protein resonance assignment problem, the Unified Modeling Language (UML) was used to describe which algorithms and objects are necessary to implement.

3.2 Modeling of the protein resonance assignment problem using UML

3.2.1 Design of core entities

Several groups of entities that handle different aspects of the protein resonance assignment problem are defined (**Figure 9**). `Processing/Management Entities` are responsible for parsing peak lists, parsing protein sequences, and creating supporting entities. `Physical Entities` are designed to represent real objects such as a peak list, peak, protein sequence, etc. `Descriptive Entities` model the objects that are necessary for the description of the protein resonance assignment problem, such as a list of expected resonance values for each specific monomer, a description of monomers, a description of spectra, a description of resonance classes, etc. `Characterized Entities` are objects that combine prior information such as secondary structure prediction and homology modeling with descriptive entities.

Potential Assignment Entities enable the mapping of spin systems to Characterized Entities representing sequence sites in the protein sequence(s). Algorithmic Entities are classes and methods that directly solve the protein resonance assignment problem.

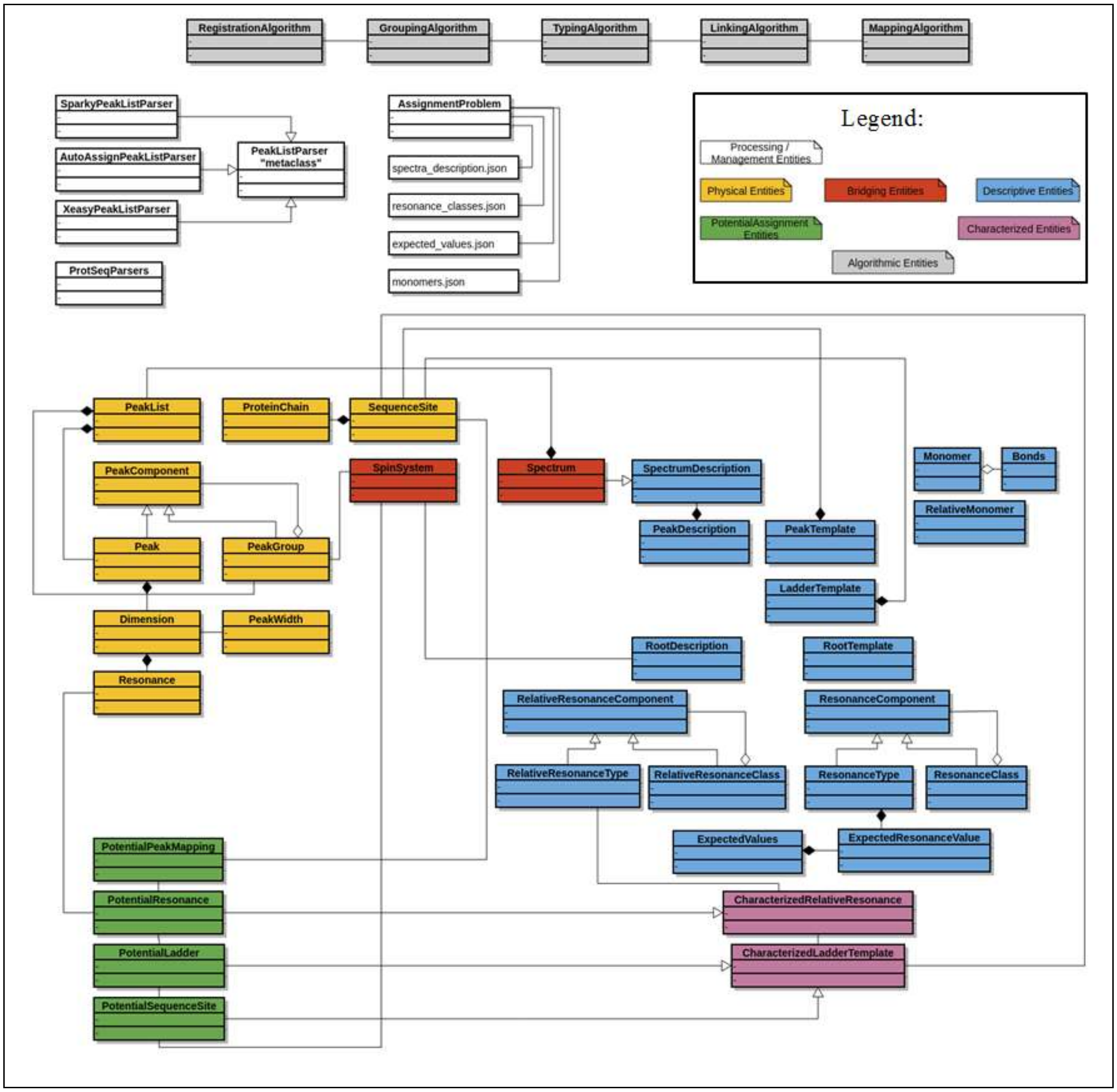


Figure 9. UML class dependency diagram that represents the overall design of data structures and algorithms for the automated protein resonance assignment.

Entities in **Figure 9** are connected using different types of connectors: a simple line represents weak association between entities; a line with filled diamond represents composition relationships between entities, when object of one type is composed of object of another type and cannot exist independently; a line with empty diamond represents aggregation relationships between entities, when object of one type is composed (weaker composition) of object of another type, but they can exist independently; a line with an arrow represents inheritance relationships between entities, when one object extends the functionality of another object.

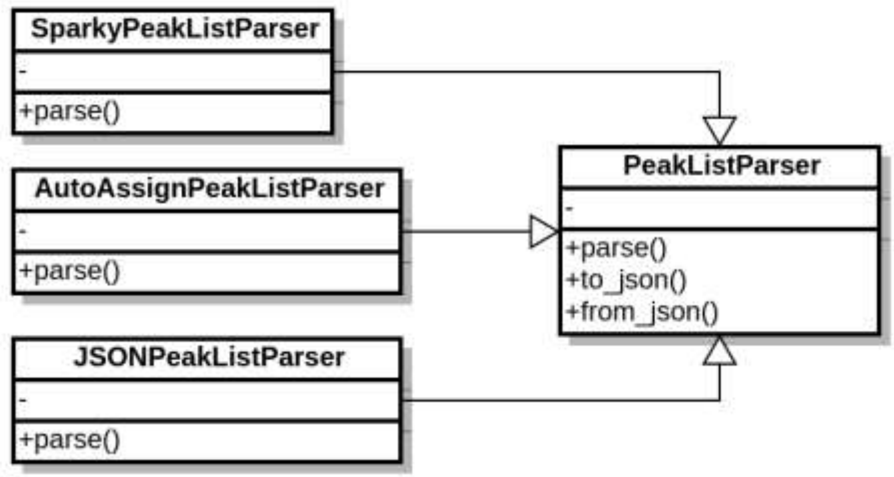


Figure 10. UML class dependency diagram of `PeakListParser` objects representing inheritance relationships.

Figure 10 shows class dependency diagram for entities responsible for parsing experimental peak lists into a `PeakList` object. Here three concrete peak list parsers `SparkyPeakListParser`, `AutoAssignPeakListParser`, and `JSONPeakListParser` inherit from an abstract `PeakListParser` object. This design provides an abstract common peak list parsing interface and each of the concrete peak lists parsers implement their own specific `parse()` method to address the specific parsing requirements of each peak list format. Additional peak list parsers can be easily defined by subclassing the abstract `PeakListParser` object.

Figure 11 shows the class dependency diagram for the `PeakFilter` objects that is not shown on the main diagram. Here an abstract `PeakFilter` object provides a common interface that specifies a list of parameters (`filter_parameters`) for the concrete peak filters. Currently, only `ChemShiftPeakFilter` is applied to filter out artefact peaks that are present within experimental peak lists, using minimum and maximum chemical shift ranges for ^{13}C, ^{15}N, and ^{1}H dimensions. In addition, an abstract `PeakFilter` class has the `filterlist` static method that operates on a peak list and uses list of specified filters to filter out unwanted peaks. Additional peak filters can be specified by subclassing `PeakFilter` class, for example, peak filters based on peak intensity or line width.

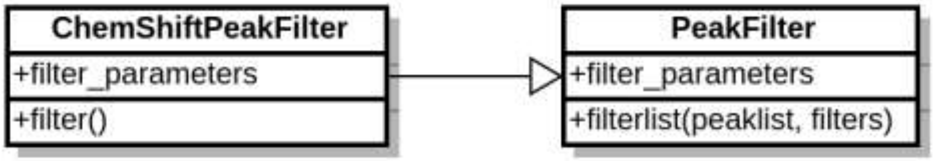


Figure 11. UML class dependency diagram of `PeakFilter` objects representing inheritance relationships.

Figure 12 shows an `AssignmentProblem` entity representing an entry point that uses different configuration files in order to facilitate creation of the other entities. It mostly consists of static methods that orchestrate the creation of all other entities, therefore it has weak association relationships with all entities it creates.

The composite design pattern is used in several places in the implementation, allowing us to treat complex objects the same way as a primitive object. Here the

PeakComponent represents an abstract class that provides a common interface for both the Peak and PeakGroup objects. Each Peak object represents a simple individual peak within the peak list. The PeakGroup object represents a more complex entity that can consists of multiple peaks, hence the PeakGroup name. The key idea is that groups of peaks can be manipulated in exactly the same way as each individual peak.

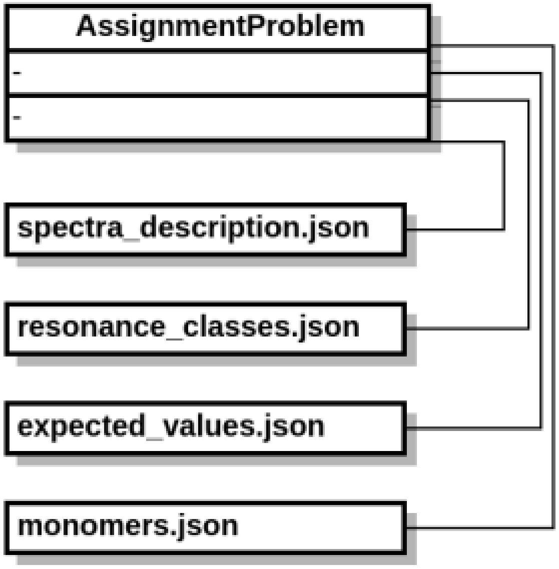


Figure 12. UML diagram of AssignmentProblem entity representing weak association relationships.

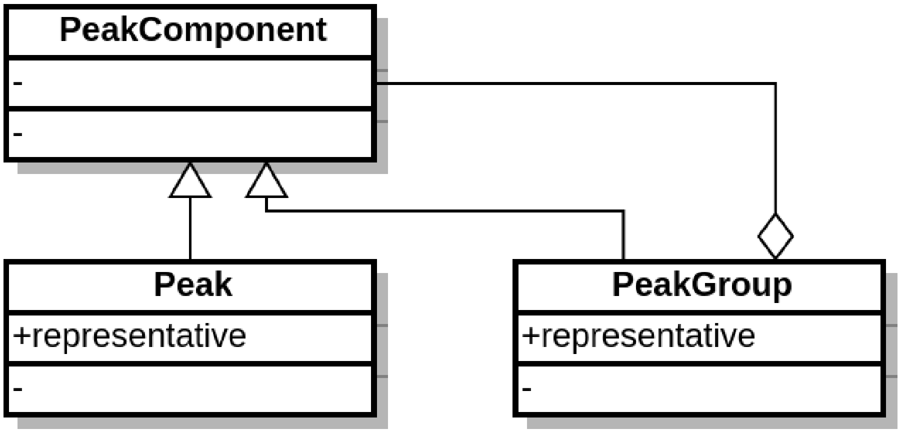


Figure 13. UML diagram of composite design pattern.

This concept is applied during the single and pairwise peak lists registration and grouping algorithms (Chapter 5 and 6) in order to create peak lists that consist of groups of peaks that belong to the same spin systems (local spin system groups) instead of individual peaks. This allow the algorithm to build global spin systems via pairwise comparison through merging peak groups and treat those peak groups as if they were individual peaks. **Figure 14** demonstrates an example of the resulting tree structure, where peak groups can consist of individual peaks (groups #1, #2, and #3) as well as mixture of peak groups and individual peaks (groups #4 and #5). In this context, both peaks and peak groups are treated as if they were same type of object.

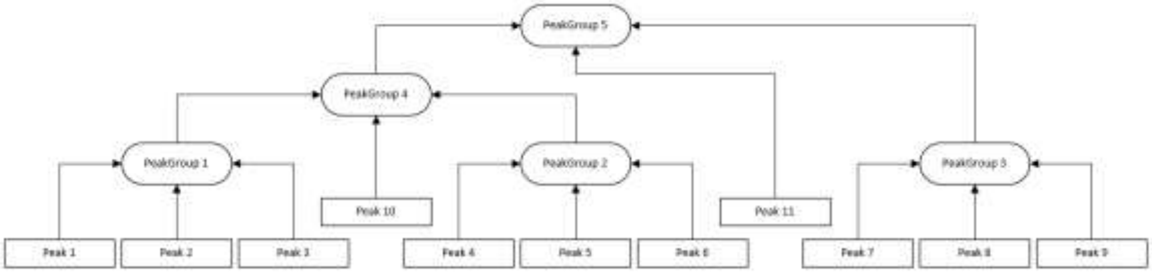


Figure 14. Example of tree structure that can be built with composite design pattern.

Figure 15 shows an example of the composition relationships where a `Peak` object is composed of multiple `Dimension` objects and each dimension has its corresponding `Resonance` object, and this group of objects cannot exist independently of each other.

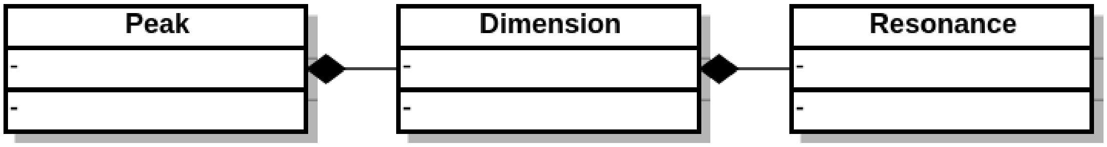


Figure 15. UML diagram of `Peak`, `Dimension`, and `Resonance` entities representing weak composite relationships.

3.2.2 Design of configuration files

To facilitate the creation of required entities, several configuration files using the JSON file format were developed. The use of JSON file format is a very important implementation decision: i) there are JSON parsers in every major computer language, facilitating future integration of this project with other major NMR software, ii) JSON is more human-readable than XML, and iii) all configuration files will have the same well-known base JSON file format, making them easier to understand and to maintain, iv) the use of human-readable and editable configuration files allows us to isolate important aspects of the protein resonance assignment problem such as description of the spectra from the actual code implementation, allowing the creation of generic algorithms that are capable of working on both solution-state and solid-state protein resonance assignment data.

The `spectra_description.json` file stores information about peak descriptions for different types of NMR experiments, enabling the easy incorporation of future solution-state and solid-state NMR experiments. **Figure 16** shows an example of spectral descriptions for the solid-state NMR experiments.

The `resonance_classes.json` configuration file stores all available individual as well as composite resonances that are available for every monomer. This configuration file allows the representation of each peak description in terms of generic resonance classes rather than amino acid specific resonance types. **Figure 17** shows an example configuration file that describes resonance classes.

```
{
    "NCA": {
        "Labels": ["N", "CA"],
        "MinNumberPeaksPerSpinSystem": 1,
        "PeakDescriptions": [
            {"fraction": 1, "dimensions": ["N", "CA"]}
        ]
    },

    "NCO": {
        "Labels": ["N", "CO-1"],
        "MinNumberPeaksPerSpinSystem": 1,
        "PeakDescriptions": [
            {"fraction": 1, "dimensions": ["N", "CO-1"]}
        ]
    },

    "NCACX": {
        "Labels": ["N", "CA", "CX"],
        "MinNumberPeaksPerSpinSystem": 2,
        "PeakDescriptions": [
            {"fraction": 1, "dimensions": ["N", "CA", "CO"]},
            {"fraction": 1, "dimensions": ["N", "CA", "CA"]},
            {"fraction": 1, "dimensions": ["N", "CA", "CB"]},
            {"fraction": 1, "dimensions": ["N", "CA", "CG"]},
            {"fraction": 1, "dimensions": ["N", "CA", "CD"]},
            {"fraction": 1, "dimensions": ["N", "CA", "CE"]},
            {"fraction": 1, "dimensions": ["N", "CA", "CZ"]}
        ]
    },

    "NCOCX": {
        "Labels": ["N", "CO-1", "CX-1"],
        "MinNumberPeaksPerSpinSystem": 2,
        "PeakDescriptions": [
            {"fraction": 1, "dimensions": ["N", "CO-1", "CA-1"]},
            {"fraction": 1, "dimensions": ["N", "CO-1", "CB-1"]},
            {"fraction": 1, "dimensions": ["N", "CO-1", "CG-1"]},
            {"fraction": 1, "dimensions": ["N", "CO-1", "CD-1"]},
            {"fraction": 1, "dimensions": ["N", "CO-1", "CE-1"]},
            {"fraction": 1, "dimensions": ["N", "CO-1", "CZ-1"]}
        ]
    },

    "CANCO": {
        "Labels": ["CA", "N", "CO-1"],
        "MinNumberPeaksPerSpinSystem": 1,
        "PeakDescriptions": [
            {"fraction": 1, "dimensions": ["CA", "N", "CO-1"]}
        ]
    },

    "CANCOCX": {
        "Labels": ["CA", "N", "CO-1", "CX-1"],
        "MinNumberPeaksPerSpinSystem": 2,
        "PeakDescriptions": [
            {"fraction": 1, "dimensions": ["CA", "N", "CO-1", "CO-1"]},
            {"fraction": 1, "dimensions": ["CA", "N", "CO-1", "CA-1"]},
            {"fraction": 1, "dimensions": ["CA", "N", "CO-1", "CB-1"]},
            {"fraction": 1, "dimensions": ["CA", "N", "CO-1", "CG-1"]},
            {"fraction": 1, "dimensions": ["CA", "N", "CO-1", "CD-1"]},
            {"fraction": 1, "dimensions": ["CA", "N", "CO-1", "CE-1"]},
            {"fraction": 1, "dimensions": ["CA", "N", "CO-1", "CZ-1"]}
        ]
    }
}
```

Figure 16. Example of spectra description file for the solid-state NMR experiments.

```
{
    "CA": ["CA"],
    "CB": ["CB"],
    "CG": ["CG", "CG1", "CG2"],
    "CD": ["CD", "CD1", "CD2"],
    "CE": ["CE", "CE1", "CE2", "CE3"],
    "CZ": ["CZ", "CZ2", "CZ3"],
    "CO": ["C"],
    "C":  ["C"],
    "CX": ["CA", "CB", "CG", "CG1", "CG2", "CD", "CD1", "CD2", "CE",
           "CE1", "CE2", "CE3", "CZ", "CZ2", "CZ3", "C"],

    "H":  ["H"],
    "HN": ["H"],
    "HA": ["HA", "HA2", "HA3"],
    "HB": ["HB", "HB2", "HB3"],
    "HG": ["HG", "HG1", "HG12", "HG13", "HG2", "HG3"],
    "HD": ["HD1", "HD2", "HD21", "HD22", "HD3"],
    "HE": ["HE", "HE1", "HE2", "HE21", "HE22", "HE3"],
    "HH": ["HH", "HH11", "HH12", "HH2", "HH21", "HH22"],
    "HZ": ["HZ", "HZ2", "HZ3"],

    "N":  ["N"],
    "ND": ["ND1", "ND2"],
    "NE": ["NE", "NE1", "NE2"],
    "NZ": ["NZ"]
}
```

Figure 17. Example of resonance classes configuration file.

```
    "Helix": {
        "A": {
            "CA": {
                "ExpectedChemShift": 54.83,
                "Stdev": 1.05
            },
            "CB": {
                "ExpectedChemShift": 18.26,
                "Stdev": 0.88
            },
            "CO": {
                "ExpectedChemShift": 179.4,
                "Stdev": 1.32
            },
            "HA": {
                "ExpectedChemShift": 4.03,
                "Stdev": 0.33
            },
            "HB": {
                "ExpectedChemShift": 1.35,
                "Stdev": 0.29
            },
            "HN": {
                "ExpectedChemShift": 8.08,
                "Stdev": 0.52
            },
            "N": {
                "ExpectedChemShift": 121.44,
                "Stdev": 2.37
            }
        }, ...
    ...
```

Figure 18. Example of expected values configuration file.

The `expected_values.json` configuration file contains information about expected chemical shift and standard deviation statistics. Expected values for ^{13}Cα, ^{13}Cβ, ^{13}CO, ^{15}N, ^{1}H, and ^{1}Hα were derived from RefDB [64], statistics for ^{13}Cγ, ^{13}Cδ, ^{13}Cε, ^{1}Hβ, ^{1}Hγ, ^{1}Hδ, ^{1}Hε, and others were derived from BMRB [15]. An example configuration file with information about expected chemical shift values for different secondary structures is shown on **Figure 18**.

3.2.3 Description of algorithms

3.2.3.1 Peak list registration algorithm

The peak list registration algorithm provides necessary registration offsets and peak list quality statistics necessary to group peaks into spin systems. The peak list registration algorithm executes in two modes: i) self-registration for a single peak list that contains multiple peaks per spin system; ii) pairwise-registration for two different peak lists. This is one of the most computationally expensive steps in implementation. Due to this fact, it is implemented as a stand-alone C++ program to improve efficiency of the algorithm. This alignment algorithm provides: (i) the best mapping of peaks from an “input” peak list to the “root” peak list for their comparable dimensions; (ii) the registration for translating the “input” peak list to the “root” peak list in their comparable dimensions; and (iii) the standard deviations of this registration, which are needed to calculate match tolerances.

3.2.3.2 Spin system grouping algorithm

The spin system grouping algorithm utilizes the registration algorithm in order to infer match tolerance values for a single peak list first and then for multiple different peak lists and therefore consists of two sub-algorithms – one for single peak lists spin system grouping and the other one for grouping peaks from several different peak lists.

By applying the self-registration algorithm, the uncertainty in the chemical shift values of peak lists that have more than one peak per spin system be statistically analyzed. Next, the algorithm performs an averaging of root resonance values of these initial peak groups to improve their estimation of chemical shift values in terms of their standard error. The pairwise grouping algorithm merges peaks across different peak lists using statistics derived from the pairwise-registration algorithm.

3.2.3.3 Amino acid typing algorithm

This algorithm creates an ordered list of the highest probable amino acid types for each spin system. Most of the current amino acid typing algorithms use chemical shift statistics directly derived from BMRB, without considering secondary structure and resonance covariance information. However, it is well-known that chemical shift values for Cα and Cβ are secondary-structure-dependent [28]. The Re-referenced Protein Chemical shift Database (RefDB) [64] contains corrected or re-referenced expected chemical shift values, derived from the BMRB, but organized in tables depending on the protein secondary structure conformation: coil, helix, beta strands, and average of three. Therefore, RefDB secondary-structure-specific tables for the underlying chemical shift statistics are used within Bayesian-based amino acid typing algorithm.

3.2.3.4 Linking and mapping algorithms

The goal of the linking algorithm is to identify nearest neighbor spin systems through the redundancy information present between intraresidue and sequential ladders during the global spin systems comparison. Identified neighbor spin systems are then linked together into longer segments. Next, the goal of mapping algorithm is to map the generated segments to the most probable locations within the protein sequence. Linking and mapping algorithms are described in more detail in the future directions section of Chapter 7.

CHAPTER 4
NMRSTARLIB – TOOL FOR ACCESSING AND MANIPULATING NMR-STAR FILES

4.1 Overview

The Biological Magnetic Resonance Data Bank (BMRB) is a public repository of Nuclear Magnetic Resonance (NMR) spectroscopic data of biological macromolecules. It is an important resource for many researchers using NMR to study structural, biophysical, and biochemical properties of biological macromolecules. It is primarily maintained and accessed in a flat file ASCII format known as NMR-STAR. While the format is human readable, the size of most BMRB entries makes computer readability and explicit representation a practical requirement for almost any rigorous systematic analysis.

To aid in the use of this public resource, the package called nmrstarlib in the popular open-source programming language Python was developed. The nmrstarlib's implementation is very efficient, both in design and execution. The library has facilities for reading and writing both NMR-STAR version 2.1 and 3.1 formatted files, parsing them into usable Python dictionary- and list-based data structures, making access and manipulation of the experimental data very natural within Python programs (i.e. "saveframe" and "loop" records represented as individual Python dictionary data structures). Another major advantage of this design is that data stored in original NMR-STAR can be easily converted into its equivalent JavaScript Object Notation (JSON) format, a lightweight data interchange format, facilitating data access and manipulation using Python and any other programming language that implements a JSON parser/generator (i.e., all popular programming languages). Also tools to easily access and visualize assigned chemical shift values and to convert between NMR-STAR and JSONized NMR-STAR formatted files were developed. The nmrstarlib package can also be used to generate a wide range of simulated peak lists and introduce multiple sources of variance in order to generate more realistic data sets. The full API Reference Documentation, User Guide and Tutorial with code examples are also available online [65].

The library was tested on all current BMRB entries: 100% of all entries are parsed without any errors for both NMR-STAR version 2.1 and version 3.1 formatted files. Also comparison of software to three currently available Python libraries is provided for parsing NMR-STAR formatted files: PyStarLib, NMRPyStar, and PyNMRSTAR.

The nmrstarlib is a simple, fast, and efficient library for accessing data from the BMRB. The library provides an intuitive dictionary-based interface with which Python programs can read, edit, and write NMR-STAR formatted files and their equivalent JSONized NMR-STAR files. The nmrstarlib can be used as a library for accessing and manipulating data stored in NMR-STAR files and as a command-line tool to convert from NMR-STAR file format into its equivalent JSON file format and vice versa, generate a large number of simulated peak lists, and visualize chemical shift values.

The library was developed with the following use cases in mind: ability to access assigned chemical shift values, ability to access experimental peak lists if they are available, ability to generate a large number of simulated peak lists from assigned chemical shift values and account for multiple sources of variance. The following chapter provides the implementation description and various tests that were performed with the library using BMRB data.

4.2 Introduction

The Biological Magnetic Resonance Data Bank (BMRB) is a free, publicly-accessible repository of data on peptides, proteins, and nucleic acids obtained through NMR Spectroscopy [15], that is part of the worldwide Protein Databank (wwPDB) [66]. It currently consists of more than 11,000 individual NMR-STAR file entries, containing a wide range of NMR spectral data, experimental details, and biochemical data collected from thousands of biological samples. The NMR-STAR format is based on the Self-defining Text Archival and Retrieving (STAR) flat file database format [67], with some modifications specific to the BMRB. STAR provides a hierarchical dictionary structure for storing arbitrary data. In NMR-STAR, the format specifies top-level dictionaries called “saveframes”, which are used to categorize the data and meta-data about the experiment. Inside each saveframe is an arbitrarily number of key-value pairs and tables of records (loops). The key-value pairs store a single piece of information under a descriptive variable name. Each loop stores a table of records, each record containing a set of values representing individual fields in the record. There are currently two active versions of the BMRB: version 2.1 and version 3.1. While they both use the same NMR-

STAR format at the most general level, the layout of the data in the two formats is different.

Python is a free, open-source scripting language which runs on all major operating systems [68], [69]. It is designed to facilitate the development and maintenance of simple, efficient, and readable code. Python has object-oriented programming facilities and includes several high-level data structure objects in its standard library. Among these are the dictionary, a data structure implemented via the `dict` class that stores data as a set of key-value pairs (specific mappings between keys and values). The `OrderedDict` class is identical to the `dict` class except that the order of inserted keys-value pairs is remembered. This is particularly useful for categorical data with sequential relationships. The dictionary data structure is the most straightforward mechanism for representing and using data from NMR-STAR files, which have a nested, mostly dictionary-like structure themselves. However, to my knowledge no NMR-STAR parsing library using this design exists. The newest major version of Python (version 3.0.0), was initially released on 2008-12-03, however many software libraries and utilities written in Python still use Python version 2.x exclusively. As Python version 3.1 brings many substantial improvements over Python 2.x (including the addition of the `OrderedDict` class, which was later back-ported to Python version 2.7 [70]). As of Python version 3.5 `OrderedDict` is implemented in C, which makes it much faster than the Python 2.7 implementation of `OrderedDict`. Moreover, in Python 3.6, the `dict` data structure implementation becomes ordered by default and `dict` and `OrderedDict` are more efficient than in any previous versions of Python. While support for Python 2.7 is provided for use by legacy code, I believe that researchers will prefer libraries and tools written in latest version of Python (currently 3.6) in order to develop maintainable codebases, especially as Python version 2.x becomes less supported over time. Moreover, Python version 2.7 will no longer be maintained after the Spring of 2020 [71]. Two publically available Python libraries for parsing NMR-STAR format files PyStarLib [72] and NMRPyStar [73] both require Python version 2.7. PyNMRSTAR [74] works with both major versions of Python (2.7 and 3.3+).

4.3 Implementation

The nmrstarlib package consists of several modules: `nmrstarlib.py`, `bmrblex.py`, `converter.py`, `csviewer.py`, `plsimulator.py`, `translator.py`, and `noise.py` (**Figure 19a**). The `nmrstarlib.py` module (**Figure 19c**) provides the `StarFile` class, which implements a nested Python dictionary/list representation of a BMRB NMR-STAR file. Once a NMR-STAR formatted file is processed into a `StarFile` object, experimental data can be accessed directly from the `StarFile` object, using bracket accessors as with any regular Python `dict` object. The `nmrstarlib.py` module relies on the `bmrblex.py` module (**Figure 19b**) for processing of tokens. The `bmrblex.py` module provides the `bmrblex` generator – BMRB lexical analyzer (parser). Two versions of the `bmrblex` module are provided: a pure Python version (`bmrblex.py`) and a Python + C extension (`bmrblex.pyx`, `cbmrblex.c`) for faster performance. The compiled C extensions are implemented in the Cython programming language [75], which I will call the Cython implementation. If the Cython implementation of `bmrblex` fails for any reason, the library will use the Python implementation, ensuring that the library always works.

The library creates an internal representation of the NMR-STAR format as a nesting of `OrderedDict` objects with the top-level object `StarFile` inheriting from the `OrderedDict` class (**Figure 19c**). This allows the user to access data in its original NMR-STAR organization using familiar Python dictionary syntax. The library provides facilities to read data from NMR-STAR formatted files into an internal `StarFile` object, to access and make modifications to this `StarFile` object, and to save the resulting `StarFile` object as a new NMR-STAR formatted file. It is also possible to create NMR-STAR files from scratch using this library; however, this requires the user to adhere to the recommended layout for NMR-STAR formatted files by adding keys and values to the `StarFile` object in the appropriate order.

The `nmrstarlib.py` module provides a memory-efficient `read_files()` generator function (**Figure 19c**) that yields (emits) `StarFile` objects, one at a time for each file parsed. When reading an NMR-STAR formatted file (**Figure 20**), the `read_files()` generator function first opens the file and passes a filehandle to the `StarFile.read()` method that reads the text into Python as a string and passes that string into the `bmrblex` object that then splits the text into tokens. As the `bmrblex` lexical analyzer keeps emitting valid tokens, the `StarFile` object is constructed sequentially. The `StarFile` object decides what type of token it is dealing with and chooses which internal method to call in order to construct itself, i.e. calls to `StarFile._build_starfile()`, `Starfile._build_saveframe()`, or `StarFile._build_loop()`.

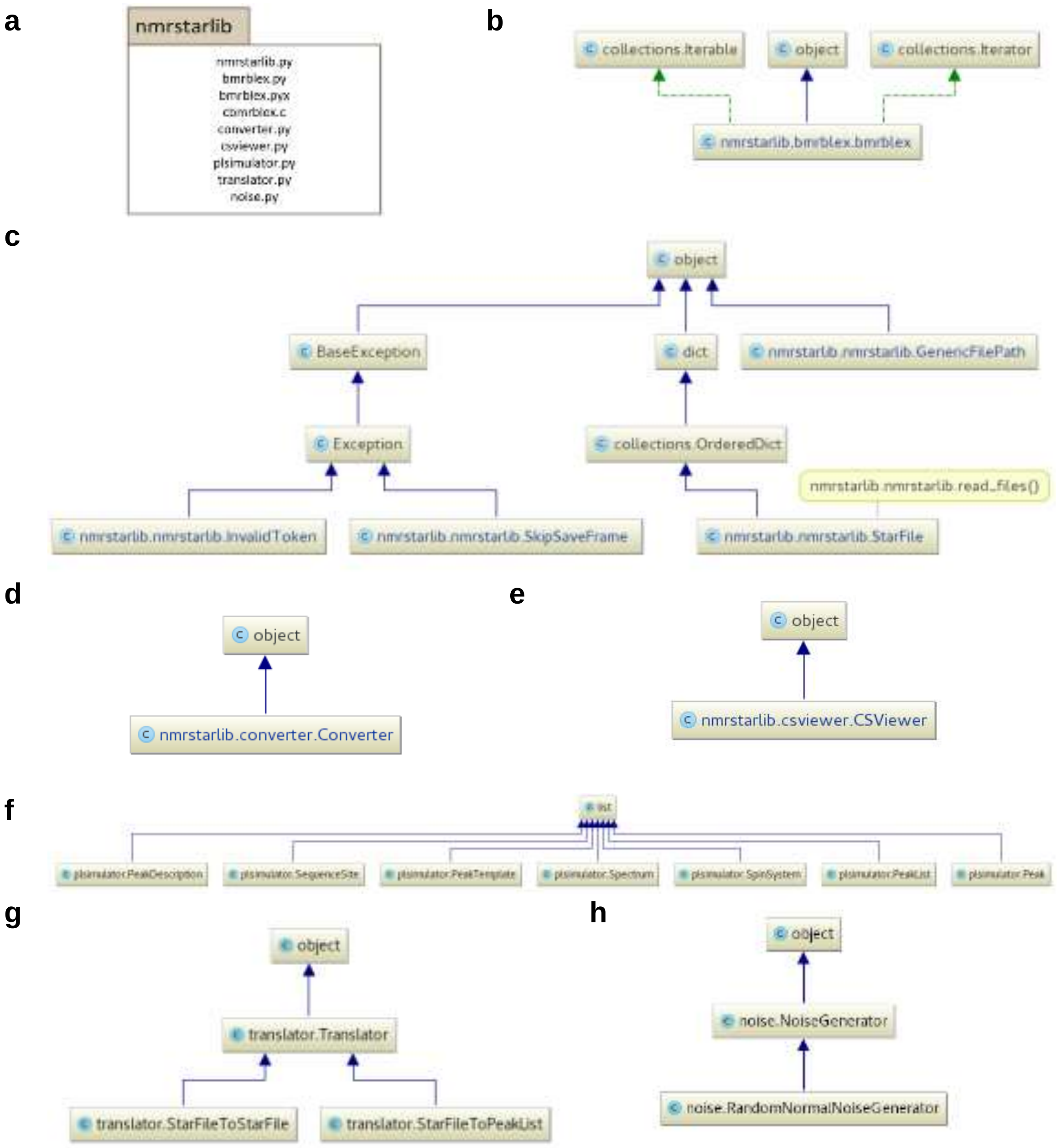


Figure 19. Organization of the nmrstarlib package version 2.0.0. **a**) UML package diagram of the nmrstarlib library; **b**) UML class diagram of the `bmrblex.py` `(bmrblex.pyx)` module; **c)** UML class diagram of the `nmrstarlib.py` module; **d)** UML class diagram of the `converter.py` module; **e)** UML class diagram of the `csviewer.py` module; **f)** UML class diagram of the `plsimulator.py` module; **g)** UML class diagram of the `translator.py` module; **h)** UML class diagram of the `noise.py` module.

For example, **Figure 20** shows the function call diagram during the `StarFile` object creation: the `_build_saveframe()` method is called 25 times and `_build_loop()` is called 37 times, meaning that the NMR-STAR file consists of 25 different saveframe categories and 37 loops. The total number of tokens processed is equal to 36,155 = 27 (from `_build_starfile`) + 786 (from `_build_saveframe`) + 35,342 (from `_build_loop`).

Each saveframe category is also an `OrderedDict` data structure that can be accessed by saveframe name as the key from the top-level `StarFile` object. Once a saveframe dictionary is constructed and populated with key-value pairs, it descends further into each loop and constructs a `tuple` of two `lists`: the first `list` corresponding to loop field keys (loop field names); the second `list` consists of `OrderedDict` objects corresponding to loop rows (loop records) in the original NMR-STAR file. By the end of parsing, a single nested dictionary/list structure in the form of a `StarFile` dictionary object (**Figure 21b**) is constructed, emulating the structure of the original NMR-STAR formatted file (**Figure 21a**). In addition, comments can be parsed and included as additional key-value pairs within the nested dictionary structure.

The `nmrstarlib.py` module provides a `GenericFilePath` (**Figure 19c** and **Figure 20**) object that is used by the `read_files()` generator function in order to open NMR-STAR formatted files from many different sources: a single file on a local machine; a URL address of a single file; a directory of files on a local machine; an archive of files on a local machine; a URL address of an archive of files; or the BMRB id of a single file.

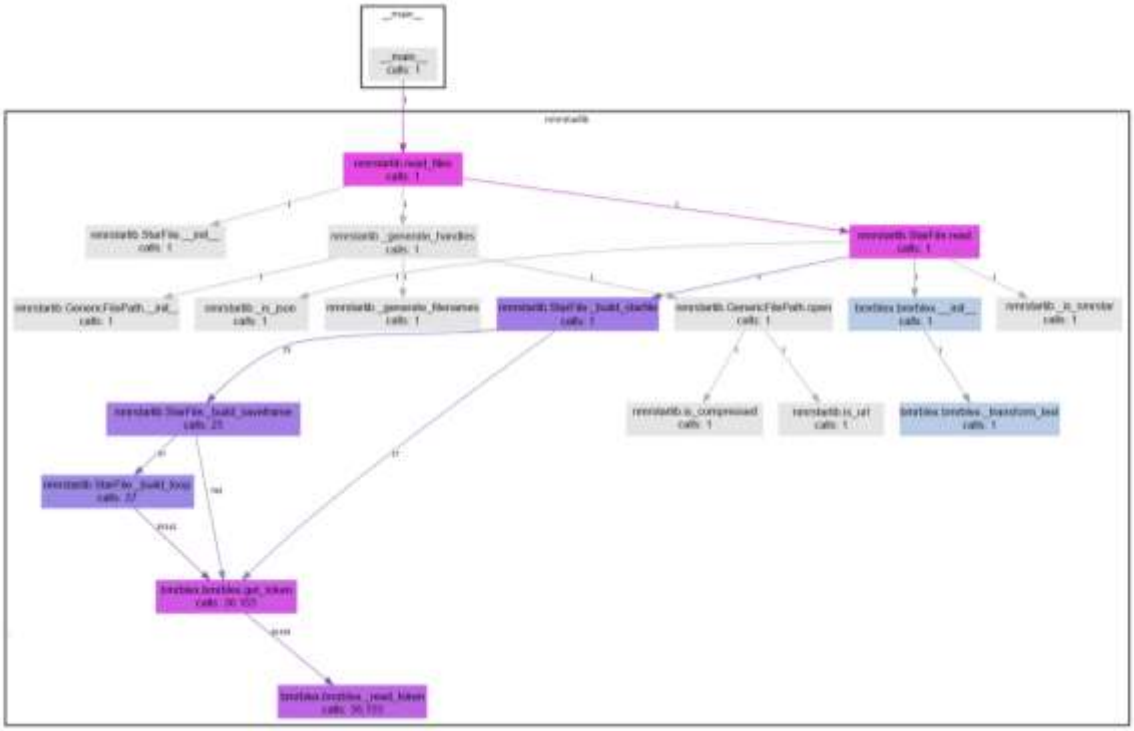


Figure 20. Diagram showing what function calls are made during the process of StarFile object creation.

To write from a StarFile object to an NMR-STAR formatted file, the library recursively crawls through the StarFile dictionary structure, formatting and printing each of the keys and corresponding values sequentially. This allows to recall the sequential order of the original NMR-STAR formatted file, due to the stored ordering of key insertion from the underlying OrderedDict objects. Using Python's json library, the entire StarFile dictionary structure can be saved as JSON (JavaScript Object Notation), which is an open, human-readable, lightweight data exchange format that is readable by most programming languages via optimized parsing libraries. This JSON conversion of StarFile objects greatly facilitated the implementation of the converter.py module which converts original NMR-STAR formatted files into their equivalent JSONized NMR-STAR files and vice versa. The converter.py module

(**Figure 19d**) consists of a single `Converter` class which can convert in both one-to-one (single file) and many-to-many (directory or archive of files) modes.

a

```
data_336

save_entry_information
  _Entry.Sf_category       entry_information
  _Entry.Sf_framecode      entry_information
  _Entry.ID       336
  _Entry.Title
;
1H-NMR studies of structural homologies between
the heme environments in horse cytochrome c and
in cytochrome c-552 from Euglena gracilis
;
  _Entry.Type     macromolecule
  _Entry.Version_type      update
  _Entry.Submission_date   1995-07-31
  _Entry.Accession_date    1996-04-12
  _Entry.Last_release_date         .
  _Entry.Original_release_date     .
  _Entry.Origination       BMRB
  _Entry.NMR_STAR_version  3.1.1.61
  _Entry.Original_NMR_STAR_version .
  _Entry.Experimental_method       NMR
  _Entry.Experimental_method_subtype        .
  _Entry.Details  .
  _Entry.BMRB_internal_directory_name       .
  loop_
    _Entry_author.Ordinal
    _Entry_author.Given_name
    _Entry_author.Family_name
    _Entry_author.First_initial
    _Entry_author.Middle_initials
    _Entry_author.Family_title
    _Entry_author.Entry_ID
   1 Regula Keller . M. . 336
   2 Kurt Wuthrich . . . 336
  stop_
save_
```

b

```
{
  "data": "336",
  "save_entry_information": {
    "Entry.Sf_category": "entry_information",
    "Entry.Sf_framecode": "entry_information",
    "Entry.ID": "336",
    "Entry.Title": "\n;\n1H-NMR studies of
structural homologies between the heme
environments in horse \ncytochrome c and in
cytochrome c-552 from Euglena gracilis\n;",
    "Entry.Type": "macromolecule",
    "Entry.Version_type": "update",
    "Entry.Submission_date": "1995-07-31",
    "Entry.Accession_date": "1996-04-12",
    "Entry.Last_release_date": ".",
    "Entry.Original_release_date": ".",
    "Entry.Origination": "BMRB",
    "Entry.NMR_STAR_version": "3.1.1.61",
    "Entry.Original_NMR_STAR_version": ".",
    "Entry.Experimental_method": "NMR",
    "Entry.Experimental_method_subtype": ".",
    "Entry.Details": ".",
    "Entry.BMRB_internal_directory_name": ".",
    "loop_0": [
      [
      "Entry_author.Ordinal",
      "Entry_author.Given_name",
      "Entry_author.Family_name",
      "Entry_author.First_initial",
      "Entry_author.Middle_initials",
      "Entry_author.Family_title",
      "Entry_author.Entry_ID"
      ],
      [
        {
        "Entry_author.Ordinal": "1",
        "Entry_author.Given_name": "Regula",
        "Entry_author.Family_name": "Keller",
        "Entry_author.First_initial": ".",
        "Entry_author.Middle_initials": "M.",
        "Entry_author.Family_title": ".",
        "Entry_author.Entry_ID": "336"
        },
        {
        "Entry_author.Ordinal": "2",
        "Entry_author.Given_name": "Kurt",
        "Entry_author.Family_name": "Wuthrich",
        "Entry_author.First_initial": ".",
        "Entry_author.Middle_initials": ".",
        "Entry_author.Family_title": ".",
        "Entry_author.Entry_ID": "336"
        }
      ]
    ]
  }
}
```

Figure 21. Internal `StarFile` object representation and correspondence to NMR-STAR format without comments: a) An example of a NMR-STAR formatted file; b) `StarFile` dictionary representation equivalent to the NMR-STAR formatted file and the JSONized version of the NMR-STAR file.

The `converter.py` module relies on the `translator.py` module (**Figure 19g**) in order to decide what type of conversion to perform, i.e. convert between NMR-STAR format and JSONized NMR-STAR format (`StarFileToStarFile`) or from NMR-STAR file to peak list file (`StarFileToPeakList`).

The `plsimulator.py` (**Figure 19f**) module provides facilities necessary to generate different types of simulated peak lists. The `noise.py` module (**Figure 19h**) provides a `NoiseGenerator` class that is responsible for addition of random normal noise to peaks during simulated peak list creation.

In order to simplify access to assigned chemical shift data, the `csviewer.py` module was created (**Figure 19e**) that includes the `CSViewer` class that can access both the NMR-STAR version 2.1 and version 3.1 assigned chemical shifts loop and visualize (organize) chemical shift values by amino acid residue type, and save this visualization as an image file or a pdf document (**Figure 22**). The `csviewer.py` module requires the graphviz Python library [76] in order to create an output file. In addition to visualizing chemical shift values, the `csviewer.py` module provide code example for utilizing the nmrstarlib library.

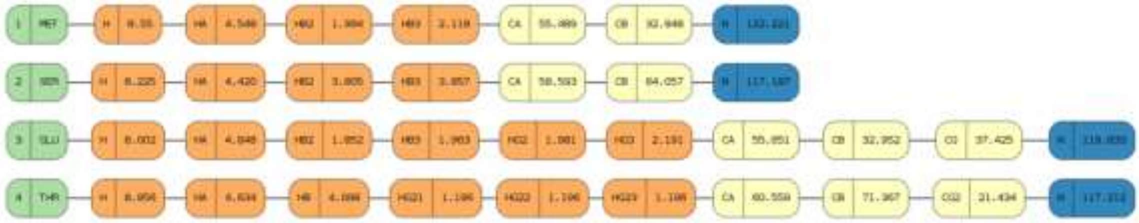


Figure 22. Example of output file: chemical shifts organized by amino acid residue type produced by `csviewer.py` module.

Overall, the nmrstarlib package can be used in two ways: 1) as a library for accessing and manipulating data stored in NMR-STAR formatted files, converting between NMR-STAR and its equivalent JSON format, create set of simulated peak lists, and visualizing assigned chemical shift values; or 2) as a standalone command-line tool for converting files in bulk and visualizing assigned chemical shift values. The docopt Python library [77] was utilized to create the nmrstarlib package command-line interface.

4.4 Results

4.4.1 Performance on NMR-STAR formatted files

As part of nmrstarlib's development process, the library was tested extensively against the entire BMRB for both NMR-STAR version 2.1 and version 3.1 [78]. To measure the performance speed of the nmrstarlib library, a simple program was used that accesses NMR-STAR files from local directory one file at a time, which then creates a `StarFile` object and records how much time in seconds it took to create the object.

Table 5. The nmrstarlib library performance test against NMR-STAR formatted files using pure Python and Python with C extension and against JSONized NMR-STAR files using the standard Python library `json` parser and the UltraJSON (`ujson`) 3^{rd} party library.

			NMR-STAR 2.1	**NMR-STAR 3.1**	**JSONized NMR-STAR 2.1**	**JSONized NMR-STAR 3.1**
Number of files			11,270	11,244	11,270	11,244
Total size of files, GB			1.1	1.8	4.6	22.0
Time, sec	Pure Python	json	326	1,100	30	130
	Python with C extension	*ujson	320	423	27	126
Average reading speed, KB/sec	Pure Python	json	3,290	1,700	158,549	176,479
	Python with C extension	*ujson	3,351	4,421	176,166	182,082

* Support for the `ujson` library for versions of Python is implemented starting with Python 3.6, because the `ujson` library does not provide methods to keep the `dict` data structure in order when parsing from JSON files; however, starting with Python 3.6, the `dict` data structure is ordered by default.

Table 5 shows that library was able to read the entire BMRB for both NMR-STAR version 2.1 and version 3.1 without any errors. With the pure Python implementation, it took 1,110 sec (~18.3 min) and 326 sec (~5.4 min) to read NMR-

STAR version 3.1 and NMR-STAR version 2.1, respectively. With the more efficient Cython implementation, it took 423 sec (~7 min) and 320 sec (~5.3 min) to read NMR-STAR version 3.1 and NMR-STAR version 2.1, respectively. The metric kilobytes per second (KB/sec) was used, because files/sec would be a misleading metric due to widely varying files sizes in the BMRB and because read times scale almost linearly (**Figure 23**) with file size. As such, the nmrstarlib's average reading speed is 1,700 KB/sec (NMR-STAR 3.1) and 3,290 KB/sec (NMR-STAR 2.1) for the Python implementation and 4,421 KB/sec (NMR-STAR 3.1) and 3,351 KB/sec (NMR-STAR 2.1) for the Cython implementation on the hardware used for testing. The NMR-STAR 3.1 is more comprehensive than NMR-STAR 2.1 and usually represents more experimental information and details. This additional complexity is computationally harder to parse. However, for Cython implementation the average reading speed for NMR-STAR 3.1 was faster than for NMR-STAR 2.1 due to multiline text pre-processing discussed in more detail in the next section.

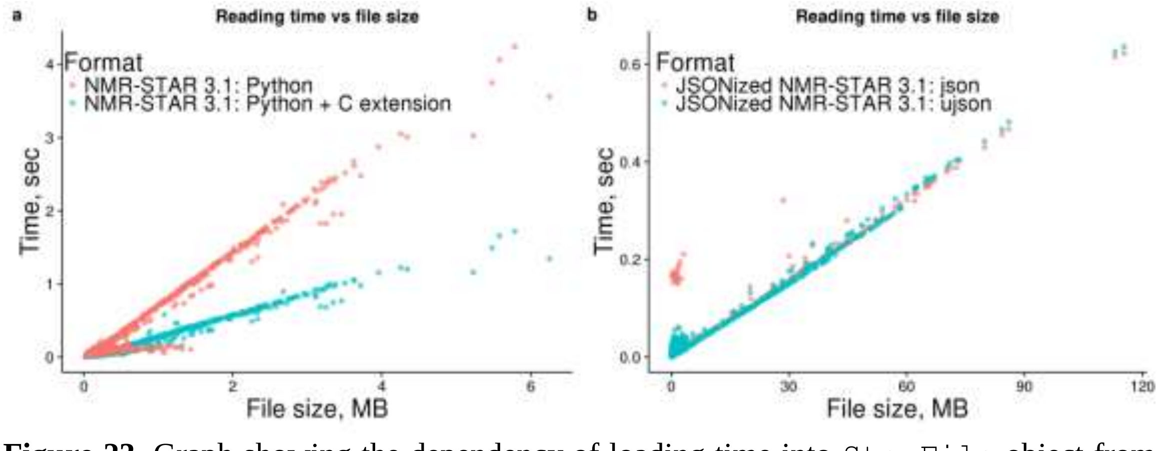


Figure 23. Graph showing the dependency of loading time into `StarFile` object from the size of file: **a)** Loading times for NMR-STAR 3.1 formatted files; **b)** Loading times for JSONized NMR-STAR 3.1 files.

4.4.2 Performance on JSONized NMR-STAR files

Next, both NMR-STAR version 2.1 and version 3.1 files were converted into their equivalent JSON format and speed tests were performed again (**Table 5**). The reading times of both JSONized NMR-STAR version 2.1 and version 3.1 were significantly faster than read times of the original NMR-STAR formatted files: 130 sec (~2.2 min) and 30 sec (~0.5 min) for NMR-STAR version 3.1 and NMR-STAR version 2.1, respectively, for the entire BMRB data set. The average read speed was 176,479 KB/sec and 158,549 KB/sec for version 3.1 and version 2.1, respectively. Next, performance tests were repeated using another compiled JSON parsing third-party library, UltraJSON (`ujson`) [79]. The reading times and average reading speeds of JSONized NMR-STAR files were slightly faster than using the built-in `json` parser: 127 sec (182,082 KB/sec) and 27 sec (176,166 KB/sec) for version 3.1 and version 2.1 respectively (**Table 5**).

Table 6. Converting NMR-STAR formatted files into their equivalent JSON format.

	Directory		**zip archive**		**tar.gz archive**		**tar.bz2 archive**	
Format	NMR-STAR 2.1	NMR-STAR 3.1	NMR-STAR 2.1	NMR-STAR 3.1	NMR-STAR 2.1	NMR-STAR 3.1	NMR-STAR 2.1	NMR-STAR 3.1
Number of files	11,270	11,244	11,270	11,244	11,270	11,244	11,270	11,244
Time, min	8	20	9	22	12	27	15	68
Total size, MB	4,756	22,942	230	470	200	409	131	222

Table 6 shows how much time it took to convert the entire BMRB into its JSONized version and how much disk space it occupied as uncompressed directory and as compressed zip and tar archives. Compressed zip and tar formats represent the entire BMRB database in a single file and save disk space. In order to simplify access, library provides facilities to directly read NMR-STAR files from zip and tar archives without the requirement to manually decompress and separate the archive into separate files first. Frequency polygons of loading times on **Figure 24** show that the majority of NMR-STAR and JSONized NMR-STAR files can be loaded into a `StarFile` object in less

than 1 second per file and JSONized NMR-STAR files can be loaded much faster than the original NMR-STAR files. **Figure 24a** and **Figure 24b** show that the fastest reading times were for parsing JSONized NMR-STAR files using the `ujson` and `json` parsers. However on **Figure 24a**, it is clear that the pure Python implementation outperformed the Cython implementation for some of the NMR-STAR 2.1 files (e.g. BMRB ID: 17192, 16692). This is because those files contain saveframe categories deposited as very large multiline blocks of text and the majority of time is spent to pre-process them, equivalent NMR-STAR 3.1 files have those saveframes properly formatted and do not require extra time to pre-process multiline text blocks. For NMR-STAR 3.1 formatted files (**Figure 24b**), the Cython implementation outperformed pure Python implementation in all cases.

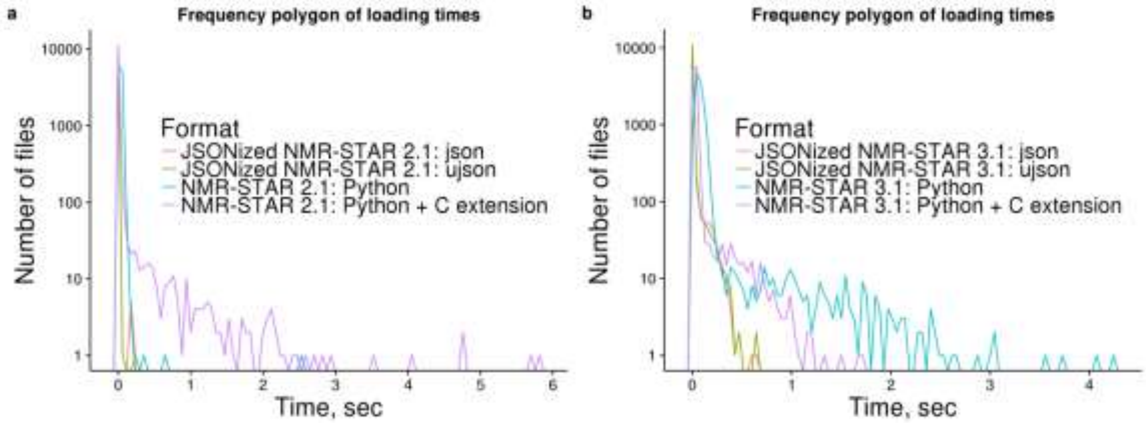


Figure 24. Frequency polygon of loading times for NMR-STAR files: **a**) Comparison of loading times between NMR-STAR 2.1 and JSONized NMR-STAR 2.1; **b**) Comparison of loading times between NMR-STAR 3.1 and JSONized NMR-STAR 3.1.

4.4.3 Comparison to similar existing software

Using the entire BMRB, speed performance tests between the nmrstarlib package and the three other publically available Python libraries for reading NMR-STAR formatted files were performed: PyStarLib [72], NMRPyStar [73], and PyNMRSTAR [74]. For each of these libraries, a simple Python program that loads a NMR-STAR formatted file from a directory, creates an object representation, and then reports how much time it took to process each file. Results of these comparisons are summarized in **Table 7**. For the pure Python implementation, PyStarLib showed the fastest reading time: 239 sec (~4 min) and 796 sec (~13.3 min) for NMR-STAR version 2.1 and version 3.1 respectively, but it was not able to parse 0.43 % (48 files) NMR-STAR version 2.1 and 4.08 % (459 files) NMR-STAR version 3.1. All errors occurred inside a function that is responsible for processing multiline quoted text, which uses regular expressions to collapse multiline quoted text into a single token. The most probable cause for these errors is a regular expression that is not capable of handling all edge cases. Examples of failures include files where: i) multiline quoted text included a semicolon character inside the text; ii) multiline quoted text that is not followed by the new line character; and iii) multiline quoted text followed by a loop.

Table 7. Performance comparison of nmrstarlib to other Python libraries.

		nmrstarlib	**PyStarLib**	**NMRPyStar**	**PyNMRSTAR**
Parsing NMR-STAR 2.1					
Number of files		11,270	11,270	11,270	11,270
Time, sec	Pure Python	326	239	N/A	547
	Python with C Extension	320	N/A	N/A	144
Success rate, %		100	99.57	0	100
Parsing NMR-STAR 3.1					
Number of files		11,244	11,244	11,244	11,244
Time, sec	Pure Python	1,100	796	56,569	2,354
	Python with C Extension	423	N/A	N/A	538
Success rate, %		100	95.92	100	100

The pure Python implementation of the nmrstarlib package was the second fastest method 326 sec (~5.4 min) and 1,110 sec (~18.3 min) and, more importantly, parsed 100% of files for both NMR-STAR 2.1 and NMR-STAR 3.1, respectively. The NMRPyStar library showed the slowest results, taking 56,569 sec (~15.7 hours) to process NMR-STAR version 3.1 and was not able to read any of the NMR-STAR version 2.1 files (error status code was reported by the program during execution). Both the nmrstarlib and PyNMRSTAR provide Python + C extension implementations in order to speed up the tokenization process. The nmrstarlib performed faster than PyNMRSTAR on NMR-STAR 3.1 files: 423 sec (~7 min) versus 538 sec (~9 min). However, PyNMRSTAR was faster than nmrstarlib on NMR-STAR 2.1 files: 144 sec (~2.4 min) versus 320 sec (~5.3 min). Overall, the nmrstarlib (Python + C extension implementation) was the fastest method to read NMR-STAR 3.1 files, and PyNMRSTAR (Python + C extension implementation) was the fastest method to read NMR-STAR 2.1 files. However, when using the JSONized versions of NMR-STAR files with the nmrstarlib library, parsing speed can be further improved to 30 sec for NMR-STAR 2.1 and 130 sec for NMR-STAR 3.1 (see **Table 5**).

All tests were performed on a single workstation desktop computer with Intel(R) Core(TM) i7-4930K CPU @ 3.40GHz processor, 64 GB memory, and a solid-state drive. The latest stable version of Python (Python 3.6.0) was used to compare libraries. Python version 2.7 was used for libraries that do not support the latest version of Python.

4.5 Discussion

4.5.1 The nmrstarlib interface

To use nmrstarlib as a library, first import the library. Next, create a `StarFile` generator that will return `StarFile` instances one at a time from many different file sources: a local file, URL address of a file, directory, archive, BMRB id. Next, the `StarFile` object can be utilized like any built-in Python `dict` object. **Table 8** shows common usage patterns for reading NMR-STAR files into `StarFile` objects, accessing and manipulating data using bracket accessors, and writing `StarFile` objects back to both NMR-STAR and JSONized NMR-STAR formats. For more detailed examples, see "The nmrstarlib Tutorial" documentation available online [65].

Table 8. Common usage patterns for the nmrstarlib module.

Usage	Example

Reading:	`sf_gen = nmrstarlib.read_files('path')`
	`starfile = next(sf_gen)`

Access/Modification:	`starfile['saveframe']['key']`
	`starfile['saveframe']['key'] = new_value`

Writing:	`starfile.write(fileobj, fileformat='nmrstar')`
	`starfile.write(fileobj, fileformat='json')`

Table 9. The nmrstarlib library command-line interface common usage patterns.

Command	Description	Example

`convert`	Convert between NMR-STAR and JSON formats	`$ python3 -m nmrstarlib convert bmr18569.str 18569.json \`
		`          --from_format=nmrstar --to_format=json`
		
		`$ python3 -m nmrstarlib convert 18569.json bmr18569.str \`
		`          --from_format=json --to_format=nmrstar`

`plsimulate`	Convert NMR-STAR formatted file into simulated peak list file	`python3 -m nmrstarlib plsimulate \`
		`        bmr18569.txt 18569_peaklist.txt HNcoCACB \`
		`        --from_format=nmrstar --to_format=sparky`
		
		`python3 -m nmrstarlib plsimulate \`
		`        18569 18569_peaklist.txt HNcoCACB \`
		`        --from_format=nmrstar --to_format=sparky \`
		`        --H_std=0.001 --N_std=0.01 --C_std=0.01`

`csview`	View assigned chemical shifts	`$ python3 -m nmrstarlib csview 18569 \`
		`          --csview_outfile=18569_cs_all`
		`          --csview_format=png`
		
		`$ python3 -m nmrstarlib csview 18569 \`
		`          --aminoacids=GLU,THR`
		`          --atoms=CA,CB,CG,CG2 \`
		`          --csview_outfile=18569_cs_GLU_THR_CA_CB_CG_CG2 \`
		`          --csview_format=png`

The nmrstarlib command-line interface provides several commands: the `convert` command in order to convert between NMR-STAR format and its equivalent JSON format; the `plsimulate` command to create simulated peak lists from assigned chemical shift values; the `csview` command for quick access to assigned chemical shift data of a single `StarFile`, organizing chemical shifts by amino acid residue type. **Table 9** shows common usage examples for the `convert`, `plsimulate` and `csview` commands. For a full list of available conversion and peak list simulation options and more detailed examples see “The nmrstarlib API Reference” and “The nmrstarlib Tutorial” documentation [65].

Also the “User Guide”, “The nmrstarlib Tutorial” and “The nmrstarlib API Reference” documentation and up-to-date online documentation were developed (**Table 10**).

Table 10. Comparison of nmrstarlib to other Python libraries.

Feature	nmrstarlib	PyStarLib	NMRPyStar	PyNMRSTAR
Read NMR-STAR 2.1	Yes	Yes	No	Yes
Read NMR-STAR 3.1	Yes	Yes	Yes	Yes
Supported Python version	2.7, 3.4+	2.7	2.7	2.6, 2.7, 3.3+
API Reference documentation	Yes	No	No	Yes
Tutorial documentation	Yes	No	No	Yes
PDF of documentation	Yes	No	No	Yes
User Guide documentation	Yes	No	Yes	No
Up to date online documentation	Yes	No	No	No
Open Source License	MIT (GitHub)	GPL (SourceForge)	MIT (GitHub)	GPL (GitHub)

4.5.2 Advantages of using nmrstarlib and JSONized NMR-STAR version

One of the main advantages of the library is that it provides a one-to-one mapping between each of the following representations of BMRB entries: NMR-STAR format, internal Python `OrderedDict`- and `list`-based objects, and JSONized NMR-STAR format. This makes the library more Python-idiomatic, providing a very intuitive programming interface for accessing and manipulating NMR data. Another benefit of the nmrstarlib package is that the `bmrblex.py` lexical analyser module is written in a generic fashion, making it easy to adapt for parsing data from other STAR-related formats, for example, the Crystallographic Information File (CIF) and its closely related macromolecular CIF (mmCIF) format.

JSON is an open, programming language independent, human-readable, data exchange standard that represents data objects in a nested dictionary/list ASCII format. JSON is one of the most common formats for asynchronous browser/server communication as an alternative to XML (Extensible Markup Language). The JSON object representation was selected, because it has a smaller overhead compared to common XML object representations, making it faster to parse and more human-readable when formatted for this purpose. But more importantly, it facilitates a one-to-one mapping with both nested Python data structures and BMRB's nested data representations of their entries. While XML is more flexible, it is not easily represented by a nesting of standard Python data structures that would produce an intuitive programming interface. Also, JSONization of the original NMR-STAR files provides several advantages: i) much faster reading times (see **Table 5**) and ii) makes the data stored in BMRB entries easily accessible to other programming languages that have JSON parsers, i.e. all modern programming languages, scripting as well as compiled, without requiring to write a specific parser for the specialized NMR-STAR format. **Figure 25**, **Figure 26**, and **Figure 27** show code examples for accessing data from JSONized NMR-STAR files using R with the `jsonlite` library [80], JavaScript with the `jQuery` library [81], and C++ with the `RapidJSON` library [82], respectively.

But one disadvantage of using JSON format is that it is more verbose in comparison to the original NMR-STAR format. As a result, uncompressed JSONized NMR-STAR files occupy more disk space (**Table 6**). However, the nmrstarlib library offers the ability to read NMR-STAR files in both uncompressed (directory of files) and compressed (zip and tar archives) forms, making storage and access of JSONized NMR-STAR files very efficient.

R Example using `jsonlite` library

```
> # install library
> install.packages("jsonlite")

> # load library
> library(jsonlite)

> # load data
> starfile <- fromJSON("bmr18569.str.json")

> # print saveframe names
> names(starfile)
 [1] "data"              "save_entry_information"
 [3] "save_entry_citation"       "save_assembly"
 [5] "save_EVH1"            "save_natural_source"
 [7] "save_experimental_source"    "save_sample_1"
 [9] "save_sample_2"          "save_sample_3"
[11] "save_sample_4"          "save_sample_conditions_1"
[13] "save_sample_conditions_2"    "save_sample_conditions_3"
[15] "save_sample_conditions_4"    "save_AZARA"
[17] "save_xwinnmr"          "save_ANSIG"
[19] "save_CNS"            "save_spectrometer_1"
[21] "save_spectrometer_2"       "save_NMR_spectrometer_list"
[23] "save_experiment_list"      "save_chemical_shift_reference_1"
[25] "save_assigned_chem_shift_list_1" "save_combined_NOESY_peak_list"

> # access saveframe key-value data
> starfile$data
[1] "18569"
>
> starfile$save_entry_information$Entry.NMR_STAR_version
[1] "3.1.1.61"
>
> # access loop data
> starfile$save_entry_information$loop_1
[[1]]
[1] "Data_set.Type"   "Data_set.Count"  "Data_set.Entry_ID"

[[2]]
       Data_set.Type Data_set.Count Data_set.Entry_ID
1 assigned_chemical_shifts       1       18569
2    spectral_peak_list       1       18569
```

Figure 25. Code example showing how to access data from JSONized NMR-STAR files using R programming language.

JavaScript Example using `jQuery`

```
<!DOCTYPE html>
<html>
  <head>
    <title>Reading JSONized NMR-STAR with jQuery</title>
  </head>
  <body>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jquery.min.js"></script>
    <script>
      $.getJSON("bmr18569.str.json", function(starfile) {
        console.log(starfile.data);          // prints data tag id
        console.log(starfile.save_entry_information); // prints entire saveframe data
        console.log(starfile.save_entry_information.loop_1); // prints loop_1 data
      });
    </script>
  </body>
</html>
```

Figure 26. Code example showing how to access data from JSONized NMR-STAR files using JavaScript programming language.

C++ example using `RapidJSON` library

```
#include <iostream>

// include rapidjson headers
#include "rapidjson/document.h"
#include "rapidjson/filereadstream.h"

using namespace std;

int main()
{
  // open file
  FILE* fp = fopen("bmr18569.str.json", "r"); // Windows use "rb"

  // read input stream via FILE pointer
  char readBuffer[65536];
  rapidjson::FileReadStream is(fp, readBuffer, sizeof(readBuffer));

  // create rapidjson::Document and parse input stream
  rapidjson::Document starfile;
  starfile.ParseStream(is);

  fclose(fp); // close file pointer

  // print saveframe names
  cout << "Accessing saveframe categories: \n";
  for (rapidjson::Value::ConstMemberIterator itr = starfile.MemberBegin();
     itr != starfile.MemberEnd(); ++itr)
  {
    cout << "  " << itr->name.GetString() << "\n";
  }

  // access saveframe key-value data
  cout << "Accessing saveframe data: \n";
  cout << "  " << "data: " << starfile["data"].GetString() << "\n";
  cout << "  " << "NMR-STAR version: " <<
      starfile["save_entry_information"]["Entry.NMR_STAR_version"].GetString() << "\n";

  // access loop data
  cout << "Accessing loop data:\n";
  const rapidjson::Value& loop_1_fields =
starfile["save_entry_information"]["loop_1"][0];
  const rapidjson::Value& loop_1_values =
starfile["save_entry_information"]["loop_1"][1];

  cout << "loop fields:\n";
  for (rapidjson::SizeType i = 0; i < loop_1_fields.Size(); i++)
  {
    cout << "  " << loop_1_fields[i].GetString() << "\n";
  }

  cout << "loop values:\n";
  for (rapidjson::SizeType i = 0; i < loop_1_values.Size(); i++)
  {
    for (rapidjson::Value::ConstMemberIterator itr = loop_1_values[i].MemberBegin();
       itr != loop_1_values[i].MemberEnd(); ++itr)
    {
      itr->name.GetString();
      cout << "  " << itr->name.GetString() << ": " << itr->value.GetString() << "\n";
    }
  }
}

// Output after compiling and executing
Accessing saveframe categories:
  data
  save_entry_information
  save_entry_citation
  save_assembly
  save_EVH1
  save_natural_source
  save_experimental_source
  save_sample_1
  save_sample_2
  save_sample_3
  save_sample_4
  save_sample_conditions_1
  save_sample_conditions_2
  save_sample_conditions_3
  save_sample_conditions_4
  save_AZARA
  save_xwinnmr
  save_ANSIG
  save_CNS
  save_spectrometer_1
  save_spectrometer_2
  save_NMR_spectrometer_list
  save_experiment_list
  save_chemical_shift_reference_1
  save_assigned_chem_shift_list_1
  save_combined_NOESY_peak_list

Accessing saveframe data:
  data: 18569
  NMR-STAR version: 3.1.1.61

Accessing loop data:
loop fields:
  Data_set.Type
  Data_set.Count
  Data_set.Entry_ID
loop values:
  Data_set.Type: assigned_chemical_shifts
  Data_set.Count: 1
  Data_set.Entry_ID: 18569
  Data_set.Type: spectral_peak_list
  Data_set.Count: 1
  Data_set.Entry_ID: 18569
```

Figure 27. Code example showing how to access data from JSONized NMR-STAR files using C++ programming language.

4.6 Conclusions

The nmrstarlib package is a useful Python library, providing classes and other facilities for parsing, accessing, and manipulating data stored in NMR-STAR and JSONized NMR-STAR formats. Also, nmrstarlib provides a simple command-line interface that can convert from NMR-STAR file format into its equivalent JSON file format and vice versa, create large number of simulated peak lists, as well as access and visualize assigned chemical shift values. The library has an easy-to-use, idiomatic dictionary-based interface, usable in programs written in Python. The library also has extensive documentation including the “User Guide”, “The nmrstarlib Tutorial”, and “The nmrstarlib API Reference”. Furthermore, the easy conversion into the JSONized NMR-STAR format facilitates utilization of BMRB entries by programs in any programming language with a JSON parser. This same basic approach can be used to quickly JSONize other older text-based scientific data formats, making the underlying scientific data easily accessible in a wide variety of programming languages. As demonstrated in this study, many available JSON parsers are highly optimized and typically much more efficient than specialized parsers for scientific data formats. Thus, JSONization of older scientific data formats provides easy steps for reaching Interoperability and Reusability goals of FAIR guiding principles [83].

CHAPTER 5
INTERNAL REGISTRATION AND GROUPING ALGORITHMS

5.1 Overview

Peak lists derived from NMR spectra are commonly used as input data for a variety of computer assisted and automated analyses. These include automated protein resonance assignment and protein structure calculation software tools. Prior to these analyses, peak lists must be aligned to each other and sets of related peaks must be grouped based on common chemical shift dimensions. Even when programs can perform peak grouping, they require the user to provide uniform match tolerances or use default values. However, peak grouping is further complicated by multiple sources of variance in peak position limiting the effectiveness of grouping methods that utilize uniform match tolerances. In addition, no method currently exists for deriving peak positional variances from single peak lists for grouping peaks into spin systems, i.e. spin system grouping within a single peak list. Therefore, a complementary pair of peak list registration and spin system grouping algorithms was designed to overcome these limitations. These algorithms are implemented into an approach that can identify multiple dimension-specific positional variances that exist in a single peak list and group peaks from a single peak list into spin systems. The resulting algorithms generate a variety of useful statistics on both a single peak list and pairwise peak list alignment, especially for quality assessment of peak list datasets.

To facilitate evaluation, a peak list simulator within the nmrstarlib package was developed that generates user-defined assigned peak lists from a given BMRB entry or set of entries. A range of low and high quality experimental solution-state and solid-state NMR peak lists was used to assess performance of registration and grouping algorithms.

Analyses show that algorithms using only single iteration and uniform match tolerances approach are only able to recover from 50 % to 80 % of spin systems due to the presence of multiple sources of variance. The registration and grouping algorithm recovers additional spin systems by reevaluating match tolerances in multiple iterations. In addition, over 100,000 simulated peak lists with one or two sources of variance were generated to evaluate the performance and robustness of these new registration and peak grouping algorithms.

5.2 Introduction

One of the prerequisite analyses for protein structure determination is the assignment of chemical shifts to specific nuclei in a protein structure. During the assignment process, spin systems are mapped to individual amino acid residues in a protein sequence. In general, a spin system can be viewed as a group of nuclear spins that interact with each other in a magnetic field. In this study, a spin system is defined as a collection of related resonances associated with specific atoms in a molecule that can be grouped within a single spectrum and across multiple spectra with common resonances. In the context of biopolymers such as proteins, spin systems often represent resonances associated with atoms within one, two, or even three bonded residues. Manual resonance assignment is tedious and can take a significant amount of time. Therefore, a variety of automated and semi-automated assignment programs have been developed to facilitate the protein resonance assignment process, specifically for solution [84], [85] and solid-state NMR [58], [59]. The process of automated resonance assignment typically involves several major steps: grouping peaks across peak lists into spin systems, classification of those spin systems by possible amino acid type, linking neighboring spin systems into segments, and then mapping those segments onto protein sequence.

5.2.1 Lack of automated tools to determine match tolerances

One of the historical problems that has limited the use of automated and semi-automated protein resonance assignment tools along with other analyses of NMR peak lists is the requirement that users either specify uniform match tolerances typically for ^{1}H and ^{15}N resonances (for solution-state NMR) and ^{15}N, and ^{13}C resonances (for solid-state NMR) to perform spin systems grouping and linking, or rely on default uniform match tolerance values. Some programs even expect the user to provide spin systems instead of peak lists [58]. In essence, the user is left to determine which match tolerances should be used for their dataset. Restated, basic peak positional variance statistics that could be derived from the peak lists data are being required from the user, limiting the utility of these tools. Also, these same peak list statistics are useful for assessing the quality of peak lists, especially for downstream analyses [86], [87].

5.2.2 Presence of multiple sources of variance

Another problem that exists in experimental peak lists derived from both solution and solid-state NMR experiments is the presence of multiple variances in dimension-specific peak positions. In effect, there is a subset of peaks within a single peak list that have a smaller variance and can be grouped into spin systems using tighter match tolerance values, and a subset of peaks that have a larger variance in one or all dimensions that require larger match tolerance values for grouping into spin systems. On the one hand, using tighter tolerance values could result in failure to group peaks with larger variances, on the other hand using larger tolerance values could result in spin system overlap in peaks that have a smaller variance. This also limits the utility of uniform match tolerances for spin system grouping, linking and mapping algorithms. **Figure 28** demonstrates the presence of peak groups (clusters) with multiple sources of variance in peak positions within experimental HN(CO)CACB peak lists.

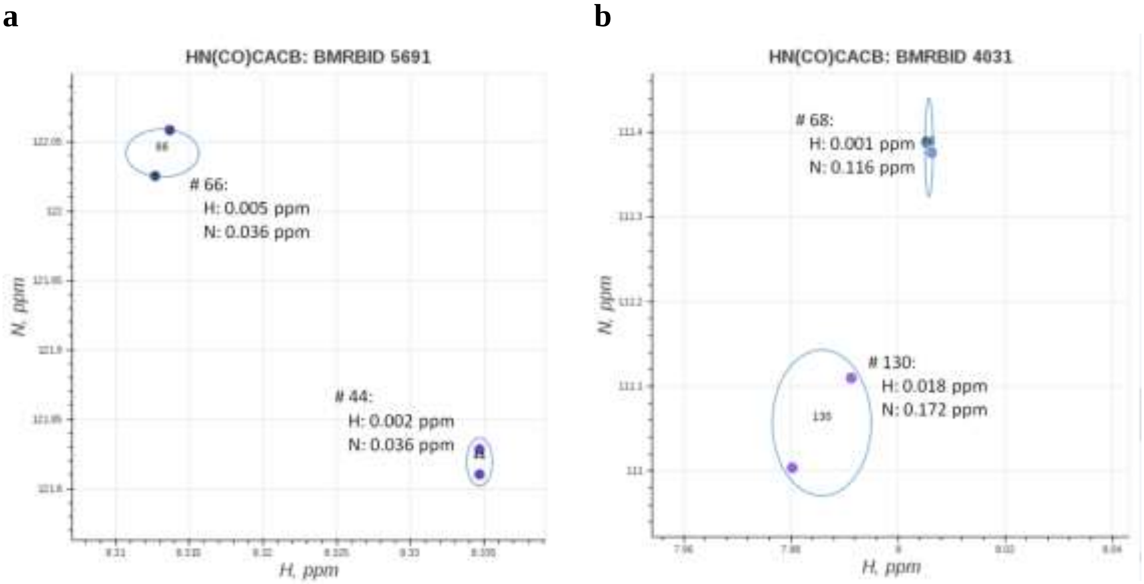


Figure 28. Zoomed-in visualization of spin systems taken from two experimental HN(CO)CACB peak lists that demonstrates the presence of multiple sources of variance within peak lists. The dots correspond to peak centers, two peaks form an individual spin system, ovals show the per-dimension variance (bivariance): **a)** for the 30S ribosomal protein S28E from *Pyrococcus horikoshii*, spin systems 44 and 66 show variance in the H dimension; **b)** for pancreatic ribonuclease both spin systems 68 and 130 show variance in both H and N dimensions.

For the 30S ribosomal protein S28E from *Pyrococcus horikoshii* in **Figure 28a**, the two visualized spins systems demonstrate different sources of variances in the amide ^{1}H dimension. For the pancreatic ribonuclease in **Figure 28b**, the visualized spin systems demonstrate multiple sources of variance in both amide ^{1}H and ^{15}N dimensions. These multiple sources of variance arise from an array of sample conditions, analytical conditions, experimental parameters, and spectral artefacts that can each contribute a difference source of variation to a peak's position, i.e. center.

AutoAssign, an automated resonance assignment software for solution-state NMR, was the first automated protein resonance assignment tool to provide the ability to register different peak lists, extract peak list quality statistics, and offset registration values necessary to align a set of peak lists against a specified reference peak list [58], [88]. The more recently developed Peakmatch algorithm can also match a set of peak lists against a reference peak list and derive offset values using a complete grid search or downhill simplex optimization [89]. Both AutoAssign's registration algorithm and the Peakmatch algorithm work in pairwise mode, i.e. they match a target peak list against a reference peak list, but they are both unable to derive statistics necessary to group peaks into spin systems within a single peak list with more than one peak per spin system (e.g. HN(CO)CACB, NCACX, CANCOCX). While single peak list registration functionality is not required to group peaks into spin systems, it facilitates the development of new grouping algorithms that use a bottom-up approach in grouping peaks into spin systems. In other words, single peak list registration can facilitate the creation of more accurate spin system groups from more reliable smaller variance peak lists first and then extend those spin systems across spectra using pairwise registration statistics derived from pairwise alignment of two different peak lists.

Therefore, a new registration algorithm that can calculate dimension-specific peak position statistics for a single peak list with multiple peaks per spin system was developed. This self-registration mode is accomplished by aligning the single peak list against itself ignoring same-peak matches in order to calculate these dimension-specific peak positional variances. This new registration algorithm provides the necessary statistics to allow inter-peak-list peak grouping and to assess the peak positional uncertainty of individual peak lists.

5.2.3 Application of registration algorithm in grouping algorithm

Since peak positions have multiple sources of variance which are difficult to handle with uniform match tolerances, a new iterative grouping algorithm that combines the peak list registration algorithm with an adaptation of the density-based spatial clustering of applications with noise (DBSCAN) clustering algorithm [90] normalized by dimension-specific peak position variances was developed. This combined algorithm is capable of grouping peaks from a single peak list into spin systems using different sets of match tolerances derived from the new registration algorithm in an iterative analysis.

5.2.4 Algorithm for generating simulated peak lists

A related problem is the limited number of assigned experimental peak lists available in the public repositories for the robust evaluation of computational NMR analysis algorithms and methods. As of July, 2017, the Biological Magnetic Resonance Data Bank (BMRB) [15] contains only a few hundred assigned peak lists from a wide variety of NMR experiments. In order to utilize these assigned peak lists for software tool evaluation, they need to be extracted and converted into appropriate file formats (e.g. Sparky [91], [92], AutoAssign, Xeasy [93], etc.). Also, thorough robustness analysis requires thousands of assigned peak lists for the rigorous testing of algorithms and methods. To provide the necessary datasets, simulated assigned peak lists can be derived from assigned protein resonance assignment entries in the BMRB. However, the simulation of assigned peak lists that provide the same level of difficulty as real experimental peak lists is difficult to generate. Historically, few published methods have been evaluated with simulated peak lists incorporating even a single source of variance. One published evaluation of protein resonance assignment methods even used simulated peak lists with no variance added, representing a very unrealistic test of performance [50].

To address these and related NMR-STAR file utilization problems, the nmrstarlib package [94], an open source library that can be used to extract experimental peak list data from BMRB entries and convert them into peak lists of appropriate format (e.g. Sparky) was used. In addition, a peak list simulator that can create peak lists of different types using the entire BMRB was implemented, allowing the creation of large number of simulated assigned peak lists that includes dimension-specific noise from multiple sources of variance as specified by the user. This new peak list simulator is part of the nmrstarlib package [94].

5.3 Materials and methods

5.3.1 Experimental data sets

The combined registration and grouping algorithm was evaluated using 16 different experimental peak lists from 13 different proteins: 10 peak lists were derived from solution NMR experiments and 6 peak lists were derived from solid-state NMR experiments (**Table 11**). Peak lists usually contain chemical shift values for each dimension that correspond to a specific pattern in a specific NMR experiment and may contain additional information such as peak intensity, line width, and peak volume.

Table 11. Solution-state and solid-state NMR derived peak lists.

Protein	Sequence length	Spectrum type	NMR type	BMRB ID / PDB ID
Bovine pancreatic trypsin inhibitor (BPTI)	58	HN(CO)CACB	Solution-state	5359 / 5PTI
Cold shock protein (CspA) [95]	70	HN(CO)CACB	Solution-state	4296 / 3MEF
Protein yggU from *E.coli* (Target ER14) [96]	108	HN(CO)CACB	Solution-state	5596 / 1N91
Fibroblast growth factor (FGF) [97]	154	HN(CO)CACB	Solution-state	4091 / 1BLD
30S ribosomal protein S28E from Pyrococcus horikoshii (Target JR19) [98]	82	HN(CO)CACB	Solution-state	5691 / 1NY4
Non-structural protein 1 (NS1) [99]	73	HN(CO)CACB	Solution-state	4317 / 1NS1
Ribonuclease pancreatic (RnaseC6572S) [100]	124	HN(CO)CACB	Solution-state	4032 / 1SRN
Ribonuclease pancreatic (RnaseWT) [100]	124	HN(CO)CACB	Solution-state	4031 / 1SRN
Z domain of staphylococcal protein A [101]	71	HN(CO)CACB	Solution-state	5656 / 1H0T
Staphylococcus aureus protein SAV1430 (Target ZR18) [102]	91	HN(CO)CACB	Solution-state	5844 / 1PQX
β1 immunoglobulin binding domain of protein G (GB1) [103]	56	CANCOCX	Solid-state	15156 / 2JSV
β1 immunoglobulin binding domain of protein G (GB1) [103]	56	NCACX	Solid-state	15156 / 2JSV
β1 immunoglobulin binding domain of protein G (GB1) [103]	56	NCOCX	Solid-state	15156 / 2JSV
Disulfide bond formation protein B (DsbB) [104]	176	NCACX	Solid-state	18493 / 2LTQ
Cytoskeleton-associated protein-glycine-rich domains (CAP-Gly) [105]	89	NCACX	Solid-state	19025 / 2M02
Cytoskeleton-associated protein-glycine-rich domains (CAP-Gly) [105]	89	NCOCX	Solid-state	19025 / 2M02

5.3.2 Simulated data sets

Simulated HN(CO)CACB peak lists were generated using the peak list simulation algorithm. For HN(CO)CACB peak lists, every amino acid in the protein sequence not followed by a proline residue should produce two peaks per spin system, except for glycine residues due to missing CB resonances. Initially, 6,896 "ideal" (0-variance) peak lists were generated. Then peak lists that had exact duplicate peaks in all three dimensions were filtered out, because it will create spin systems with more than two peaks per spin system and mark those spin systems as overlapping. Next, peak lists that had missing chemical shift values for CA or CB except for glycine residues were removed. Finally, 2,549 peak lists remained after removing peak lists with duplicate peaks or missing data. Using these remaining peak lists, additional peak lists were simulated for single source of variance in all dimensions, two sources of variance in all dimensions, and two sources of variance in N dimension by adding varying amounts of normally-distributed random noise (equation (1)):

$$\boldsymbol{p}(\boldsymbol{x}) = \frac{\mathbf{1}}{\sqrt{\mathbf{2}\boldsymbol{\pi}\boldsymbol{\sigma}^{\mathbf{2}}}} \boldsymbol{e}^{-\frac{(\boldsymbol{x}-\boldsymbol{\mu})^{2}}{2\boldsymbol{\sigma}^{2}}} \tag{1}$$

where μ is mean, and σ is standard deviation. In the case of two sources of variance, 20% of the peaks had noise standard deviation added that is five times larger than 80% of the remaining peaks in each simulated peak list.

5.3.3 Single peak list registration algorithm

Single peak list registration algorithm is based on a previously developed peak list registration algorithm within the automated protein resonance assignment program AutoAssign [58], [88]. The algorithm has similarities to a point pattern match algorithm [106] and a landsat image registration algorithm [107] developed in the 1980's, but solves a more generalized multiple mapping issue than either of those older algorithms. Extensive modifications to the algorithm that includes new functionality and significant improvement in the computational efficiency were made. The new registration algorithm can perform both pairwise-registration of two different peak lists as well as self-registration of a single peak list that has multiple peaks per spin system. In either algorithmic mode, the registration algorithm operates on two peak lists: an "input" peak list and a "root" or reference peak list. The algorithm calculates the best mapping of peaks from the "input" peak list to peaks in the "root" peak list for their comparable spectral dimensions to derive offsets needed to translate the "input" peak list to the "root" peak list in these comparable dimensions. The algorithm also calculates the standard deviation between mapped pairs of peaks in their comparable dimensions. The self-registration mode of the algorithm treats a single peak list as both the "input" and "root" peak lists and then calculates the best mapping of peaks assuming zero translation offsets and ignoring perfect matches due to self-mapping.

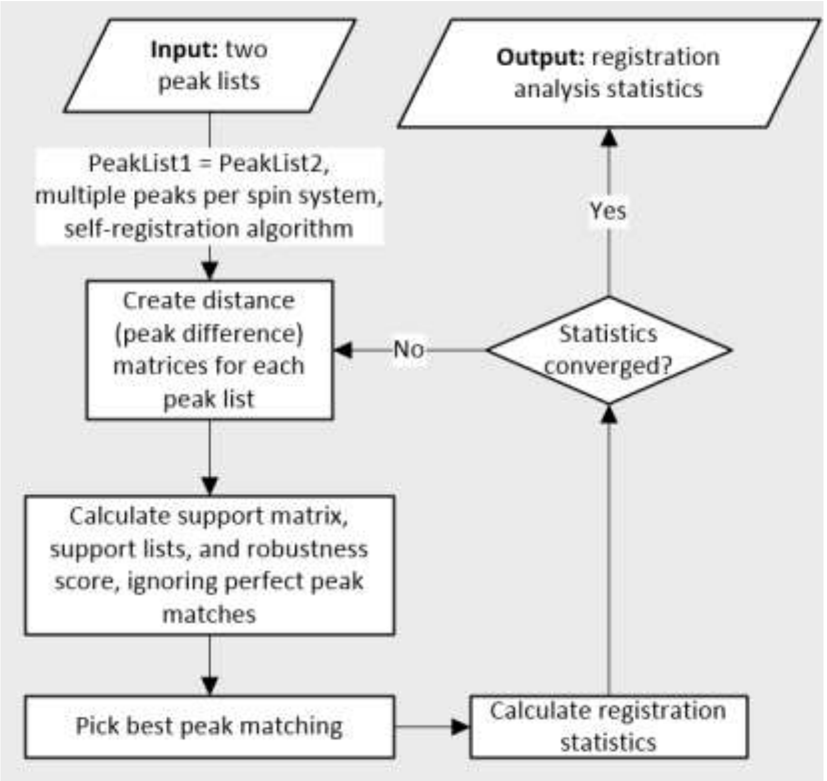

Input: two peak lists
Output: registration analysis statistics
PeakList1 = PeakList2, multiple peaks per spin system, self-registration algorithm
Yes
Create distance (peak difference) matrices for each peak list
No
Statistics converged?
Calculate support matrix, support lists, and robustness score, ignoring perfect peak matches
Pick best peak matching
Calculate registration statistics

Figure 29. Flow diagram of the single peak list registration algorithm.

Figure 29 shows the flow diagram of the new registration algorithm for self-registration execution mode. First, the algorithm parses two peak list files (i.e. the same peak list file twice for self-registration). Then for each peak list, the algorithm constructs a Euclidean distance matrix, i.e. calculates the distance between every pair of peaks within a peak list. Next, the algorithm creates a support matrix and compares each "input" peak distance matrix row to each "root" peak distance matrix row in order to calculate the set of supporting peak mapping pairs, i.e. the support set (SS). Each cell in the support matrix has a set of support pairs $(m,n) \in SS_{i,j}$, i.e. pairs of indices that identify individual coordinates in the support matrix. Using the pair of indices, a corresponding support set can be identified. Using the support pairs in the support sets, the robustness score for a given support pair (i, j) is calculated using a sum of Jaccard similarity coefficients (Jaccard indices) multiplied by corresponding peak difference matching probabilities as illustrated in equation (2):

$$\boldsymbol{robustness(i,j)} = \sum_{(\boldsymbol{m},\boldsymbol{n})\in \boldsymbol{SS}_{\boldsymbol{i},\boldsymbol{j}}} \frac{\boldsymbol{SS}_{\boldsymbol{i},\boldsymbol{j}} \cap \boldsymbol{SS}_{\boldsymbol{m},\boldsymbol{n}}}{\boldsymbol{SS}_{\boldsymbol{i},\boldsymbol{j}} \cup \boldsymbol{SS}_{\boldsymbol{m},\boldsymbol{n}}} \cdot \boldsymbol{p}_{\chi^2_{\boldsymbol{df}}(\boldsymbol{i},\boldsymbol{j},\boldsymbol{m},\boldsymbol{n})} \tag{2}$$

where i, j are the row and column coordinates of the support matrix, m, n are the row and column coordinates of the support matrix whose pair (m, n) is an element of $SS_{i,j}$, and $p_{\chi^2,df}$ is the chi-square probability calculated for corresponding peak differences in the “input” and “root” peak lists for specified degrees of freedom df, i.e. as defined by equation (3):

$$\boldsymbol{\chi^2_{df}(i,j,m,n)} = \sum_{\boldsymbol{l}=\boldsymbol{0}}^{\boldsymbol{df}} \left(\frac{(\boldsymbol{input\,peak\,list}_{\boldsymbol{i}}[\boldsymbol{l}] - \boldsymbol{input\,peak\,list}_{\boldsymbol{m}}[\boldsymbol{l}]) - (\boldsymbol{root\,peak\,list}_{\boldsymbol{j}}[\boldsymbol{l}] - \boldsymbol{root\,peak\,list}_{\boldsymbol{n}}[\boldsymbol{l}])}{\boldsymbol{std}[\boldsymbol{l}] \cdot \boldsymbol{2}} \right)^{\boldsymbol{2}} \tag{3}$$

where l specifies the index of the comparable dimension of a peak in both the “input” and “root” peak lists and their corresponding standard deviation std. A supporting peak mapping pair is determined by a match tolerance defined in terms of standard deviation units. The default is four standard deviation units. The self-registration execution mode excludes identical peak mappings from this comparison. Using the support list, a robustness score is calculated for each comparison. The robustness score indicates how many peaks in the “input” peak list are mapped to corresponding peaks in “root” peak list in a concordant manner (i.e. below match tolerances) with a single mapping peak-pair representing the center of the concordance. The higher the robustness score, the larger the concordance. Next, the algorithm uses the support list of the peak mapping pair with the best robustness score to calculate the registration offsets and statistics, which is used to derive new match tolerances. The algorithm iterates until the statistics of registration converge, i.e. until per dimension standard deviations stop changing.

One detail to note in equation (3) is the use of $std[l] \cdot 2$ in calculating the chi-square statistic. Based on linear error analysis and independent variable propagation rules, one would expect $std[l] \cdot \sqrt{2}$ to be the correct estimate of the standard deviation to use in this equation. However, in this iterative registration approach, $std[l] \cdot 2$ provides superior performance. I believe that the use of 2 instead of $\sqrt{2}$ accounts for non-independent error propagation in the given difference of differences analysis.

5.3.4 Single peak list grouping algorithm

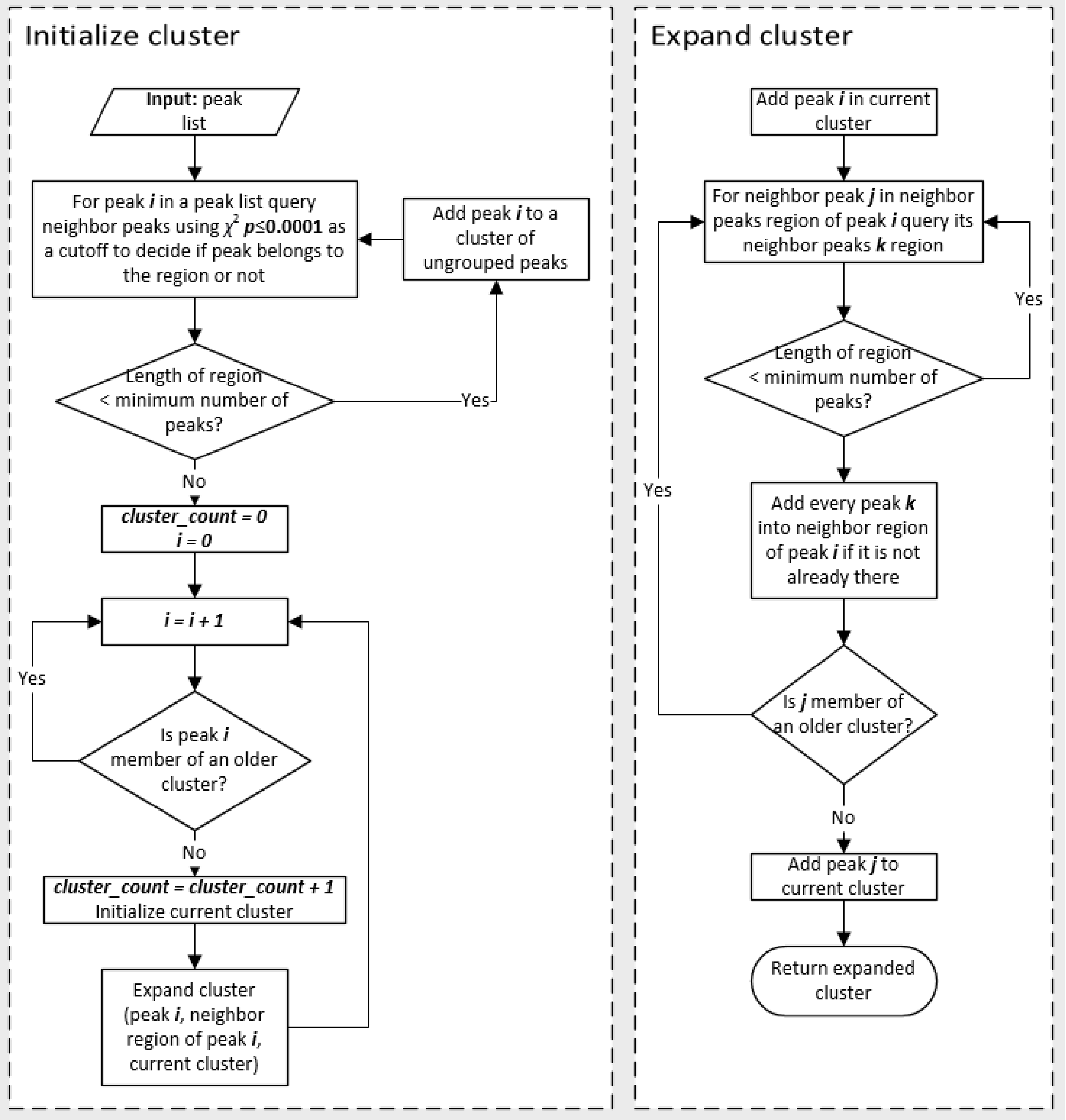
Single peak list spin system grouping algorithm is based on the widely-used density-based clustering algorithm DBSCAN [90], which can detect clusters of varying size and shape. The original DBSCAN algorithm requires two global parameters: radius ε, which defines the ε-neighborhood of a point and the minimum number of points μ that can form a cluster. The DBSCAN algorithm uses a region query similarity function to initialize clusters where it calculates the Euclidean distance between core point and every other point in the data and function that expands cluster by examining neighborhoods of points in the initialized cluster in order to discover cluster points [90].

In this case, each peak represents a point in a peak list data and in order to group peaks into clusters (spin systems) without overlap or split, we would have to know the radius ε for each of the clusters in advance. For peak list data, it is not easy to know those parameters in advance and requires domain expert to identify tolerances needed for grouping peaks into spin systems (clusters). This is further complicated by the presence of multiple sources of variance affecting subsets of peaks within a single peak list, i.e. some peaks will require larger tolerances for grouping them into spin systems than others. Therefore, uniform tolerances cannot be used to discover optimal peak grouping.

For the grouping algorithm, the region query function that uses the neighborhood radius ε and the Euclidean distance similarity function was replaced with version that uses a chi-square distance cutoff and variance-normalized distance (chi-square value) to decide if a peak can be included into a spin system cluster or not. Equation (4) describes the criteria for inclusion or exclusion of peaks from the initialized spin system cluster:

$$\begin{cases} \sqrt{\sum_{k=0}^{df}\left(\frac{peak_i[k]-peak_j[k]}{std[k]}\right)^2} \leq \sqrt{F^{-1}(p,df)} \\ \sqrt{\sum_{k=0}^{df}\left(\frac{peak_i[k]-peak_j[k]}{std[k]}\right)^2} > \sqrt{F^{-1}(p,df)} \end{cases} \tag{4}$$

where $peak_i$ and $peak_j$ is every pair of peaks within a single peak list, df – number of degrees of freedom that correspond to the number of comparable dimensions, k – specifies index of comparable dimension within a peak and its corresponding standard deviation std obtained from the registration algorithm, $F^{-1}(p, df)$ – chi-square inverse cumulative distribution function for a given p-value and degrees of freedom. If the normalized distance between peaks is less than or equal to the inverse survival function for a given p-value and corresponding degrees of freedom, the peak belongs to the spin system cluster, otherwise the peak is excluded from the spin system cluster. The variances used to calculate the normalized distance are supplied by the self-registration algorithm. The use of a chi-square value allows the cutoff parameter to be provided in terms of a chi-square probability. The default for the algorithm is a p-value ≤ 0.0001.



Initialize cluster
Input: peak list
For peak i in a peak list query neighbor peaks using χ² p≤0.0001 as a cutoff to decide if peak belongs to the region or not
Add peak i to a cluster of ungrouped peaks
Length of region < minimum number of peaks?
Yes
No
cluster_count = 0
i = 0
i = i + 1
Yes
Is peak i member of an older cluster?
No
cluster_count = cluster_count + 1
Initialize current cluster
Expand cluster (peak i, neighbor region of peak i, current cluster)
Expand cluster
Add peak i in current cluster
For neighbor peak j in neighbor peaks region of peak i query its neighbor peaks k region
Yes
Length of region < minimum number of peaks?
Add every peak k into neighbor region of peak i if it is not already there
Yes
Is j member of an older cluster?
No
Add peak j to current cluster
Return expanded cluster

Figure 30. Flow diagram of the single peak list grouping algorithm.

Figure 30 shows the flow diagram of the peak grouping algorithm that groups peaks within a single peak list into spin systems. The grouping algorithm consists of two main functions – one that initializes the clusters and the other that expands clusters by examining the neighborhood of an initialized cluster in a similar fashion to DBSCAN [90].

5.3.5 Combined single peak list registration and grouping algorithm

In order to address the presence of multiple sources of peak positional variance, an iterative algorithm that combines both the self-registration algorithm and grouping algorithm to derive spin system clusters using multiple variance-based match tolerances calculated with the help of the registration algorithm was developed. **Figure 31** shows the flow diagram of the combined algorithm.

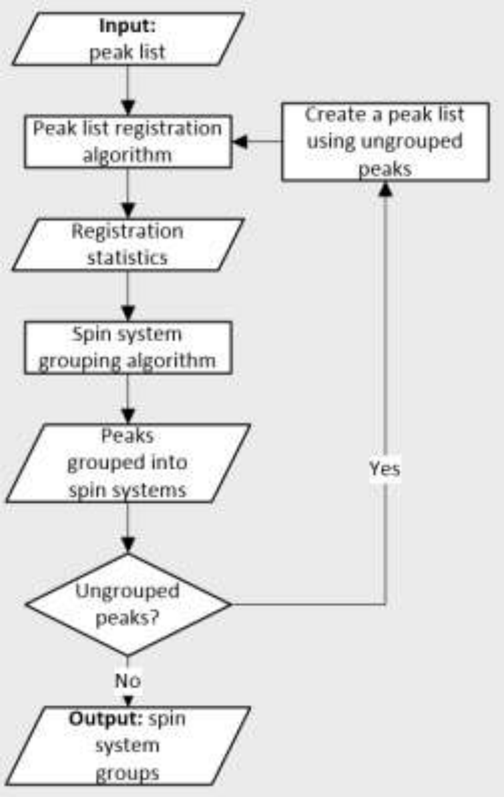

Input: peak list
Peak list registration algorithm
Create a peak list using ungrouped peaks
Registration statistics
Spin system grouping algorithm
Peaks grouped into spin systems
Yes
Ungrouped peaks?
No
Output: spin system groups

Figure 31. Flow diagram overview of the entire registration and grouping process.

First, the combined algorithm reads a single peak list in and runs the self-registration algorithm to identify initial variance values for each comparable dimension.

Next, the grouping algorithm uses per dimension variance values to group peaks into spin system clusters. Then, the combined algorithm checks if there are unclustered peaks left. From the unclustered peaks, the algorithm creates a new peak list file and attempts to register it against itself again to determine new larger variances that can be used to group peaks into spin system clusters.

5.3.6 Peak list simulation algorithm

To create additional data sets for robustness analysis, an algorithm that can simulate peak lists using assigned chemical shift values deposited in BMRB entries was developed. This algorithm is implemented as a peak list simulator submodule within the nmrstarlib Python package [94], which facilitates the reading and writing of NMR-STAR formatted files, especially entry files maintained by BMRB. This algorithm uses the nmrstarlib functionality to access assigned chemical shift values for H, C and N resonances for each residue in a protein chain and then saves them as a peak list file in different formats (e.g. Sparky, AutoAssign, JSON). Moreover, the algorithm provides the ability to add varying amounts of noise to each dimension of the peak list in order to create more realistic data sets. The peak list simulator uses a very generic spectrum definitions based on different resonance classes (e.g. CA, CB, N, etc.) and their relative positions (-1, 0, +1, etc.), therefore different through-bond experiments can be described for both solution and solid-state NMR spectra very easily. The local contact peaks for through-space experiments can be simulated as well using the relative position descriptions (0, +1, +2, +3, +4).

5.4 Results and discussion

5.4.1 Performance on experimental data sets

First, the performance of combined registration and grouping algorithm on manually assigned peak lists derived from solution and solid-state NMR experiments was evaluated. **Table 12** shows the summary of results for peak lists derived from solution NMR HN(CO)CACB type experiments [108]. The expected number of peaks for the HN(CO)CACB peak list can be estimated from a protein sequence, i.e. for every spin system in a protein there should be at least two peaks except for glycine (due to missing CB resonance) and proline (due to missing amide H resonance) residues ([number of amino acids in sequence – number of prolines – number of glycines]×2 + number of glycines – 1). Similarly, the expected number of spin systems (clusters) for the HN(CO)CACB peak list can be estimated from a known sequence (number of amino acids in sequence – 1 – number of GLY residues – number of PRO residues). The number of observed peaks is usually larger than the number of expected peaks for a given protein sequence due to NMR artefacts and the presence of multiple conformations with slow exchange. The number of ungrouped peaks shows how many peaks were left ungrouped after the iterative registration and grouping procedure. This number is proportional to number of glycine residues (because of a missing corresponding peak for CB resonance) in the protein sequence, and the number of artefact peaks that appear in the spectrum. The numbers of missing, overlapped, and split spin systems were inferred directly from the assigned peak lists. For example, a split in spin systems occurs when two peaks that should form their own spin system cluster end up being added into other neighbor spin system clusters. Results of the iterative grouping algorithm summarized in **Table 12** show that it is capable of grouping peaks into spin system clusters that correspond to real spin systems in a protein sequence. When the grouping algorithm was limited to a single registration-grouping iteration, the number of identified clusters decreased dramatically (see **Table 12** value in parenthesis) ranging from 13% fewer recovered clusters for the 30S ribosomal protein (BMRBID 5691) to 57% fewer recovered clusters for pancreatic ribonuclease (BMRBID 4032).

Table 12. Spin system grouping results for solution-state NMR derived peak lists using combined registration and grouping algorithm.

Protein / Peak list	Expected peaks	Observed peaks	Ungrouped peaks	Expected spin systems	Identified spin systems*	Missing spin systems	Overlapped spin systems	Split spin systems
BPTI / HN(CO)CACB	101	134	17	47	54 (30)	0	0	2
CSP / HN(CO)CACB	125	145	39	57	53 (32)	12	0	0
ER14 / HN(CO)CACB	194	181	7	93	87 (57)	8	2	0
FGF / HN(CO)CACB	273	303	24	128	139 (112)	13	2	1
JR19 / HN(CO)CACB	151	141	7	71	67 (58)	4	0	0
NS1 / HN(CO)CACB	137	203	36	66	81 (43)	26	8	2
RnaseC6572S / HN(CO)CACB	235	282	16	116	130 (56)	18	4	2
RnaseWT / HN(CO)CACB	235	403	19	116	181 (122)	9	2	1
ZDOM / HN(CO)CACB	134	153	29	67	55 (40)	15	3	5
ZR18 / HN(CO)CACB	172	163	3	85	80 (52)	5	0	0

* Value in parenthesis shows how many spin systems were identified if only uniform tolerances were used and single iteration of grouping algorithm was performed.

Table 13 contains similar summary results for solid-state NMR derived peak lists. CANCOCX [109], NCACX [30], and NCOCX [30] peak lists for the GB1 protein were nearly complete and therefore showed low number of overlapped and split spin systems. Peak lists for the DsbB and Cap-Gly proteins had a large number of missing and artefact peaks, therefore a higher number of overlapped and split spin systems were observed. The quality of peak list registration and therefore spin system grouping is highly correlated with the quality of peak lists. Also, the larger the number of missing and artefact peaks in the peak lists, the larger the overlap in spin systems that were generally observed. Similar to solution-state NMR derived peak lists, the algorithm was limited to a single registration-grouping iteration. However, the solid-state NMR derived peak lists were more consistent and did not have as much dimension-specific variance in comparison to solution-state NMR derived peak lists (see **Table 13** value in parenthesis). This may seem surprising, given the typical lower spectral quality of solid-state NMR spectra in comparison to solution NMR spectra in terms of sensitivity and peak widths. However, when good quality solid-state NMR spectra are obtainable, the greater spread of peaks across the ^{15}N and ^{13}C dimensions used for grouping provides advantages over the more crowded amide ^{1}H and ^{15}N dimensions used for grouping in solution NMR spectra.

Table 13. Spin system grouping results for solid-state NMR derived peak lists using combined registration and grouping algorithm.

Protein / Peak list	Expected peaks*	Observed peaks	Ungrouped peaks	Expected spin systems	Identified spin systems**	Missing spin systems	Overlapped spin systems	Split spin systems
GB1 / CANCOCX	268	240	70	55	56 (56)	1	6	28
GB1 / NCACX	268	463	62	55	65 (65)	0	0	19
GB1 / NCOCX	268	474	16	55	82 (67)	0	4	10
DsbB / NCACX	940	215	43	175	47 (47)	126	14	1
CapGly / NCACX	410	515	16	88	50 (50)	33	25	0
CapGly / NCOCX	410	218	25	88	47 (47)	38	32	5

* Number of expected peaks estimated based on magnetization transfer pattern and amino acid sequence. Alternative magnetization transfer pathways increase the number of peaks present.

** Value in parenthesis shows how many spin systems were identified if only uniform tolerances were used and single iteration of grouping algorithm was performed.

The best and worst spin system grouping results are visualized on **Figure 32**: panel **a)** shows the best grouping result for solution NMR derived peak lists – clean non-overlapped clusters with a small number of artifact peaks; panel **b)** shows the worst result for solution NMR derived peak lists, which has more overlap and more artifact peaks; panels **c)** and **d)** show the best and worst results for solid-state NMR peak lists, with more artifact peaks observed in comparison to solution NMR peak lists and significantly higher overlap due to the lower quality of the peak lists.

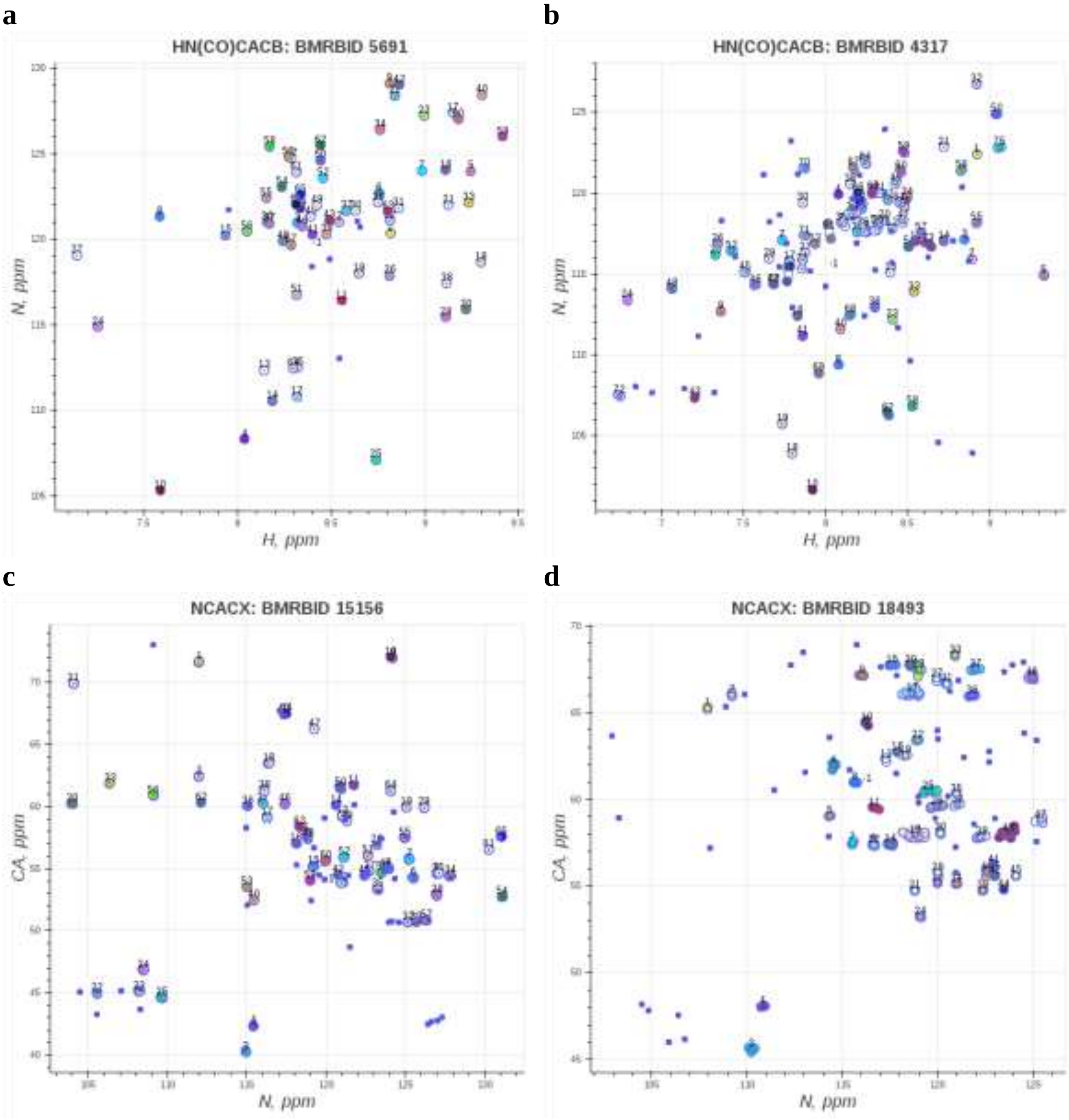


Figure 32. Visualization of spin system grouping results where colored points correspond peak centers grouped into spin systems, peak centers of the same color belong to the same spin system (spin systems are numbered sequentially), unnumbered blue points correspond to either spurious unassigned peaks or in case of HN(CO)CACB peak lists peaks corresponding to glycine residues (due to missing CB resonance): **a)** example of best spin system clustering for 30S ribosomal protein S28E from *Pyrococcus horikoshii* (HN(CO)CACB peak list); **b)** example of worst spin system clustering non-structural protein 1 (HN(CO)CACB peak list); **c)** example of best spin system clustering for GB1 protein (NCACX peak list); **d)** example of worst spin system clustering for DsbB protein (NCACX peak list).

5.4.2 Performance on simulated data sets

To evaluate robustness of algorithms, large numbers of simulated HN(CO)CACB peak lists were generated (see **Table 14**). To create peak lists that better reflect what is observed in experimental peak lists, varying amounts of noise were introduced based on random normal distributions for several conditions: i) single source of variance in all dimensions; ii) two sources of variance in all dimensions; iii) two sources of variance in one dimension.

Table 14. Summary on simulated HN(CO)CACB peak lists.

Number of variance sources	Minimum standard deviation values	Maximum standard deviation values	Total number of simulated peak lists
Single source of variance in all dimensions	H: 0.001 C: 0.01 N: 0.01	H: 0.050 C: 0.50 N: 0.50	127,450
Two sources of variance in all dimensions	H: 0.001, 0.005 C: 0.01, 0.05 N: 0.01, 0.05	H: 0.010, 0.050 C: 0.10, 0.50 N: 0.10, 0.50	25,490
Two sources of variance in N dimension, single source of variance in C and H dimensions	H: 0.001 C: 0.01 N: 0.01, 0.05	H: 0.010 C: 0.10 N: 0.10, 0.50	25,490

Figure 33 demonstrates results for the single source of variance condition, where peak lists were simulated with increasing random noise from 0.001 ppm to 0.050 ppm for ^{1}H dimension and from 0.01 ppm to 0.50 ppm for ^{13}C and ^{15}N dimensions. The percentage of accurately grouped peaks versus percentage of overlapped peaks are plotted as a function of dimension-specific standard deviations. The red vertical line separates high quality versus low quality peak lists with larger peak positional variance and overlap. Normally, good quality peak lists have ^{1}H, ^{13}C, and ^{15}N chemical shift standard deviations on the left side of the red line. It is clear from the diagram that for the smallest variance in peak positions, the algorithm groups 99% of peaks into correct non-overlapped spin systems across all simulated peak lists. As variance in peak positions increases percentage of overlapped peaks increases. At larger dimension-specific

variance condition (0.01 for ^{1}H dimension and 0.1 for ^{13}C and ^{15}N dimensions), it is still capable of grouping 77% of peaks into clean non-overlapped spin systems.

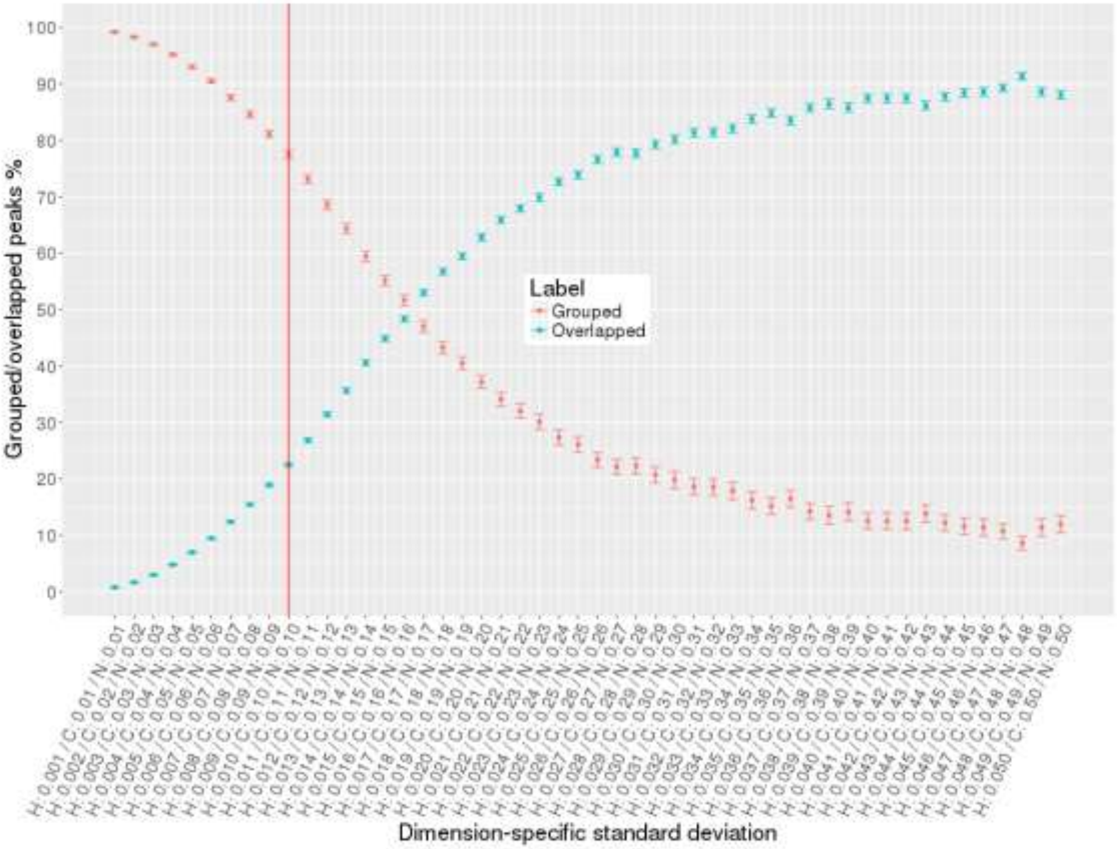


Figure 33. Single source of variance in all dimensions: percentage of grouped (non-overlapped) and overlapped peaks with increase in standard deviation values of peak dimensions. The dots correspond to the percentage of the grouped/overlapped peaks, whiskers are calculated standard error of the mean.

Figure 34 shows similar results but for two sources of variance in all dimensions, i.e. 80% of peaks had random normal noise added from 0.001 ppm to 0.01 ppm for ^{1}H dimension and from 0.01 ppm to 0.1 ppm for ^{13}C and ^{15}N dimensions, the remaining 20% of peaks had random normal noise five times higher (from 0.005 ppm to 0.05 ppm for ^{1}H dimension and from 0.05 ppm to 0.5 ppm for ^{13}C and ^{15}N dimensions).

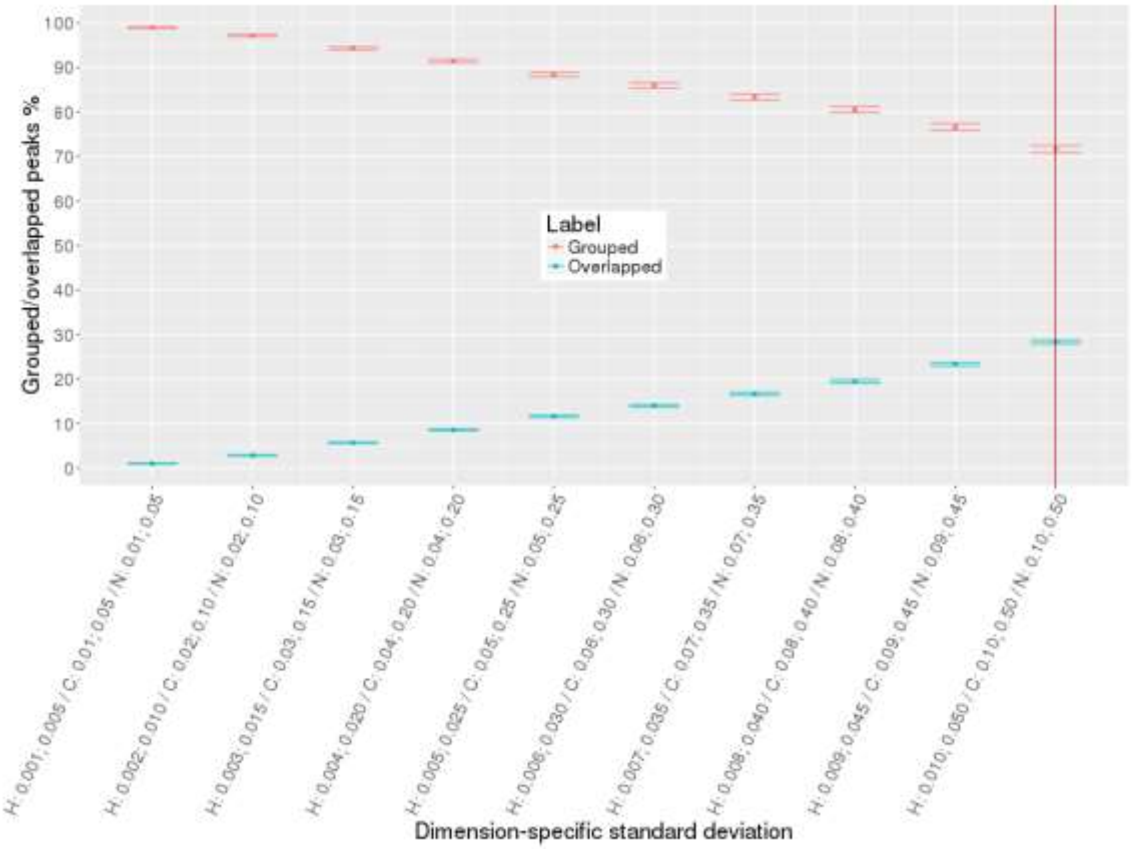


Figure 34. Two sources of variance in all dimensions: percentage of grouped (non-overlapped) and overlapped peaks with increase in standard deviation values of peak dimensions, 20% of peaks have five times larger variance than the remaining 80% of peaks in all dimensions. The dots correspond to the percentage of the grouped/overlapped peaks, whiskers are calculated standard error of the mean.

Figure 35 shows results for the case were ^{15}N dimension had two sources of variance, and ^{1}H and ^{13}C dimensions had only one source of variance. Results in **Figure 34** and **Figure 35** demonstrate that the iterative grouping algorithm can handle peak lists with multiple sources of variance in single or all dimensions and can group 99% of peaks for the smallest variance values in peak dimensions and 71% of peaks at the 0.01 ^{1}H chemical shift standard deviation level.

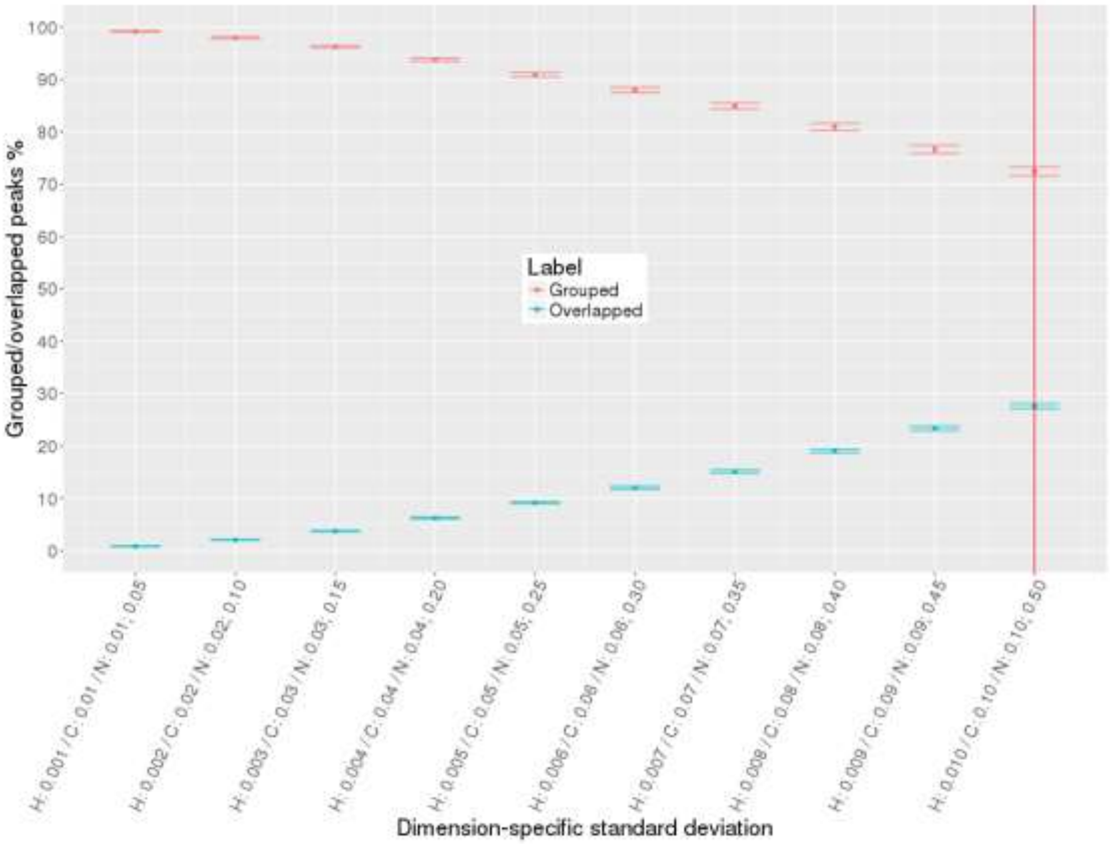


Figure 35. Two sources of variance in one dimension: percentage of grouped (non-overlapped) and overlapped peaks with increase in standard deviation values of peak dimensions, 20% of peaks have five times larger variance than the remaining 80% of peaks in N dimension. The dots correspond to the percentage of the grouped/overlapped peaks, whiskers are calculated standard error of the mean.

5.4.3 Comparison to hierarchical DBSCAN algorithm

In order to test if other clustering algorithms can be used to group peaks within single peak list into spin system clusters, a recently developed variation of DBSCAN called hierarchical DBSCAN (HDBSCAN) was used [110], [111]. This clustering algorithm was chosen, because it has several advantages over other clustering algorithms: it does not require the expected number of clusters upfront (as opposed to k-means) and it does not require specification of the ε-neighborhood parameter (as opposed to the regular

DBSCAN clustering algorithm). This hierarchical version performs DBSCAN using varying values of radius ε and integrates all results to find the best clustering solution.

Table 15. Spin system grouping results for solution-state NMR derived peak lists using HDBSCAN algorithm.

Protein / Peak list	Expected peaks	Observed peaks	Ungrouped peaks	Expected spin systems	Identified spin systems	Missing spin systems	Overlapped spin systems	Split spin systems
BPTI / HN(CO)CACB	101	134	15	47	24	0	31	0
CSP / HN(CO)CACB	125	145	37	57	21	12	35	1
ER14 / HN(CO)CACB	194	181	33	93	26	8	77	1
FGF / HN(CO)CACB	273	303	43	128	53	13	108	3
JR19 / HN(CO)CACB	151	141	18	71	23	4	66	3
NS1 / HN(CO)CACB	137	203	49	66	31	26	43	8
RnaseC6572S / HN(CO)CACB	235	282	38	116	45	18	90	4
RnaseWT / HN(CO)CACB	235	403	68	116	68	9	75	9
ZDOM / HN(CO)CACB	134	153	22	67	25	15	49	5
ZR18 / HN(CO)CACB	172	163	42	85	22	5	59	0

Table 15 shows results of HDBSCAN for solution NMR peak lists. The number of overlapping spin systems was significantly higher in comparison to combined registration and grouping algorithm implementation. Also, for solid-state NMR derived peak lists, HDBSCAN performed slightly worse (see **Table 16**). The implementation of iterative registration and grouping algorithm is slower than HDBSCAN due to the complexity of the registration algorithm step, but it produces more accurate and more consistent results for both solution and solid-state NMR derived experimental peak lists as well as for simulated peak lists.

Table 16. Spin system grouping results for solid-state NMR derived peak lists using HDBSCAN algorithm.

Protein / Peak list	Expected peaks*	Observed peaks	Ungrouped peaks	Expected spin systems	Identified spin systems	Missing spin systems	Overlapped spin systems	Split spin systems
GB1 / CANCOCX	268	240	16	55	51	1	29	9
GB1 / NCACX	268	463	14	55	63	0	2	1
GB1 / NCOCX	268	474	14	55	67	0	4	7
DsbB / NCACX	940	215	27	175	37	126	31	3
CapGly / NCACX	410	515	36	88	70	33	21	17
CapGly / NCOCX	410	218	20	88	42	38	46	7

* Number of expected peaks estimated based on magnetization transfer pattern and amino acid sequence. Alternative magnetization transfer pathways increase the number of peaks present.

5.5 Conclusions

A new peak list registration algorithm was developed. The algorithm is capable of executing in two modes: self-registration and pairwise-registration. Self-registration mode allows the derivation of registration statistics for a single unassigned peak list that has multiple peaks per spin system. Pairwise-registration allows alignment of two different unassigned peak lists in order to calculate registration statistics. Using this self-registration algorithm, a new bottom-up iterative grouping algorithm was developed. This algorithm can group peaks into spin systems within a single peak list and can handle multiple sources of variance that are present within experimental data sets. Utilization of the single peak list registration algorithm will facilitate the development of more sophisticated and automated spin system grouping algorithms that produce more accurate spin systems for downstream data analyses.

Automated tools that allow the creation of simulated peak lists with a range of positional variances using assigned chemical shifts in BMRB entries were developed. These tools were applied for generation of a very large simulated dataset from the entire BMRB to rigorously test the performance and robustness of algorithms. These tests showed that algorithms can detect multiple sources of variance introduced into simulated data sets and reliably group peaks into spin systems for peak lists that are far from ideal.

CHAPTER 6
PAIRWISE REGISTRATION AND GROUPING ALGORITHMS

6.1 Overview

Protein resonance assignment is the first critical step in protein structure determination. A typical protein resonance assignment strategy uses a set of peak lists derived from different types of NMR experiments. This requires an agreement in chemical shift values between different peak lists. Due to chemical shift referencing problems, chemical shift values can become shifted relative to each other, which causes severe problems in spin system grouping and as a result affects all downstream resonance assignment steps. The pair of complimentary pairwise peak list registration and grouping algorithms was developed. These algorithms utilize single peak list registration and grouping algorithms first in order to create global spin systems groups across all peak lists in a bottom-up merge fashion. In other words, the most consistent data is leveraged first in order to create local spin system groups for a single peak list and then grow the spin systems by comparing spin systems groups or individual peaks from different peak lists.

6.2 Introduction

A set of peak lists derived from different types of NMR experiments is required to assign resonances within protein NMR spectra. Both solution-state and solid-state protein NMR assignment strategies require at least three peak lists in order to produce reliable resonance assignments. Prior to the resonance assignment, it is very important to analyze the quality of the peak lists in terms of consistency chemical shift values between different peak lists as well as reliably estimate match tolerance values for grouping peaks into spin systems. Failure to register several different peak lists against each other is a strong indicator of insufficient peak list quality. As a result, such peak lists cannot be used in the resonance assignment process, because we cannot reliably estimate match tolerance values for peaks grouping because of poor peak (chemical shifts) matching between different peak lists. Such problems can result from inconsistent chemical shift referencing during the data acquisition phase, inadequate resolution in a match dimension, or a variety of issues that can arise during the data acquisition or processing phases.

In order to solve inconsistency problems in chemical shift values between different peak lists, a pairwise registration algorithm that can derive the offset values to make one peak list match the other was developed. In addition, the algorithm produces standard deviations per each comparable dimension that are used in the complimentary pairwise grouping algorithm. The pairwise grouping algorithm works in a bottom-up fashion and utilizes the single peak list grouping algorithm first to derive internal groups of peaks within single peak list for peak lists that have more than one peak per spin system. Then starting with the best peak list, i.e. the peak list with the smallest standard deviation values in comparable dimensions, the pairwise grouping algorithm merges peaks from different peak lists into global spin systems.

The pairwise grouping has all the properties of the single peak list grouping algorithm described in Chapter 5, i.e. it works in an iterative fashion and as a result can account for multiple sources of variance. In addition, the pairwise grouping algorithm has rules to detect split spin systems, overlapped spin systems, and missing peaks or spin systems by pairwise comparison of peaks or groups of peaks that are being merged together.

6.3 Materials and Methods

6.3.1 Experimental data sets

The pairwise registration and grouping algorithm was evaluated using a set of different peak lists derived from solid-state NMR spectra of β1 immunoglobulin binding domain of protein G (GB1) (**Table 17**).

Table 17. The solid-state NMR derived peak lists for pairwise algorithm testing.

Protein	Sequence length	Spectrum type	NMR type	BMRB ID / PDB ID
β1 immunoglobulin binding domain of protein G (GB1) [103]	56	CAN(CO)CA	Solid-state	15156 / 2JSV
β1 immunoglobulin binding domain of protein G (GB1) [103]	56	NCACX	Solid-state	15156 / 2JSV
β1 immunoglobulin binding domain of protein G (GB1) [103]	56	CANCOCX	Solid-state	15156 / 2JSV

6.3.2 Pairwise peak list registration algorithm

The single peak list registration algorithm described in Chapter 5 and pairwise registration algorithms are implemented within the same code base and have a common command-line interface. The difference is that the single peak list operates on single peak list and pairwise algorithm operates on two different peak lists (an “input” peak list and a “root” or reference peak list). Both algorithms calculate the best mapping of peaks from the “input” peak list to peaks in the “root” peak list for their comparable spectral dimensions. The pairwise registration algorithm derives needed offsets from match peaks in “root” peak list to “input” peak list. Based on the registration, the algorithm calculates the standard deviation between mapped pairs of peaks in their comparable dimensions, which can be used as an estimation of match tolerance values for the pairwise grouping. **Figure 36** shows the flow diagram for both single peak list registration and pairwise peak list registration algorithms. If the “input” peak list is identical to the “root” peak list, the self-registration branch of the algorithm executes. If the “input” and “root” peak list are different, the pairwise-registration branch of the algorithm executes.

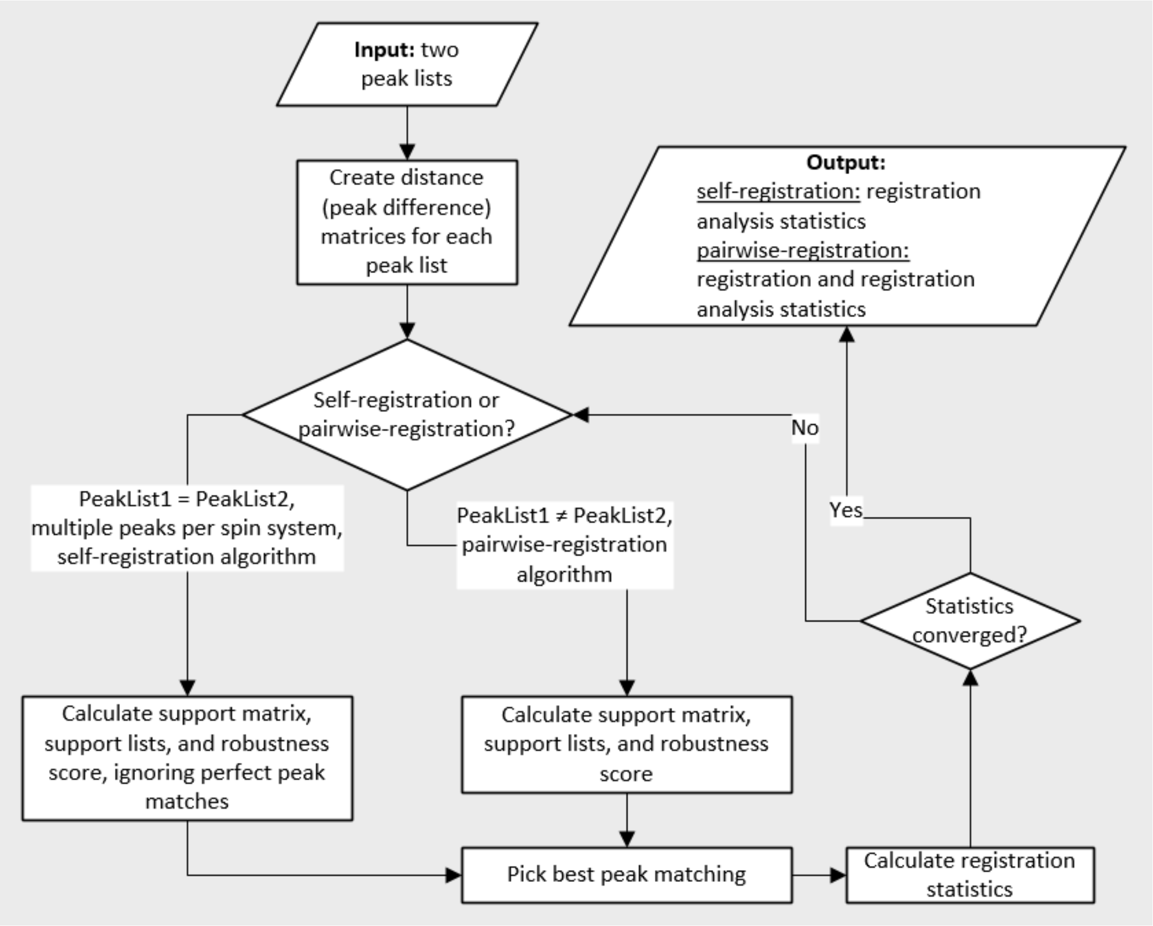

Input: two peak lists
Create distance (peak difference) matrices for each peak list
Output:
self-registration: registration analysis statistics
pairwise-registration: registration and registration analysis statistics
Self-registration or pairwise-registration?
No
PeakList1 = PeakList2, multiple peaks per spin system, self-registration algorithm
PeakList1 ≠ PeakList2, pairwise-registration algorithm
Yes
Statistics converged?
Calculate support matrix, support lists, and robustness score, ignoring perfect peak matches
Calculate support matrix, support lists, and robustness score
Pick best peak matching
Calculate registration statistics

Figure 36. Flow diagram of the combined single peak list registration algorithm and pairwise peak list registration algorithm.

The robustness score is calculated according to equations (2) and (3). The only conceptual difference is that since “root” and “input” peak lists are different, every pairwise peak comparison is allowed. In single peak list registration algorithm only nonidentical comparisons are allowed due to the fact that “root” and “input” peak lists are the same.

To give a specific example, let’s consider four peaks in “peaklist1” and five peaks in “peaklist2” in **Table 18** and Figure 37.

Table 18. Example of two peak lists used in registration algorithm.

Peaklist1			Peaklist2		
#	***CA, ppm***	***N, ppm***	**#**	***CA, ppm***	***N, ppm***
1	54.319	122.274	…	…	…
2	60.062	117.66	11	54.119	121.826
3	57.132	125.585	12	59.848	117.36
4	53.815	124.145	13	56.968	125.285
…	…	…	14	53.615	123.855
…	…	…	15	54.118	118.102
…	…	…	…	…	…

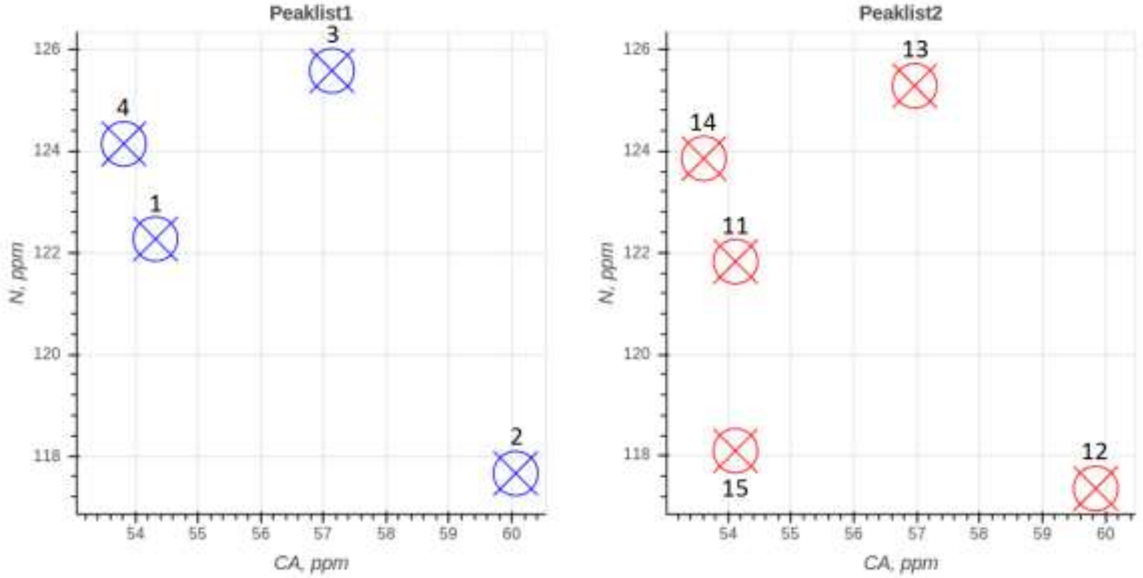


Figure 37. Visualization of “peaklist1” and “peaklist2” used in pairwise registration.

First, for every peak the peak difference matrix is constructed by calculating peak differences for comparable dimensions, for example:

CA: 55.319 – 55.319 = 0
N: 122.274 – 122.274 = 0
CA: 55.319 – 60.062 = –4.743

N: 122.274 – 117.66 = 4.614

…

Table 19 and **Table 20** show the peak difference matrices for “peaklist1” and “peaklist2”, respectively.

Table 19. Peak difference matrix for “peaklist1”.

Peak #	1		2		3		4	
	CA	***N***	***CA***	***N***	***CA***	***N***	***CA***	***N***
1	0	0	−4.743	4.614	−1.813	−3.311	1.504	−1.871
2	4.743	−4.614	0	0	2.93	−7.925	6.247	−6.485
3	1.813	3.311	−2.93	7.925	0	0	3.317	1.44
4	−1.504	1.871	−6.247	6.485	−3.317	−1.44	0	0

Table 20. Peak difference matrix for “peaklist2”.

Peak #	11		12		13		14		15	
	CA	***N***	***CA***	***N***	***CA***	***N***	***CA***	***N***	***CA***	***N***
11	0	0	−5.729	4.466	−2.849	−3.459	0.504	−2.029	0.001	3.724
12	5.729	−4.466	0	0	2.88	−7.925	6.233	−6.495	5.73	−0.742
13	2.849	3.459	−2.88	7.925	0	0	3.353	1.43	2.85	7.183
14	−0.504	2.029	−6.233	6.495	−3.353	−1.43	0	0	−0.503	5.753
15	−0.001	−3.724	−5.73	0.742	−2.85	−7.183	0.503	−5.753	0	0

Next, using peak difference matrix, variance normalized Euclidean distance can be calculated for every pair of peaks for each peak list as shown on **Figure 38**. **Table 21**and

Table 22 show calculated Euclidean distance matrices for “peaklist1” and “peaklist2”, respectively.

$$\boldsymbol{distance = \sqrt{\sum_l \left(\frac{peak\_difference_l}{std_l \cdot 2 \cdot tolerance}\right)^2}},$$

where l is comparable dimension (*CA* or *N*), std is initial standard deviation (0.075 ppm for *C* and *N* dimensions), match $tolerance$ value is set to 4. For example, variance normalized distance between peak #1 and peak #2 is equal to:

$$distance = \sqrt{\left(\frac{-4.743}{0.075 \cdot 2 \cdot 4}\right)^2 + \left(\frac{4.614}{0.075 \cdot 2 \cdot 4}\right)^2} = 11.0283$$

Table 21. Euclidean distance matrix for “peaklist1” (distances $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{f}$).

Peak #	1	2	3	4
1	0	11.0283	6.2914	4.0009
2	11.0283	0	14.0821	15.0074
3	6.2914	14.0821	0	6.0268
4	4.0009	15.0074	6.0268	0

Table 22. Euclidean distance matrix for “peaklist2” (distances $\boldsymbol{a'}, \boldsymbol{b'}, \boldsymbol{c'}, \boldsymbol{d'}, \boldsymbol{e'}, \boldsymbol{f'}, \boldsymbol{g'}, \boldsymbol{h'}, \boldsymbol{i'}, \boldsymbol{j'}$).

Peak #	11	12	13	14	15
11	0	12.1068	7.4687	3.4844	6.2066
12	12.1068	0	14.053	15.003	9.6297
13	7.4687	14.053	0	6.0753	12.8796
14	3.4844	15.003	6.0753	0	9.6249
15	6.2066	9.6297	12.8796	9.6249	0

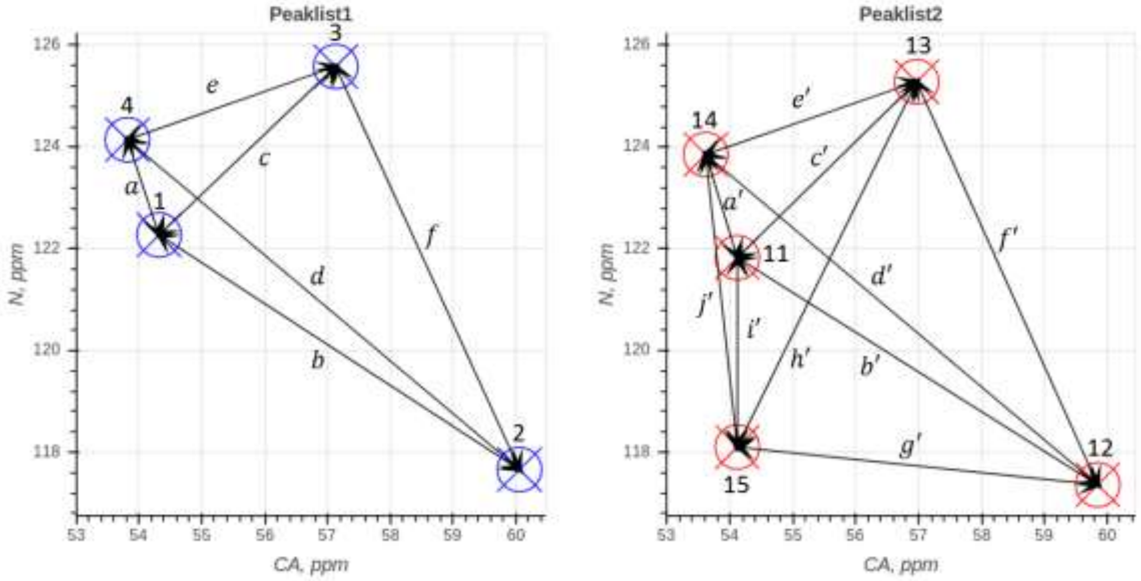


Figure 38. Visualization of distances between every pair of peaks, a, b, c, d, e, f in “peaklist1” and $a', b', c', d', e', f', g', h', i', j'$ in “peaklist2”.

Using the Euclidean distance matrices the data structure called support matrix is calculated which contains the coordinates of a clique of peaks that match the best between two peak lists based on the distances, i.e. support pairs. Distances are compared row by row and those that match best within tolerances are selected. The support pairs in this instance are the following:

$$(1, 11), (2, 12), (3, 13), (4, 14)$$

Detection of the maximal clique, i.e. clique with the largest number of support pairs is a classic graph theoretical problem [112] that is NP-complete in its computational complexity. The registration offsets are calculated for each dimension for every support pair and then averaged:

$$\boldsymbol{registration}[\boldsymbol{l}] = \frac{\mathbf{1}}{\boldsymbol{n}}\sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}}\left|\boldsymbol{peak_i}[\boldsymbol{l}] - \boldsymbol{peak_j}[\boldsymbol{l}]\right|,$$

where $peak_i$ – peaks from peaklist1, $peak_j$ – peaks from peaklist2 such that they form support pair ($peak_i$, $peak_j$), n – number of support pairs, l – comparable dimension (*CA* or *N*). **Table 23** shows an example calculation for the identified support pairs.

Table 23. Example of registration offset calculation for identified support pairs.

CA	***N***
55.319 – 55.119 = 0.2 ppm	122.274 – 121.826 = 0.448 ppm
60.062 – 59.848 = 0.214 ppm	117.66 – 117.36 = 0.3 ppm
57.132 – 56.968 = 0.164 ppm	125.585 – 125.285 = 0.3 ppm
53.815 – 53.615 = 0.2 ppm	124.145 – 123.855 = 0.29 ppm

Finally, average of registration offset values can be calculated per each comparable dimension:

$$registration[CA] = \frac{0.2+0.214+0.164+0.2}{4} = 0.194 \text{ ppm}$$

$$registration[N] = \frac{0.448+0.3+0.3+0.29}{4} = 0.334 \text{ ppm}$$

6.3.3 Pairwise grouping algorithm

Similarly to the pairwise registration algorithm, the pairwise grouping algorithm was developed within a single code base and has all the properties of the single peak list grouping algorithm. The difference is that when two different peak lists are compared, the pairwise grouping algorithm has additional rules to detect and resolve spin system split, spin system overlap, or missing peaks or spin systems. **Figure 39** shows the flow diagram that describes the process for grouping peaks into spin systems. Instead of working on a single experimental peak list, the pairwise grouping algorithm works on a combined "input" and "root" peak list. For every peak or internal spin system that "input" and/or peak list composed of it first initializes the spin system cluster and queries all neighbor peaks or internals spin system clusters using match tolerances derived from the pairwise registration algorithm. Then similarly to single peak list grouping, it passes all identified neighbors and seed to the expansion phase in order to find additional peaks or internal spin system clusters that may belong to the initialized spin system cluster. The criteria for inclusion of a peak or peak group into a cluster are the same as for single peak list grouping algorithm as described by equation (4). Then, distinctly from the single peak list grouping algorithm, the pairwise grouping algorithm passes all identified peaks and internal clusters to examine cluster step in order to identify potential problems in the final cluster, such as spin system split, spin system overlap. Once the problem is detected, the algorithm tries to resolve it and return clean spin system clusters. There are few pairwise comparison outcomes that are possible: one-to-one, one-to-many, many-to-one, and many-to-many group mappings.

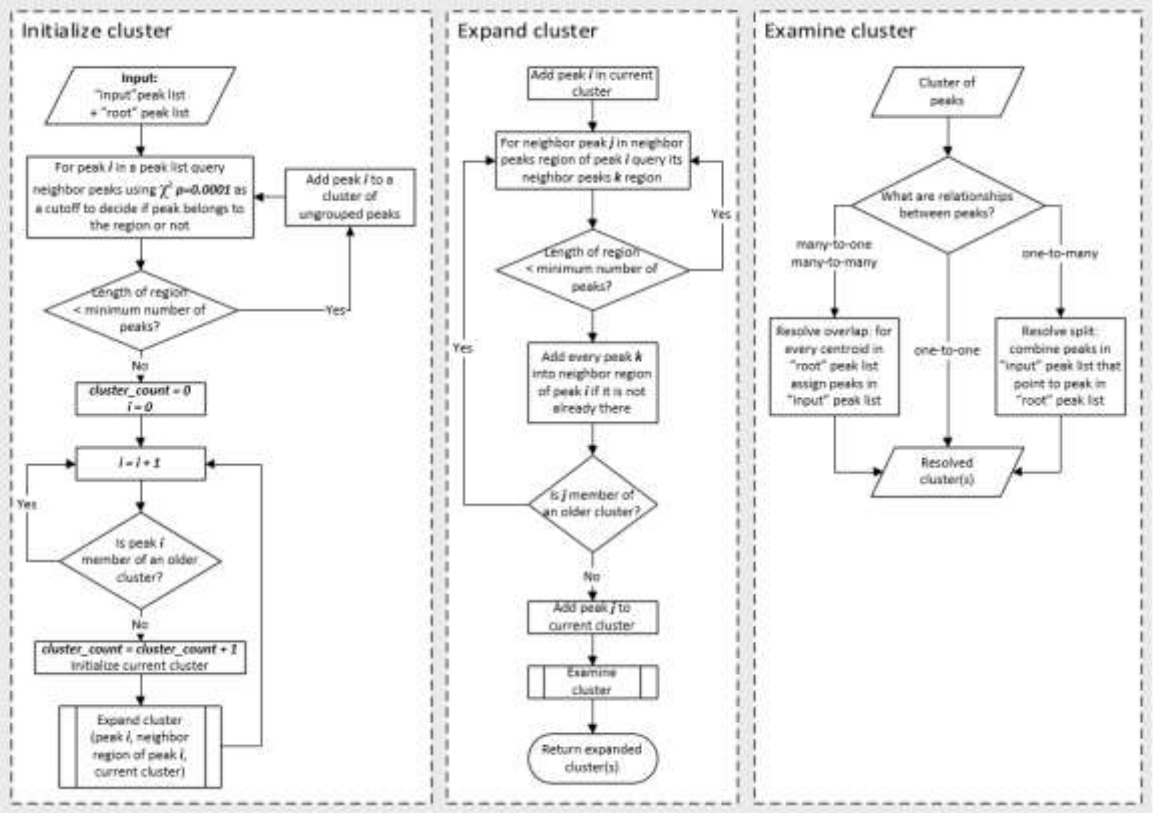

Initialize cluster
Input:
"input" peak list
+ "root" peak list
For peak i in a peak list query neighbor peaks using χ^2 p=0.0001 as a cutoff to decide if peak belongs to the region or not
Add peak i to a cluster of ungrouped peaks
Length of region < minimum number of peaks?
Yes
No
cluster_count = 0
i = 0
i = i + 1
Yes
Is peak i member of an older cluster?
No
cluster_count = cluster_count + 1
Initialize current cluster
Expand cluster (peak i, neighbor region of peak i, current cluster)
Expand cluster
Add peak i in current cluster
For neighbor peak j in neighbor peaks region of peak i query its neighbor peaks k region
Yes
Length of region < minimum number of peaks?
Yes
Add every peak k into neighbor region of peak i if it is not already there
Is j member of an older cluster?
No
Add peak j to current cluster
Examine cluster
Return expanded cluster(s)
Examine cluster
Cluster of peaks
What are relationships between peaks?
many-to-one
many-to-many
one-to-many
one-to-one
Resolve overlap: for every centroid in "root" peak list assign peaks in "input" peak list
Resolve split: combine peaks in "input" peak list that point to peak in "root" peak list
Resolved cluster(s)

Figure 39. Flow diagram of the pairwise grouping algorithm.

6.3.3.1. One-to-one pairwise comparison

The one-to-one pairwise comparison case is the best possible outcome. This means that during every pairwise comparison that occurred during merging peak or internal spin systems into global spin systems there was always a one-to-one matching between peaks or groups of peaks, i.e. the resulting global spin system is clean and does not contain any overlap. **Figure 40** demonstrates the merging of peaks and internal spin systems into global spin systems for the one-to-one pairwise comparison case.

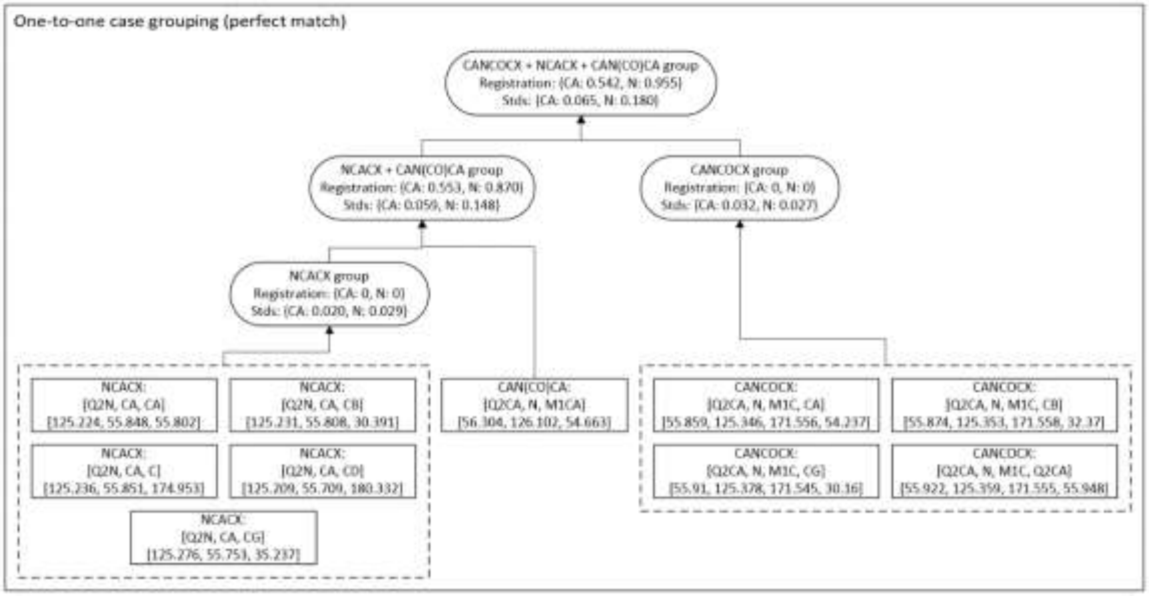


Figure 40. One-to-one pairwise comparison case.

6.3.3.2 One-to-many pairwise comparison

The one-to-many comparison case usually means that during merging of peaks into global spin system clusters, there was a situation when several internal spin systems or peaks pointed to a single spin system in a global comparison. This usually indicates the split between internal spin systems or internal spin systems and individual groups. The split case is resolved by merging several split entities in order to produce a clean global spin system. **Figure 41** demonstrates the split case when two internal spin systems in the NCACX peak list were merged into a single spin system during the pairwise comparison of internal NCACX spin systems with CAN(CO)CA peak list.

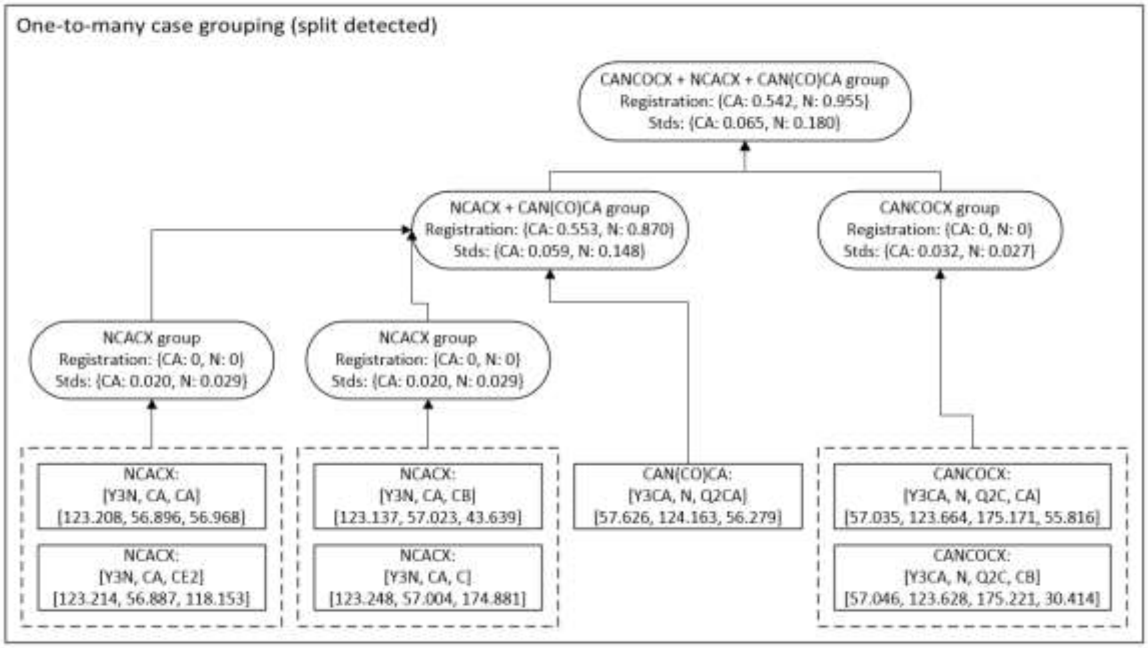


Figure 41. One-to-many pairwise comparison case.

6.3.3.3 Many-to-one pairwise comparison

The many-to-one pairwise comparison case during creation of global spin systems is the undesirable situation, which usually indicates there was an overlap in one of the internal spin system groups that needs to be resolved before creating the final global spin systems. **Figure 42** shows the case with the overlap that happened in the internal grouping of the CANCOCX peak list. Two internal spin system groups that were created from NCACX and CAN(CO)CA pairwise comparisons point to a single CAN(CO)CX internal spin system group. **Figure 43** shows the overlap is being resolved, i.e. right before creating the final overlapped global spin system, the overlapped CANCOCX internal spin system group is disassembled into individual peaks. Next, the two internal spin systems that were created from the NCACX and CAN(CO)CA comparison act as centroids for assigning each individual peak from the CANCOCX peak list. The variance normalized Euclidean distance is used in order to identify the closest centroid and assign each individual peak. The variances for normalizing Euclidean distance are derived from the pairwise registration algorithm. In the end, instead of one overlapped global spin system, two resolved global spin systems are created.

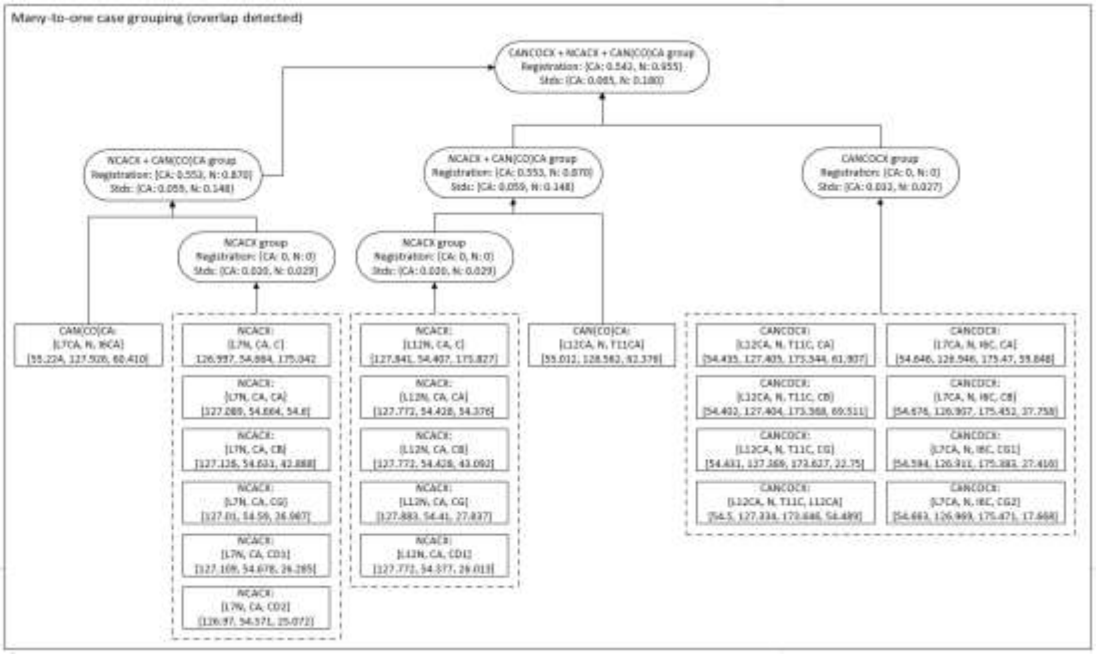


Figure 42. Many-to-one pairwise comparison case (overlapped spin systems).

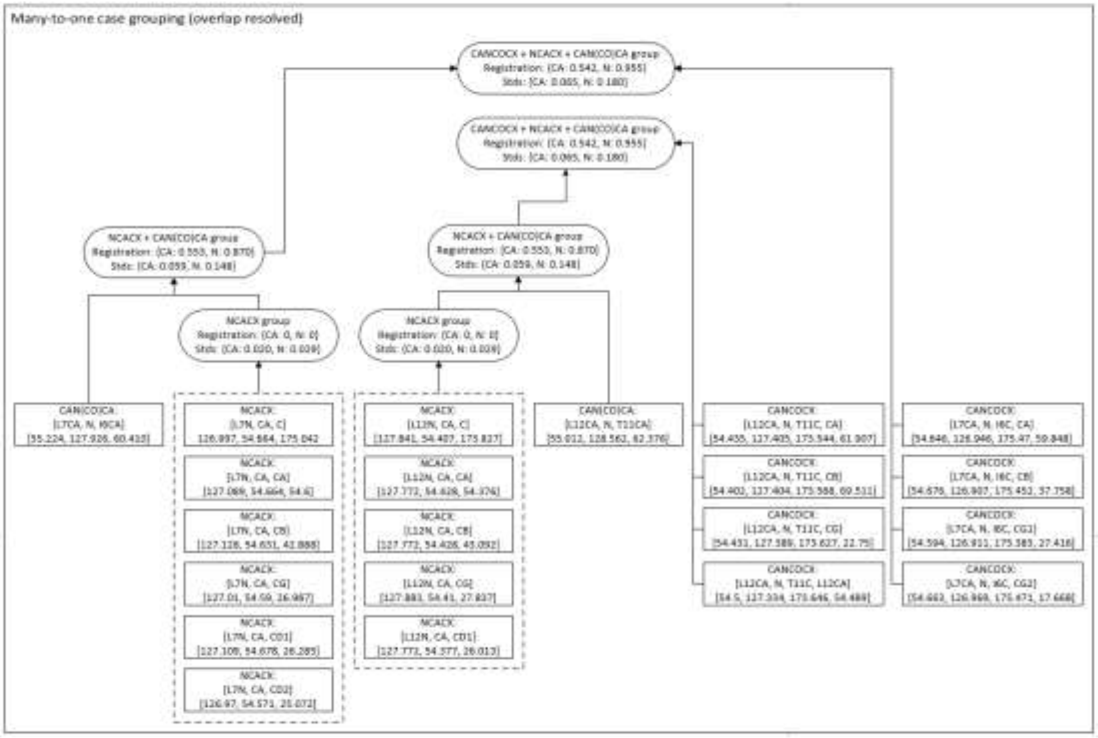


Figure 43. Many-to-one pairwise comparison case (resolved spin systems).

6.3.3.4 Many-to-many pairwise comparison

The many-to-many pairwise comparison case is very similar to the many-to-one pairwise comparison case and also indicates the overlap during the global spin system creation process. **Figure 44** demonstrates a many-to-many pairwise comparison case where two different internal spin systems created from NCACX and CAN(CO)CA pairwise comparison point to two different internal spin systems created from CANCOCX. Once this case is detected, it is handled in a similar way as many-to-one comparison, i.e. CANCOCX internal spin systems are disassembled into individual peaks and pairwise spin system groups created from NCACX and CAN(CO)CA act as centroids. The variance normalized Euclidean distance is calculated in order to find the closest spin system. **Figure 45** shows how the overlapped spin system is being resolved into two different global spin systems.

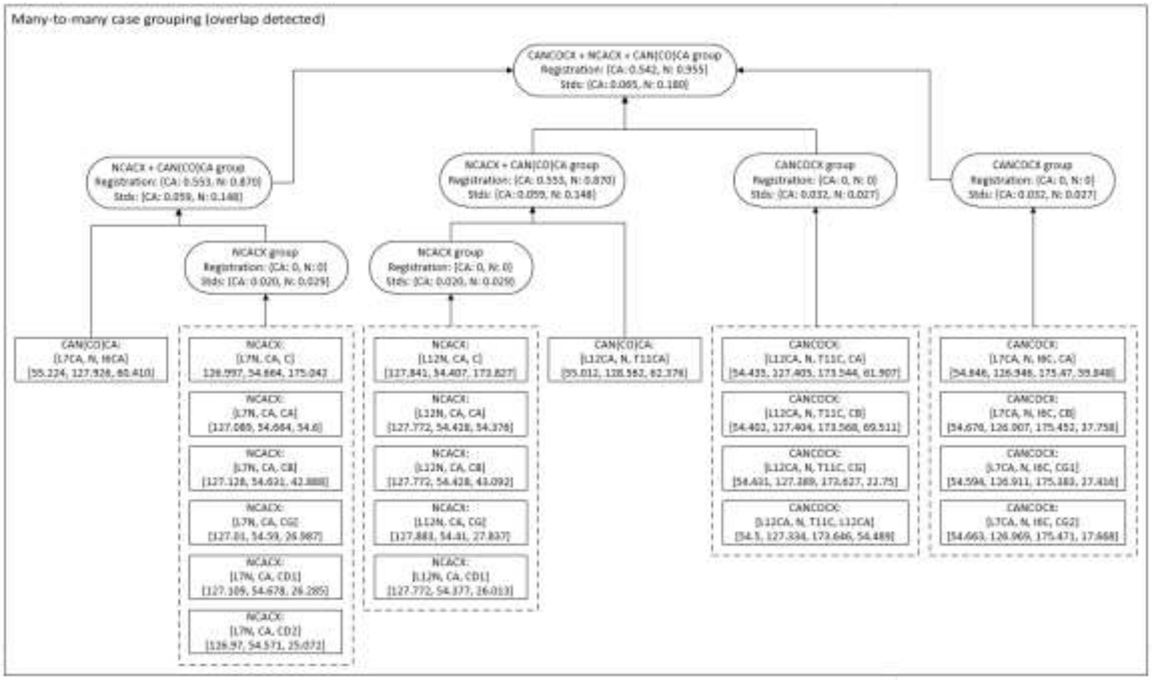


Figure 44. Many-to-one pairwise comparison case (overlapped spin systems).

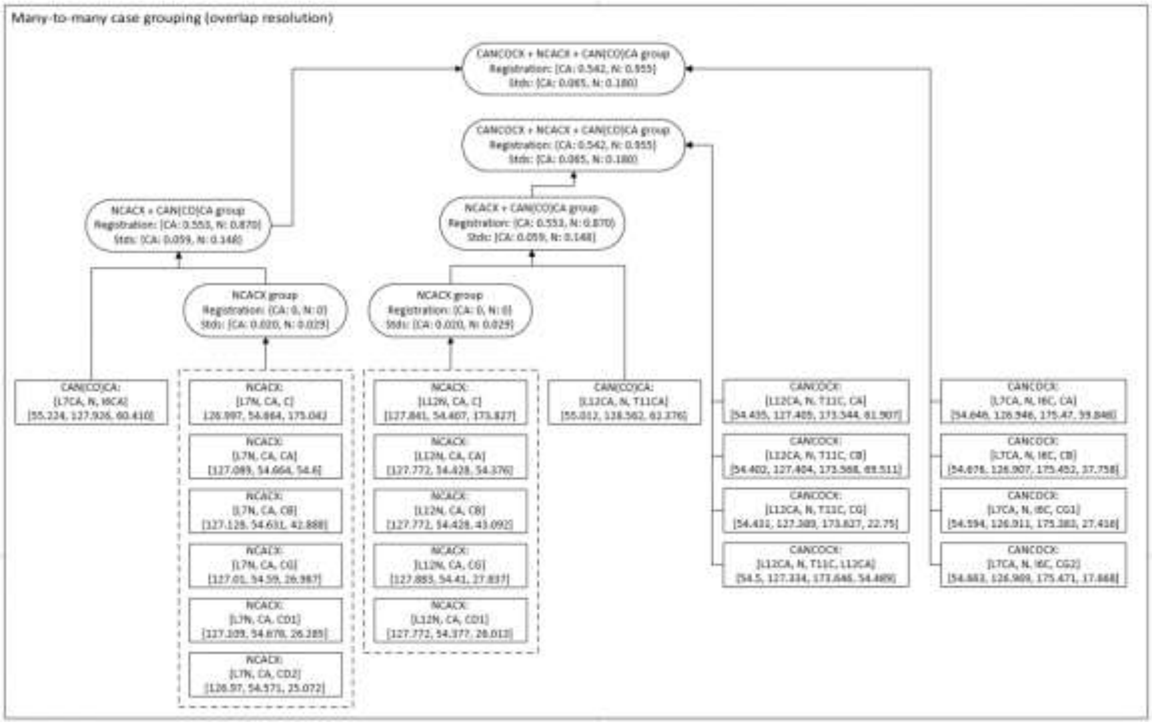


Figure 45. Many-to-one pairwise comparison case (resolved spin systems).

6.3.3.5 Missing spin system recovery

In addition to detecting split and overlap cases during pairwise comparison, it is also possible to recover spin systems or peaks that are present in one peak list but missing in the other. **Figure 46** demonstrates an example when the peak is missing in the CAN(CO)CA peak list and the internal spin system group is not formed due to this fact. The group of peaks is not discarded during the pairwise comparison and is used for pairwise comparison at later stages and if a corresponding peak or group of peaks is found within the CANCOCX peak list, the global clean cluster can be created. In other words, to recover a missing internal spin system, it is necessary that it is found in at least two different peak lists.

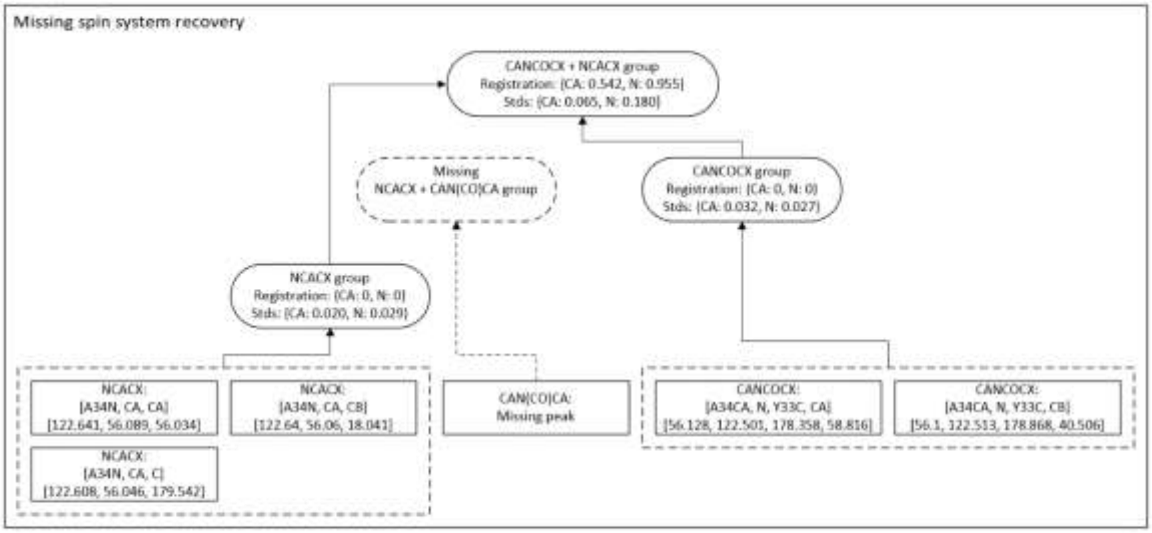


Figure 46. Missing spin system recovery.

6.4 Results and Discussion

6.4.1 Importance of peak list registration

Inaccuracies in NMR referencing cause variance in chemical shift values between different experimental peak lists. As a result, inconsistencies between different peak lists present a serious challenge for the automated protein resonance assignment algorithms. The pairwise registration and grouping algorithm produces both the registration offset values and standard deviations values for each comparable dimension between different peak lists. The standard deviations are then used to calculate match tolerance values for each dimension for grouping peaks into spin systems. Prior to grouping, the offset values must be applied to one of the peak lists in order for peaks to match within match tolerance values. Failure to apply registration offsets typically result in failure to group peaks into spin systems and every peak ends up ungrouped (considered as “noise” data point) or results in incorrect peak grouping with severe overlap.

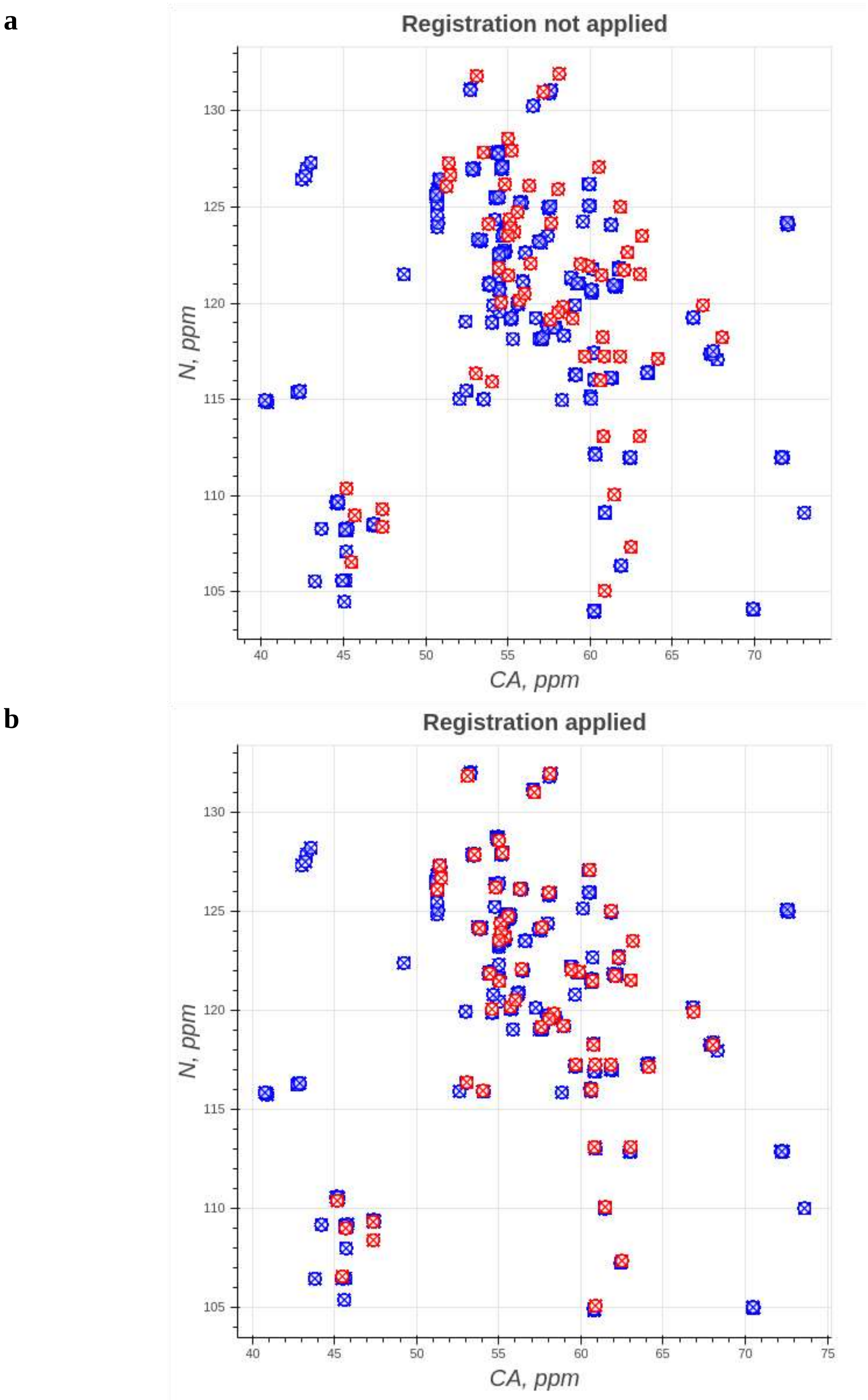


Figure 47. CAN(CO)CA peak list (red crosses) and NCACX peak list (blue crosses) without registration (**a**) and with registration applied (**b**).

Figure 47 shows the ^{15}N and ^{13}CA peak positions of the "root" CAN(CO)CA peak list versus "input" NCACX without (**Figure 47a**) and with (**Figure 47b**) applied dimension-specific registration offset values. It is clear that in order to group peaks into spin systems without registration applied (**Figure 47a**), the match tolerances values would have to be increased which in turn will result in spin system overlap. On the other hand, calculated registration offset values applied to the "input" NCACX peak list (**Figure 47a**) make "input" peaks match "root" peaks and results in correct grouping with no overlap.

6.4.2 Correction of manually assigned peak lists

With the help of the pairwise grouping algorithm, I was able to correct the manual expert assignment of the peak lists and verify it using the deposited chemical shift values in the BMRB (BMRBID 18397).

Table 24 shows an example of an experimental manually assigned CAN(CO)CA peak list. Using the pairwise grouping algorithm, incorrect assignments were identified as well as additional assignments were made directly within the same peak list using unassigned peaks ("?-?-?" assignment designates manually unassigned peak lists but were present in the spectrum and have been retained). Red rows in

Table 24 indicate incorrect assignment within the CAN(CO)CA peak list, for example, peaks #27 and #28 that belong to neighboring spin systems T18-T17 and T17-T16 were assigned incorrectly. In addition, peak #63 that belongs to the D40-V39 spin system was incorrectly assigned to the E56-T55 spin system. Moreover, the grouping algorithm was able to identify the correct E56-T55 spin system (peak #7) and missing Y45-T44 spin system (peak #11).

Table 24. Manually assigned CAN(CO)CA peak list example.

Peak #	Assignment	w1	w2	w3
1	?-?-?	54.770	126.179	54.918
2	?-?-?	61.880	117.174	73.084
3	?-?-?	60.822	117.328	70.994
4	?-?-?	47.369	108.372	53.976
5	?-?-?	58.146	131.956	72.584
6	?-?-?	58.020	131.913	58.106
7	?-?-?	58.118	131.926	61.703
8	?-?-?	63.044	121.502	54.907
9	?-?-?	63.157	123.491	52.942
10	?-?-?	55.168	123.925	55.013
11	?-?-?	58.358	119.808	61.319
12	Q2CA-N-M1CA	56.304	126.102	54.663
13	Y3CA-N-Q2CA	57.626	124.163	56.279
14	K4CA-N-Y3CA	55.387	123.710	57.563
15	L5CA-N-K4CA	53.518	127.842	55.228
16	I6CA-N-L5CA	60.533	127.075	53.453
17	L7CA-N-I6CA	55.224	127.926	60.410
18	N8CA-N-L7CA	51.263	126.082	54.975
19	G9CA-N-N8CA	45.176	110.361	51.203
20	K10CA-N-G9CA	59.885	121.926	45.111
21	T11CA-N-K10CA	62.498	107.326	59.759
22	L12CA-N-T11CA	55.012	128.562	62.376
23	K13CA-N-L12CA	53.824	124.124	54.888
24	G14CA-N-K13CA	45.484	106.539	53.810
25	E15CA-N-G14CA	54.450	121.845	45.419
26	T16CA-N-E15CA	60.643	115.975	54.358
27	T17CA-N-T16CA	61.827	117.237	60.837
28	T18CA-N-T17CA	60.864	117.237	60.548
29	E19CA-N-T18CA	54.823	126.186	61.774
30	A20CA-N-E19CA	51.505	126.664	54.788
31	V21CA-N-A20CA	64.132	117.118	51.220
32	D22CA-N-V21CA	53.055	116.343	64.049
33	A23CA-N-D22CA	55.021	123.515	52.959
34	A24CA-N-A23CA	55.035	121.450	54.973
35	T25CA-N-A24CA	68.033	118.227	54.969
36	A26CA-N-T25CA	55.582	124.728	67.876
37	E27CA-N-A26CA	59.686	117.233	55.541
38	K28CA-N-E27CA	60.776	118.234	59.600
39	V29CA-N-K28CA	66.870	119.901	60.656
40	F30CA-N-V29CA	58.074	119.534	66.658
41	K31CA-N-F30CA	60.706	121.458	57.950
42	Q32CA-N-K31CA	59.427	122.031	60.501
43	Y33CA-N-Q32CA	62.080	121.725	59.295
44	N35CA-N-A34CA	57.588	119.148	56.507
45	D36CA-N-N35CA	56.395	122.065	57.537
46	N37CA-N-D36CA	54.041	115.918	56.419
47	G38CA-N-N37CA	47.362	109.297	53.993
48	V39CA-N-G38CA	62.284	122.646	47.257
49	G41CA-N-D40CA	45.692	108.969	53.266
50	E42CA-N-G41CA	55.712	120.157	45.616
51	W43CA-N-E42CA	58.048	125.926	55.649
52	T44CA-N-W43CA	61.481	110.041	57.960
53	D46CA-N-Y45CA	51.405	127.294	58.240
54	D47CA-N-D46CA	55.107	124.372	51.411
55	A48CA-N-D47CA	54.587	120.039	54.916
56	T49CA-N-A48CA	60.880	105.053	54.496
57	K50CA-N-T49CA	56.014	120.492	60.772
58	T51CA-N-K50CA	63.023	113.082	56.019
59	F52CA-N-T51CA	57.166	130.990	62.949
60	T53CA-N-F52CA	60.815	113.073	57.045
61	V54CA-N-T53CA	58.945	119.200	60.808

62	T55CA-N-V54CA	61.846	125.006	58.929
63	E56CA-N-T55CA	53.102	131.816	62.197

Table 25 shows the CAN(CO)CA peak list that has been corrected after the pairwise algorithm comparison.

Table 25. Corrected manually assigned CAN(CO) CA peak list example.

Peak #	**Assignment**	**w1**	**w2**	**w3**
1	?-?-?	54.770	126.179	54.918
2	?-?-?	61.880	117.174	73.084
3	?-?-?	60.822	117.328	70.994
4	?-?-?	47.369	108.372	53.976
5	?-?-?	58.146	131.956	72.584
6	?-?-?	58.020	131.913	58.106
7	E56CA-N-T54CA	58.118	131.926	61.703
8	?-?-?	63.044	121.502	54.907
9	?-?-?	63.157	123.491	52.942
10	?-?-?	55.168	123.925	55.013
11	Y45CA-N-T44CA	58.358	119.808	61.319
12	Q2CA-N-M1CA	56.304	126.102	54.663
13	Y3CA-N-Q2CA	57.626	124.163	56.279
14	K4CA-N-Y3CA	55.387	123.710	57.563
15	L5CA-N-K4CA	53.518	127.842	55.228
16	I6CA-N-L5CA	60.533	127.075	53.453
17	L7CA-N-I6CA	55.224	127.926	60.410
18	N8CA-N-L7CA	51.263	126.082	54.975
19	G9CA-N-N8CA	45.176	110.361	51.203
20	K10CA-N-G9CA	59.885	121.926	45.111
21	T11CA-N-K10CA	62.498	107.326	59.759
22	L12CA-N-T11CA	55.012	128.562	62.376
23	K13CA-N-L12CA	53.824	124.124	54.888
24	G14CA-N-K13CA	45.484	106.539	53.810
25	E15CA-N-G14CA	54.450	121.845	45.419
26	T16CA-N-E15CA	60.643	115.975	54.358
27	T18CA-N-T17CA	61.827	117.237	60.837
28	T17CA-N-T16CA	60.864	117.237	60.548
29	E19CA-N-T18CA	54.823	126.186	61.774
30	A20CA-N-E19CA	51.505	126.664	54.788
31	V21CA-N-A20CA	64.132	117.118	51.220
32	D22CA-N-V21CA	53.055	116.343	64.049
33	A23CA-N-D22CA	55.021	123.515	52.959
34	A24CA-N-A23CA	55.035	121.450	54.973
35	T25CA-N-A24CA	68.033	118.227	54.969
36	A26CA-N-T25CA	55.582	124.728	67.876
37	E27CA-N-A26CA	59.686	117.233	55.541
38	K28CA-N-E27CA	60.776	118.234	59.600
39	V29CA-N-K28CA	66.870	119.901	60.656
40	F30CA-N-V29CA	58.074	119.534	66.658
41	K31CA-N-F30CA	60.706	121.458	57.950
42	Q32CA-N-K31CA	59.427	122.031	60.501
43	Y33CA-N-Q32CA	62.080	121.725	59.295
44	N35CA-N-A34CA	57.588	119.148	56.507
45	D36CA-N-N35CA	56.395	122.065	57.537
46	N37CA-N-D36CA	54.041	115.918	56.419
47	G38CA-N-N37CA	47.362	109.297	53.993
48	V39CA-N-G38CA	62.284	122.646	47.257
49	G41CA-N-D40CA	45.692	108.969	53.266
50	E42CA-N-G41CA	55.712	120.157	45.616
51	W43CA-N-E42CA	58.048	125.926	55.649
52	T44CA-N-W43CA	61.481	110.041	57.960
53	D46CA-N-Y45CA	51.405	127.294	58.240
54	D47CA-N-D46CA	55.107	124.372	51.411

55	A48CA-N-D47CA	54.587	120.039	54.916
56	T49CA-N-A48CA	60.880	105.053	54.496
57	K50CA-N-T49CA	56.014	120.492	60.772
58	T51CA-N-K50CA	63.023	113.082	56.019
59	F52CA-N-T51CA	57.166	130.990	62.949
60	T53CA-N-F52CA	60.815	113.073	57.045
61	V54CA-N-T53CA	58.945	119.200	60.808
62	T55CA-N-V54CA	61.846	125.006	58.929
63	D40CA-N-V39CA	53.102	131.816	62.197

6.4.3 Accuracy of pairwise registration algorithm on simulated peak lists with known offsets

6.4.3.1 Peak lists with small amount of variance

To evaluate how accurately algorithm can calculate offset registration values for each of the comparable dimensions, two peak lists using the peak list simulation algorithm (described in Chapter 5) were simulated. Two peak list from GB1 protein entry (BMRBID 18397): one peak list is used as a “root” peak list (CAN(CO)CA) and the other peak list is used as “input” peak list (NCACX). The amount of variance (in terms of standard deviations) that was added to the peak lists was 0.01 ppm for both ^{13}CA and ^{15}N dimension. Next, offset values were added to each of the comparable dimensions in “input” peak list, i.e. to every peak in simulated NCACX peak list registration offset value 0.55 ppm for every ^{13}CA dimension and 0.87 ppm for every ^{15}N dimension were added. The offset values were chosen arbitrarily. The registration algorithm calculated the registration offset values for ^{13}CA and ^{15}N dimensions that match initially specified offset values (see **Table 26**). This means that in order for the “input” NCACX peak list to match the “root” CAN(CO)CA peak list, we need to subtract 0.55 ppm for every peak in the ^{13}CA dimension and 0.87 ppm for every peak in the ^{15}N dimension. Here, the introduced registration values are positive, because they were added to the simulated “input” peak list; calculated registration values are negative, meaning that we need to subtract those offset values in order to make “input” peak list match “root” peak list.

Also, the introduced standard deviation is the parameter that was used to set the amount of noise drawn from random normal distribution in order to create a simulated peak list; therefore, the calculated standard deviation is typically smaller than the introduced standard deviation parameter.

Table 26. The offset values calculated by registration algorithm during pairwise comparison of CAN(CO)CA and NCACX simulated peak lists with minimum variance.

Dimension	Introduced registration offset value, ppm	Calculated registration offset value, ppm	Introduced standard deviation, ppm	Calculated standard deviation, ppm
^{15}N	0.55	–0.5521	0.01	0.009
^{13}CA	0.87	–0.8693	0.01	0.008

The exact simulated CAN(CO)CA and NCACX peak lists that were used to calculate the registration offsets are shown in **Table B1** and **Table B2**, respectively.

6.4.3.1 Peak lists with larger variance

The original experimental peak lists contained the larger variance in their ^{13}CA and ^{15}N dimensions, 0.02 ppm and 0.03 ppm, respectively, in terms of standard deviations. Simulated peak lists with the amount of variance corresponding to 0.02 ppm for ^{13}CA and 0.03 ppm for the ^{15}N dimensions in terms of standard deviation values were created. Next offset values were added to both ^{13}CA (0.55 ppm) and ^{15}N (0.87 ppm) as in the previous simulation. **Table 27** shows that the registration algorithm was able to identify correct offset values with slightly higher amount of variance introduced into the peak lists.

Next, a larger amount of variance was introduced to the peak lists. The peak lists were simulated using a variance equal to 0.1 ppm and 0.15 ppm in terms of standard deviation values for the ^{13}CA and ^{15}N dimensions. The offset values 0.55 ppm and 0.87

ppm were added to ^{13}CA and ^{15}N dimensions of the NCACX peak list as in the previous simulations, and the registration algorithm was tested again.

Table 27. The offset values calculated by registration algorithm during pairwise comparison of CAN(CO)CA and NCACX simulated peak lists with amount of variance corresponding to experimental peak lists.

Dimension	Introduced registration offset value, ppm	Calculated registration offset value, ppm	Introduced standard deviation, ppm	Calculated standard deviation, ppm
^{15}N	0.55	–0.5518	0.03	0.02
^{13}CA	0.87	–0.8692	0.02	0.01

Table 28 shows that the registration algorithm was able to handle the amount of variance five times larger than that in the original experimental peak lists and was able to report correct offset registration values.

Table 28. The offset values calculated by registration algorithm during pairwise comparison of CAN(CO)CA and NCACX simulated peak lists with larger amount of variance.

Dimension	Introduced registration offset value, ppm	Calculated registration offset value, ppm	Introduced standard deviation, ppm	Calculated standard deviation, ppm
^{15}N	0.55	–0.5518	0.15	0.13
^{13}CA	0.87	–0.8748	0.1	0.08

6.4.4 Accuracy of the pairwise spin system grouping algorithm

6.4.4.1 Pairwise spin system grouping on experimental peak lists

Table 29 shows the summary of the pairwise grouping algorithm on experimental NCACX, CAN(CO)CA, and CANCOCX. First, two internally grouped NCACX and CAN(CO)CA peak lists were grouped pairwise. Next, CANCOCX groups were grouped with the groups from NCACX and CAN(CO)CA. The spin system groups were analyzed in terms of a number of overlaps and splits at each pairwise grouping step. The results of the pairwise grouping algorithm show that the majority of the spin system clusters are grouped with no overlap.

Table 29. Accuracy of the pairwise grouping algorithm on experimental peak lists.

Protein/ Group	Expected spin systems	Observed spin systems	Registration offsets	Calculated stds	Overlapped spin systems	Split spin systems
GB1 [NCACX + CAN(CO)CA]	55	55	CA: 0.553 N: 0.870	CA: 0.059 N: 0.148	2	1
GB1 [[NCACX + CAN(CO)CA]+CANCOCX]	55	60	CA: 0.542 N: 0.955	CA: 0.065 N: 0.180	4	2

6.4.4.2 Pairwise spin system grouping on simulated peak lists

Table 30 shows the same summary of the grouping algorithm but using simulated peak lists. Due to the smaller variance within simulated peak lists the number of overlap and split spin systems are minimized to 0 at each pairwise grouping step.

Table 30. Accuracy of the pairwise grouping algorithm on simulated peak lists.

Protein/ Group	Expected spin systems	Observed spin systems	Registration offsets	Calculated stds	Overlapped spin systems	Split spin systems
GB1 [NCACX + CAN(CO)CA]	55	55	CA: -0.002 N: 0.001	CA: 0.009 N: 0.010	0	0
GB1 [[NCACX + CAN(CO)CA]+CANCOCX]	55	55	CA: -0.003 N: -0.001	CA: 0.008 N: 0.009	0	0

6.5 Conclusions

A new pairwise peak list registration and grouping algorithms were developed. The pairwise registration and grouping algorithms rely on single peak list registration algorithms in order to create internal spin system groups and expand those groups by a pairwise comparison of different peak lists starting from the best quality peak lists first. The algorithm can take into account multiple sources of variance present within the single peak list as well as between different peak lists due to the iterative nature of the algorithm and the coupling registration and grouping steps. The algorithms can detect spin systems split, overlap, or recover missing spin systems in one or more peak lists. Also it was demonstrated on simulated peak lists that the registration algorithm can accurately determine registration offset values as well as standard deviation values.

CHAPTER 7
DISCUSSION

7.1 Evaluation of performance

The proposed algorithms are implemented as individual programs with their own command-line interfaces and documentation. The nmrstarlib package is written in the Python programming language with C-extensions implemented using the Cython programming language to improve speed efficiency for processing NMR-STAR files. The computational time and space complexity of the nmrstarlib library linearly depends on the size of the file. Typically, it takes a fraction of a second to process a single NMR-STAR file (see **Figure 23**).

The single and pairwise peak list registration algorithm is implemented in C++ programming language and is the most computationally intensive algorithm in the discussed research. The algorithm is optimized to a computational complexity of $\mathrm{O}(mn^2 \cdot \log n)$ where m and n represent the lengths of the "root" and "input" peak lists, respectively.

The spin system grouping algorithm is implemented in Python and is coupled with the registration algorithm in order to discover multiple sources of variance present in a single peak list as well as between different peak lists. The average computational complexity of the grouping algorithm alone without coupling with the registration algorithm is $\mathrm{O}(n \cdot \log n)$, where n represents a total number of peaks being grouped by the grouping algorithm. It depends on the region query function that queries peaks according to defined distance function, the worst case running time is $\mathrm{O}(n^2)$.

The jpredapi package is designed to submit queries to the Jpred4 secondary structure prediction server [29] and is implemented in Python. The running time depends on the load of the third-party Jpred4 secondary structure prediction server.

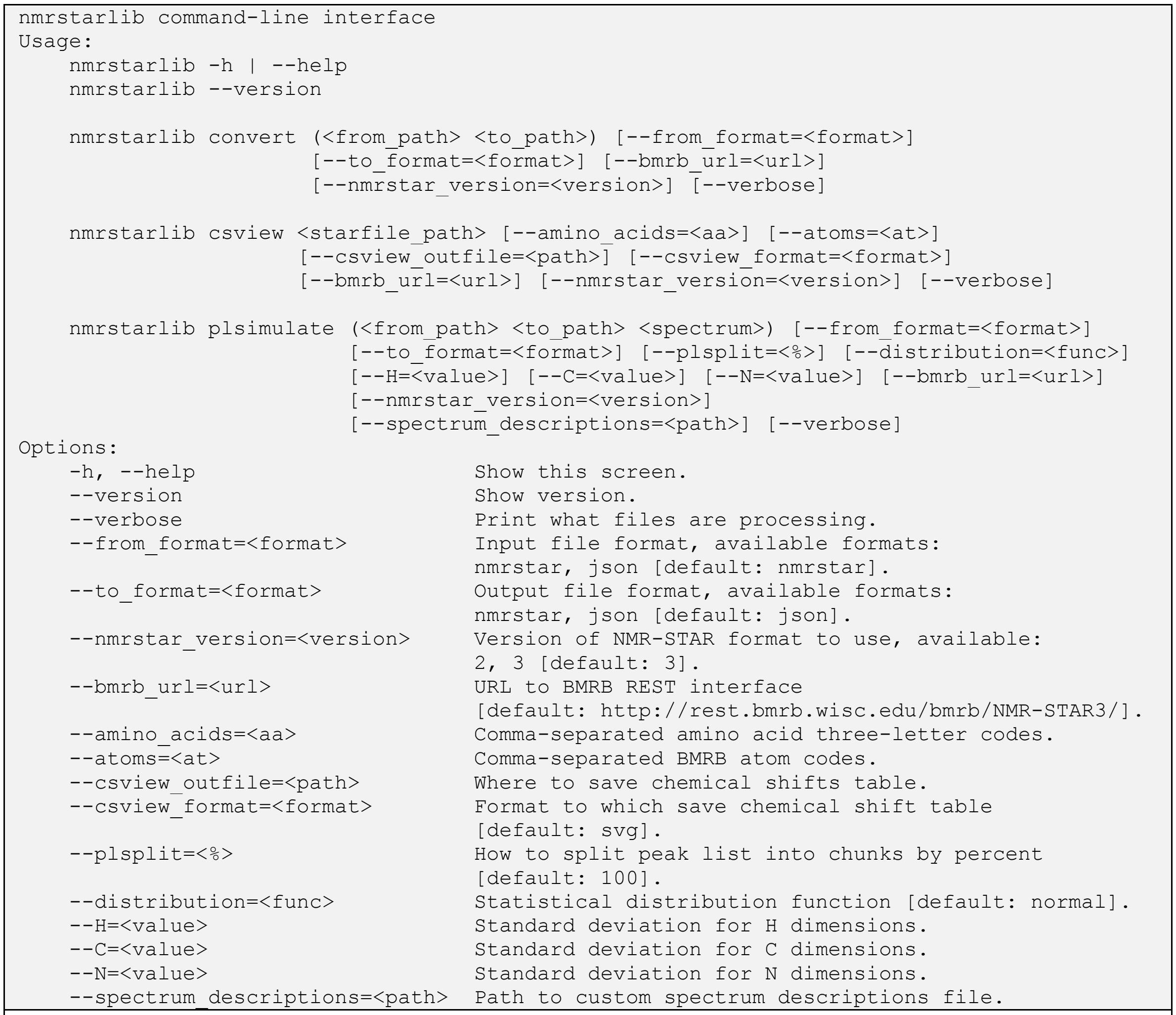
7.2 Command-line interfaces

7.2.1 The nmrstarlib command-line interface

The main use cases of the nmrstarlib command-line interface is to convert original NMR-STAR files to their JSONized representation (using `convert` command) and generate simulated peak lists (using `plsimulate` command) utilizing assigned chemical shift values and the spectrum description describing the magnetization pathway transfer which in turn describes the specific dimensions that will be added to each peak within a peak list. The varying amount of variance can be added to the simulated peak lists using options to specify the standard deviation for each of the ^{1}H, ^{13}C, and ^{15}N dimensions. **Figure 48** shows the complete command-line interface.

7.2.2 Registration algorithm command-line interface

Figure 49 shows the command-line interface for the single and pairwise peak list registration algorithm. The execution requires providing two peak lists “root” and “input” to calculate per dimension offset registration values and standard deviations. It is also necessary to specify the correct order of dimensions using option (`--dim` parameter) and control the registration mode (`--noi` parameter must be specified in order to execute the algorithm in single peak list registration mode).



```
nmrstarlib command-line interface
Usage:
    nmrstarlib -h | --help
    nmrstarlib --version

    nmrstarlib convert (<from_path> <to_path>) [--from_format=<format>]
                       [--to_format=<format>] [--bmrb_url=<url>]
                       [--nmrstar_version=<version>] [--verbose]

    nmrstarlib csview <starfile_path> [--amino_acids=<aa>] [--atoms=<at>]
                      [--csview_outfile=<path>] [--csview_format=<format>]
                      [--bmrb_url=<url>] [--nmrstar_version=<version>] [--verbose]

    nmrstarlib plsimulate (<from_path> <to_path> <spectrum>) [--from_format=<format>]
                          [--to_format=<format>] [--plsplit=<%>] [--distribution=<func>]
                          [--H=<value>] [--C=<value>] [--N=<value>] [--bmrb_url=<url>]
                          [--nmrstar_version=<version>]
                          [--spectrum_descriptions=<path>] [--verbose]
Options:
    -h, --help                        Show this screen.
    --version                         Show version.
    --verbose                         Print what files are processing.
    --from_format=<format>            Input file format, available formats:
                                      nmrstar, json [default: nmrstar].
    --to_format=<format>              Output file format, available formats:
                                      nmrstar, json [default: json].
    --nmrstar_version=<version>       Version of NMR-STAR format to use, available:
                                      2, 3 [default: 3].
    --bmrb_url=<url>                  URL to BMRB REST interface
                                      [default: http://rest.bmrb.wisc.edu/bmrb/NMR-STAR3/].
    --amino_acids=<aa>                Comma-separated amino acid three-letter codes.
    --atoms=<at>                      Comma-separated BMRB atom codes.
    --csview_outfile=<path>           Where to save chemical shifts table.
    --csview_format=<format>          Format to which save chemical shift table
                                      [default: svg].
    --plsplit=<%>                     How to split peak list into chunks by percent
                                      [default: 100].
    --distribution=<func>             Statistical distribution function [default: normal].
    --H=<value>                       Standard deviation for H dimensions.
    --C=<value>                       Standard deviation for C dimensions.
    --N=<value>                       Standard deviation for N dimensions.
    --spectrum_descriptions=<path>  Path to custom spectrum descriptions file.
```

Figure 48. Command-line interface of the nmrstarlib package.

7.2.3 Grouping algorithm command-line interface

Figure 50 shows the command-line interface for the single peak list grouping algorithm. The algorithm requires a single peak list that contain multiple peaks per spin system and path to the registration algorithm executable. The `group` command is used to analyze peaks and returns groups of peaks (spin systems). The `visualize` command is used to visualize the spin systems in 2D space.

```
crs (calculate registration statistics) command-line interface
Calculates the registration statistics that will make input_peaklist match root_peaklist
(reference peak list).

Usage:
    crs (<input_peaklist> <root_peaklist>) [options]

    input_peaklist    The peak list you wish to register and filter.
    root_peaklist     The reference peak list.

Options:
    --verbose                            Print more information.
    --noi                                Run in self-registration mode.
    --nobounds                           Do not perform bounds checking.
    --dim <i1> <i2> <...> : <r1> <r2> <...> Description of matching dimensions in input
                                         and root peak lists.
    --tolerance <num_units>              Number of stds to use as the match tolerance
                                         [default: 4].
    --H <init_std>                       Set starting std to try for H dimensions
                                         [default: 0.0075].
    --C <init_std>                       Set starting std to try for C dimensions
                                         [default: 0.075].
    --N <init_std>                       Set starting std to try for N dimensions
                                         [default: 0.075].
    --i <max>                            Maximum number of iteration to perform
                                         [default: 20].
    --save <json_filename>               Save results of the registration algorithm
                                         into JSON file.
```

Figure 49. Command-line interface of the single and pairwise peak list registration algorithms.

```
ssc (Spin System Creator) command-line interface

Usage:
    ssc -h | --help
    ssc --version
    ssc group (--plpath=<path>) (--plformat=<format>) (--stype=<type>)
              (--dims=<labels>) (--rdims=<labels>)
              [--result=<path>] [--crs=<path>]
    ssc visualize <grouping_result> <x_idx> <y_idx> <x_label> <y_label> <plot_title>

Options:
    -h, --help                 Show this screen.
    --version                  Show version.
    --plpath=<path>            Path to peak list.
    --plformat=<format>        Peak list format.
    --stype=<type>             Spectrum type.
    --dims=<labels>            Comma-separated dimension labels.
    --rdims=<labels>           Comma-separated root dimension labels.
    --crs=<path>               Registration algorithm executable path.
    --result=<path>            Path to directory where results will be saved.
```

Figure 50. Command-line interface of single peak list grouping algorithm (the combined registration and grouping algorithm).

7.2.4 The jpredapi command-line interface

Figure 51 shows the command-line interface for the jpredapi package that is used to submit queries to the secondary structure prediction server. The submit command is used to `submit` queries. The `status` command shows the current status of the submitted job (e.g. processing or completed), and `get_results` command is used to retrieve the results of completed job.

```
jpredapi command-line interface

The RESTful API allows JPred users to submit jobs from the command-line.

Usage:
    jpredapi -h | --help
    jpredapi --version

    jpredapi submit (--mode=<mode> --format=<format>)
                    (--file=<filename> | --seq=<sequence>)
                    [--email=<name@domain.com>] [--name=<job_name>] [--skipPDB=<value>]
                    [--rest=<address>] [--jpred4=<address>] [--silent]

    jpredapi status (--job_id=<id>) [--results_dir=<path>]
                    [--wait_interval=<interval>] [--extract] [--silent]

    jpredapi get_results (--job_id=<id>) [--results_dir=<path>]
                         [--wait_interval=<interval>] [--extract] [--silent]

    jpredapi quota (--email=<name@domain.com>)

Options:
    -h, --help                    Show this help message.
    --version                     Show jpredapi version.
    --silent                      Do not print messages.
    --extract                     Extract results tar.gz archive into folder.
    --mode=<mode>                 Submission mode, possible values: single, batch, msa.
    --format=<format>             Submission format, possible values: raw, fasta, msf,
                                  blc.
    --file=<filename>             Filename of a file with the job input (sequence(s)).
    --seq=<sequence>              Instead of passing input file, for single-sequence
                                  submission.
    --email=<name@domain.com>     E-mail address where job report will be sent
                                  (optional for all but batch submissions).
    --name=<job_name>             Job name.
    --job_id=<job_id>             Job id.
    --skipPDB=<value>             PDB check, possible values: True, False [default: True].
    --results_dir=<path>          Path where to save archive with results.
    --rest=<address>              REST address of server
                                  [default: www.compbio.dundee.ac.uk/jpred4/cgi-bin/rest].
    --jpred4=<address>            Address of Jpred4 server
                                  [default: www.compbio.dundee.ac.uk/jpred4].
    --wait_interval=<interval>    Wait interval before retrying to check job status in
                                  seconds [default: 60].
```

Figure 51. Command-line interface for the jpredapi package.

7.3 Future directions

7.3.1 Advanced spin system typing algorithm

Development of a new spin system typing algorithm that utilizes secondary structure prediction prior information, chemical shift statistics derived from the RefDB [64], and use of covariance matrices to predict the list of most probable amino acid types is the first immediate next step. The Bayesian-based amino acid typing algorithm which utilizes secondary structure prediction typing information can be used for each of the specific ladders within each spin system in order to determine the list of most probable amino acid types. Equation (5) specifies the Bayesian probability to predict the most probable amino acid types for specific ladders within spin system:

$$\boldsymbol{P\left(SS_j \middle| CS_i\right) = \frac{\sum_k P(CS_i|SS_{j,k}) \cdot P\left(SS_{j,k}\right)}{\sum_{j,k}\left(P(CS_i|SS_{j,k}) \cdot P\left(SS_{j,k}\right)\right)}} \tag{5}$$

where $SS_{j,k}$ – sequence site j and secondary structure k (helix, sheet, coil), CS_i – chemical shift values for spin system i, $P\left(SS_j \middle| CS_i\right)$ – probability of SS_j given CS_i, $P(CS_i|SS_{j,k})$ – probability of CS_i given $SS_{j,k}$, $P\left(SS_{j,k}\right)$ – prior information probability of $SS_{j,k}$, $\sum_{j,k}\left(P(CS_i|SS_{j,k}) \cdot P\left(SS_{j,k}\right)\right)$ – sum over all possibilities.

7.3.2 Spin system linking and mapping algorithm

The next step logical step is to use typed spin systems in order to sequentially assign them by linking the nearest neighbor spin systems into segments and mapping segments into protein sequence. The linking of spin systems can be calculated as the difference between sequential and intraresidue ladders. Equation (6) describes the linking score that can be used to identify the neighbor spin systems and form spin system segments:

$$\boldsymbol{L = e^{-\sqrt{\sum_j (S_j - I_j)^2}}} \tag{6}$$

where, S_j – chemical shift j from sequential ladder S, I_j – chemical shift j from intraresidue ladder I (see **Figure 3**).

The segment mapping algorithm maps linked spin system segments uniquely to the protein sequence. Every generated segment can be scored using the equation (7):

$$\boldsymbol{M} = \prod_{\boldsymbol{j}=\boldsymbol{1}}^{\boldsymbol{N}} \boldsymbol{P}(\boldsymbol{SS_j} | \boldsymbol{CS_{i,CL}}) \tag{7}$$

where $P(SS_j|CS_{i,CL})$ is the probability of sequence site SS_j given chemical shifts CS_i within combined ladder (CL).

CHAPTER 8
CONCLUSIONS

To summarize the research, several new general software packages and algorithms were designed and implemented in order to aid the automated resonance assignment of peak lists derived from both solution-state and solid-state NMR spectra. The nmrstarlib library was designed to easily access NMR data from the NMR-STAR formatted files, i.e. access assigned chemical shift values, experimental peak lists if they are available, and generate a large number of simulated peak lists without variance and single or multiple sources of variance for algorithms robustness testing. The jpredapi package was designed to easily submit queries to the secondary structure prediction server and utilize this prior information for the amino acid typing algorithm. A pair of single peak list registration and spin system grouping algorithms was designed in order to address the problem of presence of multiple sources of variance within single peak list that have multiple peaks per spin system and create the initial spin systems based on calculated match tolerance values. It was shown that algorithms using only single iteration and uniform match tolerances approach are only able to recover from 50 % to 80 % of spin systems due to the presence of multiple sources of variance. The single peak list registration and grouping algorithm are able to recover additional spin systems by reevaluating match tolerances in multiple iterations. The pairwise registration and grouping algorithms were designed to solve the problem of multiple sources of variance that exist between different peak lists, calculate offset registration values, i.e. account for inconsistencies between different peak lists that occur due to incorrect chemical shift referencing. In addition, through pairwise comparison of different peak lists the pairwise registration and grouping algorithms can identify the spin system overlap, spin system split, or missing spin systems. Using simulated peak lists from different NMR experiments it was shown that the algorithms can correctly identify artificially introduced offset values as well as match tolerance values required for spin system grouping across peak lists. Together, these methods development and implementation provide valuable tools for protein NMR quality assessment and provide a basis for the development of an effective and robust automated protein resonance assignment package amenable to both solution-state and solid-state NMR peak list datasets.

REFERENCES

[1] K. Wüthrich, G. Wider, G. Wagner, and W. Braun, “Sequential resonance assignments as a basis for determination of spatial protein structures by high resolution proton nuclear magnetic resonance,” *J. Mol. Biol.*, vol. 155, no. 3, pp. 311–319, Mar. 1982.

[2] W. P. Aue, E. Bartholdi, and R. R. Ernst, “Two-dimensional spectroscopy. Application to nuclear magnetic resonance,” *J. Chem. Phys.*, vol. 64, no. 5, pp. 2229–2246, Mar. 1976.

[3] A. Kumar, R. R. Ernst, and K. Wüthrich, “A two-dimensional nuclear Overhauser enhancement (2D NOE) experiment for the elucidation of complete proton-proton cross-relaxation networks in biological macromolecules,” *Biochem. Biophys. Res. Commun.*, vol. 95, no. 1, pp. 1–6, Jul. 1980.

[4] D. Marion, P. C. Driscoll, L. E. Kay, P. T. Wingfield, A. Bax, A. M. Gronenborn, and G. M. Clore, “Overcoming the overlap problem in the assignment of proton NMR spectra of larger proteins by use of three-dimensional heteronuclear proton-nitrogen-15 Hartmann-Hahn-multiple quantum coherence and nuclear Overhauser-multiple quantum coherence spectroscopy:,” *Biochemistry*, vol. 28, no. 15, pp. 6150–6156, Jul. 1989.

[5] B. A. Messerle, G. Wider, G. Otting, C. Weber, and K. Wüthrich, “Solvent suppression using a spin lock in 2D and 3D NMR spectroscopy with H2O solutions,” *J. Magn. Reson.*, vol. 85, no. 3, pp. 608–613, Dec. 1989.

[6] M. Ikura, L. E. Kay, and A. Bax, “A novel approach for sequential assignment of proton, carbon-13, and nitrogen-15 spectra of larger proteins: heteronuclear triple-resonance three-dimensional NMR spectroscopy. Application to calmodulin,” *Biochemistry*, vol. 29, no. 19, pp. 4659–4667, May 1990.

[7] L. E. Kay, M. Ikura, R. Tschudin, and A. Bax, “Three-dimensional triple-resonance NMR spectroscopy of isotopically enriched proteins,” *J. Magn. Reson.*, vol. 89, no. 3, pp. 496–514, Oct. 1990.

[8] A. McDermott, T. Polenova, A. Bockmann, K. W. Zilm, E. K. Paulson, R. W. Martin, and G. T. Montelione, “Partial NMR assignments for uniformly (13C, 15N)-enriched BPTI in the solid state.,” *J. Biomol. NMR*, vol. 16, no. 3, pp. 209–219, 2000.

[9] C. M. Rienstra, L. Tucker-Kellogg, C. P. Jaroniec, M. Hohwy, B. Reif, M. T.

McMahon, B. Tidor, T. Lozano-Pérez, and R. G. Griffin, "De novo determination of peptide structure with solid-state magic-angle spinning NMR spectroscopy," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 99, no. 16, pp. 10260–10265, 2002.

[10] F. Castellani, B. Van Rossum, A. Diehl, M. Schubert, K. Rehbein, and H. Oschkinat, "Structure of a protein determined by solid-state magic-angle-spinning NMR spectroscopy," *Nature*, vol. 420, no. 6911, pp. 98–102, 2002.

[11] D. H. Zhou, G. Shah, M. Cormos, C. Mullen, D. Sandoz, and C. M. Rienstra, "Proton-detected solid-state NMR spectroscopy of fully protonated proteins at 40 kHz magic-angle spinning.," *J. Am. Chem. Soc.*, vol. 129, no. 38, pp. 11791–11801, 2007.

[12] V. Kurauskas, E. Crublet, P. Macek, R. Kerfah, D. F. Gauto, J. Boisbouvier, and P. Schanda, "Sensitive proton-detected solid-state NMR spectroscopy of large proteins with selective CH 3 labelling: application to the 50S ribosome subunit," *Chem. Commun.*, vol. 52, no. 61, pp. 9558–9561, 2016.

[13] P. Fricke, V. Chevelkov, M. Zinke, K. Giller, S. Becker, and A. Lange, "Backbone assignment of perdeuterated proteins by solid-state NMR using proton detection and ultrafast magic-angle spinning," *Nat. Protoc.*, vol. 12, no. 4, pp. 764–782, Mar. 2017.

[14] H. M. Berman, "The Protein Data Bank," *Nucleic Acids Res.*, vol. 28, no. 1, pp. 235–242, Jan. 2000.

[15] E. L. Ulrich, H. Akutsu, J. F. Doreleijers, Y. Harano, Y. E. Ioannidis, J. Lin, M. Livny, S. Mading, D. Maziuk, Z. Miller, E. Nakatani, C. F. Schulte, D. E. Tolmie, R. Kent Wenger, H. Yao, and J. L. Markley, "BioMagResBank," *Nucleic Acids Res.*, vol. 36, no. SUPPL. 1, pp. D402–D408, 2008.

[16] A. Krogh, B. Larsson, G. von Heijne, and E. L. Sonnhammer, "Predicting transmembrane protein topology with a hidden Markov model: application to complete genomes.," *J. Mol. Biol.*, vol. 305, no. 3, pp. 567–80, Jan. 2001.

[17] A. McDermott, "Structure and dynamics of membrane proteins by magic angle spinning solid-state NMR.," *Annu. Rev. Biophys.*, vol. 38, pp. 385–403, Jan. 2009.

[18] G. Von Heijne, "The membrane protein universe: what's out there and why bother?," *J. Intern. Med.*, vol. 261, no. 6, pp. 543–57, 2007.

[19] G. W. Abbott, "Molecular mechanisms of cardiac voltage-gated potassium channelopathies.," *Curr. Pharm. Des.*, vol. 12, no. 28, pp. 3631–3644, 2006.

[20] J. J. Gargus, "Ion channel functional candidate genes in multigenic neuropsychiatric disease.," *Biol. Psychiatry*, vol. 60, no. 2, pp. 177–185, 2006.

[21] A. M. Spiegel, "Defects in G protein-coupled signal transduction in human disease.," *Annu. Rev. Physiol.*, vol. 58, pp. 143–170, 1996.

[22] L. S. King, D. Kozono, and P. Agre, "From structure to disease: the evolving tale of aquaporin biology.," *Nat. Rev. Mol. Cell Biol.*, vol. 5, no. 9, pp. 687–698, 2004.

[23] M. Tang, G. Comellas, and C. M. Rienstra, "Advanced Solid-State NMR Approaches for Structure Determination of Membrane Proteins and Amyloid Fibrils," *Acc. Chem. Res.*, vol. 46, no. 9, pp. 2080–2088, 2013.

[24] "Membrane Proteins of Known 3D Structure Database." [Online]. Available: http://blanco.biomol.uci.edu/mpstruc/.

[25] D. T. Murray, N. Das, and T. A. Cross, "Solid State NMR Strategy for Characterizing Native Membrane Protein Structures," *Acc. Chem. Res.*, 2013.

[26] R. N. Rambaran and L. C. Serpell, "Amyloid fibrils: abnormal protein assembly.," *Prion*, vol. 2, no. 3, pp. 112–7, 2008.

[27] R. Tycko, "Solid State NMR Studies of Amyloid Fibril Structure," *Annu Rev Phys Chem*, vol. 62, pp. 279–299, 2011.

[28] S. Spera and A. Bax, "Empirical correlation between protein backbone conformation and C.alpha. and C.beta. 13C nuclear magnetic resonance chemical shifts," *J. Am. Chem. Soc.*, vol. 113, no. 14, pp. 5490–5492, 1991.

[29] A. Drozdetskiy, C. Cole, J. Procter, and G. J. Barton, "JPred4: a protein secondary structure prediction server," *Nucleic Acids Res.*, vol. 43, no. W1, pp. W389–W394, Jul. 2015.

[30] J. Pauli, M. Baldus, B. van Rossum, H. de Groot, and H. Oschkinat, "Backbone and side-chain 13C and 15N signal assignments of the alpha-spectrin SH3 domain by magic angle spinning solid-state NMR at 17.6 Tesla.," *Chembiochem*, vol. 2, no. 4, pp. 272–81, Apr. 2001.

[31] Y. Li, D. a Berthold, H. L. Frericks, R. B. Gennis, and C. M. Rienstra, "Partial (13)C and (15)N chemical-shift assignments of the disulfide-bond-forming enzyme DsbB by 3D magic-angle spinning NMR spectroscopy.," *Chembiochem*, vol. 8, no. 4, pp. 434–42, Mar. 2007.

[32] N. E. . Buchler, E. R. . Zuiderweg, H. Wang, and R. A. Goldstein, "Protein Heteronuclear NMR Assignments Using Mean-Field Simulated Annealing," *J. Magn. Reson.*, vol. 125, no. 1, pp. 34–42, Mar. 1997.

[33] J. A. Lukin, A. P. Gove, S. N. Talukdar, and C. Ho, "Automated probabilistic method for assigning backbone resonances of (13 C , 15 N) -labeled proteins," vol. 9, pp. 151–166, 1997.

[34] M. Leutner, R. M. Gschwind, J. Liermann, C. Schwarz, G. Gemmecker, and H. Kessler, "Automated backbone assignment of labeled proteins using the threshold accepting algorithm.," *J. Biomol. NMR*, vol. 11, no. 1, pp. 31–43, 1998.

[35] M. A. C. Reed, A. M. Hounslow, K. H. Sze, I. G. Barsukov, L. L. P. Hosszu, A. R.

Clarke, C. J. Craven, and J. P. Waltho, “Effects of Domain Dissection on the Folding and Stability of the 43 kDa Protein PGK Probed by NMR,” *J. Mol. Biol.*, vol. 330, no. 5, pp. 1189–1201, Jul. 2003.

[36] T. K. Hitchens, J. A. Lukin, Y. Zhan, S. A. McCallum, and G. S. Rule, “MONTE: An automated Monte Carlo based approach to nuclear magnetic resonance assignment of proteins,” *J. Biomol. NMR*, vol. 25, no. 1, pp. 1–9, 2003.

[37] C. Bartels, P. Güntert, M. Billeter, and K. Wüthrich, “GARANT-a general algorithm for resonance assignment of multidimensional nuclear magnetic resonance spectra,” *J. Comput. Chem.*, vol. 18, no. 1, pp. 139–149, Jan. 1997.

[38] J. Volk, T. Herrmann, and K. Wüthrich, “Automated sequence-specific protein NMR assignment using the memetic algorithm MATCH.,” *J. Biomol. NMR*, vol. 41, no. 3, pp. 127–38, Jul. 2008.

[39] E. Schmidt and P. Güntert, “A New Algorithm for Reliable and General NMR Resonance Assignment.,” *J. Am. Chem. Soc.*, vol. 134, no. 30, pp. 12817–29, 2012.

[40] D. E. Zimmerman, C. A. Kulikowski, Y. Huang, W. Feng, M. Tashiro, S. Shimotakahara, C. Chien, R. Powers, and G. T. Montelione, “Automated analysis of protein NMR assignments using methods from artificial intelligence.,” *J. Mol. Biol.*, vol. 269, no. 4, pp. 592–610, 1997.

[41] H. N. Moseley and G. T. Montelione, “Automated analysis of NMR assignments and structures for proteins.,” *Curr. Opin. Struct. Biol.*, vol. 9, no. 5, pp. 635–642, 1999.

[42] K.-B. Li and B. C. Sanctuary, “Automated Resonance Assignment of Proteins Using Heteronuclear 3D NMR. 1. Backbone Spin Systems Extraction and Creation of Polypeptides,” *J. Chem. Inf. Comput. Sci.*, vol. 37, no. 2, pp. 359–366, Mar. 1997.

[43] W. Gronwald, L. Willard, T. Jellard, R. F. Boyko, D. S. Wishart, F. D. Sönnichsen, and B. D. Sykes, “CAMRA : Chemical shift based computer aided protein NMR assignments,” pp. 395–405, 1998.

[44] P. Güntert, M. Salzmann, D. Braun, and K. Wüthrich, “Sequence-specific NMR assignment of proteins by global fragment mapping with the program MAPPER,” *J. Biomol. NMR*, vol. 18, no. 2, pp. 129–137, 2000.

[45] H. S. Atreya, S. C. Sahu, K. V. R. Chary, and G. Govil, “A tracked approach for automated NMR assignments in proteins (TATAPRO),” pp. 125–136, 2000.

[46] M. Andrec and R. M. Levy, “Protein sequential resonance assignments by combinatorial enumeration using 13 Cα chemical shifts and their (i, i−1) sequential connectivities,” *J. Biomol. NMR*, vol. 23, no. 4, pp. 263–270, 2002.

[47] S. G. Hyberts and G. Wagner, “IBIS – a tool for automated sequential assignment

of protein spectra from triple resonance experiments.," *J. Biomol. NMR*, vol. 26, no. 4, pp. 335–344, 2003.

[48] B. E. Coggins and P. Zhou, "PACES: Protein sequential assignment by computer-assisted exhaustive search.," *J. Biomol. NMR*, vol. 26, no. 2, pp. 93–111, 2003.

[49] Y.-S. Jung and M. Zweckstetter, "Mars – robust automatic backbone assignment of proteins.," *J. Biomol. NMR*, vol. 30, no. 1, pp. 11–23, 2004.

[50] J. Wang, T. Wang, E. R. P. Zuiderweg, and G. M. Crippen, "CASA: an efficient automated assignment of protein mainchain NMR data using an ordered tree search algorithm.," *J. Biomol. NMR*, vol. 33, no. 4, pp. 261–79, Dec. 2005.

[51] H. R. Eghbalnia, A. Bahrami, L. Wang, A. Assadi, and J. L. Markley, "Probabilistic Identification of Spin Systems and their Assignments including Coil-Helix Inference as Output (PISTACHIO).," *J. Biomol. NMR*, vol. 32, no. 3, pp. 219–233, 2005.

[52] Xiang Wan and Guohui Lin, "CISA: Combined NMR Resonance Connectivity Information Determination and Sequential Assignment," *IEEE/ACM Trans. Comput. Biol. Bioinforma.*, vol. 4, no. 3, pp. 336–348, Jul. 2007.

[53] X. WAN and G. LIN, "GASA: A GRAPH-BASED AUTOMATED NMR BACKBONE RESONANCE SEQUENTIAL ASSIGNMENT PROGRAM," *J. Bioinform. Comput. Biol.*, vol. 5, no. 02a, pp. 313–333, Apr. 2007.

[54] M. Alipanahi, B., Gao, X., Karakoc, E., Balbach, F., Donaldson, L., Arrowsmith, C. and Li, "IPASS: Error tolerant NMR backbone resonance assignment by linear programming," *Univ. Waterloo Tech. Rep. CS-2009, 16*, 2009.

[55] G. M. Crippen, A. Rousaki, M. Revington, Y. Zhang, and E. R. P. Zuiderweg, "SAGA: rapid automatic mainchain NMR assignment for large proteins.," *J. Biomol. NMR*, vol. 46, no. 4, pp. 281–98, Apr. 2010.

[56] E. R. P. Zuiderweg, I. Bagai, P. Rossi, and E. B. Bertelsen, "EZ-ASSIGN, a program for exhaustive NMR chemical shift assignments of large proteins from complete or incomplete triple-resonance data," *J. Biomol. NMR*, vol. 57, no. 2, pp. 179–191, Oct. 2013.

[57] R. Tycko and K.-N. Hu, "A Monte Carlo/simulated annealing algorithm for sequential resonance assignment in solid state NMR of uniformly labeled proteins with magic-angle spinning.," *J. Magn. Reson.*, vol. 205, no. 2, pp. 304–14, Aug. 2010.

[58] H. N. B. Moseley, L. J. Sperling, and C. M. Rienstra, "Automated protein resonance assignments of magic angle spinning solid-state NMR spectra of β1 immunoglobulin binding domain of protein G (GB1).," *J. Biomol. NMR*, vol. 48, no. 3, pp. 123–8, Nov. 2010.

[59] E. Schmidt, J. Gath, B. Habenstein, F. Ravotti, K. Székely, M. Huber, L. Buchner,

A. Böckmann, B. H. Meier, and P. Güntert, “Automated solid-state NMR resonance assignment of protein microcrystals and amyloids,” *J. Biomol. NMR*, vol. 56, no. 3, pp. 243–254, May 2013.

[60] J. T. Nielsen, N. Kulminskaya, M. Bjerring, and N. C. Nielsen, “Automated robust and accurate assignment of protein resonances for solid state NMR,” *J. Biomol. NMR*, vol. 59, no. 2, pp. 119–134, Jun. 2014.

[61] G. T. M. Moseley, H.N., D. Monleon, “Automatic Determination of Protein Backbone Resonance Assignments from Triple Resonance Nuclear Magnetic Resonance Data,” vol. 339, no. 1994, pp. 91–108, 2001.

[62] H. N. B. Moseley, G. Sahota, and G. T. Montelione, “Assignment validation software suite for the evaluation and presentation of protein resonance assignment data.,” *J. Biomol. NMR*, vol. 28, no. 4, pp. 341–55, 2004.

[63] Y. J. Huang, H. N. B. Moseley, M. C. Baran, C. Arrowsmith, R. Powers, R. Tejero, T. Szyperski, and G. T. Montelione, “An integrated platform for automated analysis of protein NMR structures.,” *Methods Enzymol.*, vol. 394, pp. 111–141, 2005.

[64] H. Zhang, S. Neal, and D. S. Wishart, “RefDB: a database of uniformly referenced protein chemical shifts.,” *J. Biomol. NMR*, vol. 25, no. 3, pp. 173–195, 2003.

[65] A. Smelter and H. N. B. Moseley, “nmrstarlib - Python library that facilitates reading and writing NMR-STAR formatted files.” [Online]. Available: https://github.com/MoseleyBioinformaticsLab/nmrstarlib.

[66] H. Berman, K. Henrick, H. Nakamura, and J. L. Markley, “The worldwide Protein Data Bank (wwPDB): Ensuring a single, uniform archive of PDB data,” *Nucleic Acids Res.*, vol. 35, no. SUPPL. 1, 2007.

[67] S. R. Hall, “The STAR file: a new format for electronic data transfer and archiving,” *J. Chem. Inf. Model.*, vol. 31, no. 2, pp. 326–333, 1991.

[68] Python Software Foundation, “Python Language Reference, version 2.7,” *Python Software Foundation*. 2013.

[69] G. Van Rossum and F. L. Drake, “The Python Library Reference,” *October*, pp. 1–1144, 2010.

[70] A. Ronacher and R. Hettinger, “PEP 372 -- Adding an ordered dictionary to collections.” [Online]. Available: https://www.python.org/dev/peps/pep-0372/.

[71] “Python 2.7 Countdown.” [Online]. Available: https://pythonclock.org/.

[72] J. Doreleijers, “PyStarLib.” [Online]. Available: https://sourceforge.net/projects/pystarlib/.

[73] M. Fenwick, “NMRPyStar.” [Online]. Available:

https://github.com/mattfenwick/NMRPyStar.

[74] J. Wedell, “PyNMRSTAR.” [Online]. Available: https://github.com/uwbmrb/PyNMRSTAR.

[75] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, “Cython: The best of both worlds,” *Comput. Sci. Eng.*, vol. 13, no. 2, pp. 31–39, Mar. 2011.

[76] “graphviz Python library.” [Online]. Available: http://graphviz.readthedocs.io/en/latest/index.html.

[77] “docopt Python Library for creating command-line interfaces.” [Online]. Available: http://docopt.readthedocs.io/en/latest/.

[78] “Biological Magnetic Resonance Bank.” [Online]. Available: http://www.bmrb.wisc.edu/.

[79] “UltraJSON. UltraJSON is an ultra fast JSON encoder and decoder written in pure C with bindings for Python 2.5+ and 3.” [Online]. Available: https://github.com/esnme/ultrajson.

[80] J. Ooms, T. D. Lang, and H. Lloyd, “jsonlite: A Robust, High Performance JSON Parser and Generator for R.” [Online]. Available: https://cran.r-project.org/web/packages/jsonlite/index.html.

[81] “jQuery is a cross-platform JavaScript library.” [Online]. Available: http://jquery.com/.

[82] M. Yip, “RapidJSON - A fast JSON parser/generator for C++ with both SAX/DOM style API.” [Online]. Available: http://rapidjson.org/.

[83] M. D. Wilkinson, M. Dumontier, Ij. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. G. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. a. . ’t Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. a. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, “The FAIR Guiding Principles for scientific data management and stewardship,” *Sci. Data*, vol. 3, p. 160018, 2016.

[84] P. Guerry and T. Herrmann, *Advances in automated NMR protein structure determination.*, vol. 44, no. 3. 2011.

[85] E. Schmidt and P. Güntert, “A New Algorithm for Reliable and General NMR Resonance Assignment,” *J. Am. Chem. Soc.*, vol. 134, no. 30, pp. 12817–12829, Aug. 2012.

[86] M. C. Baran, Y. J. Huang, H. N. B. Moseley, and G. T. Montelione, “Automated analysis of protein NMR assignments and structures.,” *Chem. Rev.*, vol. 104, no. 8, pp. 3541–3556, 2004.

[87] H. N. B. Moseley, D. Monleon, and G. T. Montelione, “Automatic Determination of Protein Backbone Resonance Assignments from Triple Resonance Nuclear Magnetic Resonance Data,” no. 732, 2001, pp. 91–108.

[88] D. Monleón, K. Colson, H. N. B. Moseley, C. Anklin, R. Oswald, T. Szyperski, and G. T. Montelione, “Rapid analysis of protein backbone resonance assignments using cryogenic probes, a distributed Linux-based computing architecture, and an integrated set of spectral analysis tools,” in *Journal of Structural and Functional Genomics*, 2002, vol. 2, no. 2, pp. 93–101.

[89] L. Buchner, E. Schmidt, and P. Güntert, “Peakmatch: a simple and robust method for peak list matching.,” *J. Biomol. NMR*, vol. 55, no. 3, pp. 267–77, Mar. 2013.

[90] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,” in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

[91] T. D. Goddard and D. G. Kneller, “SPARKY 3, University of California, San Francisco.” [Online]. Available: http://www.cgl.ucsf.edu/home/sparky/.

[92] W. Lee, M. Tonelli, and J. L. Markley, “NMRFAM-SPARKY: enhanced software for biomolecular NMR spectroscopy.,” *Bioinformatics*, pp. 1–2, 2014.

[93] C. Bartels, T. he Xia, M. Billeter, P. Güntert, and K. Wüthrich, “The program XEASY for computer-supported NMR spectral analysis of biological macromolecules,” *J. Biomol. NMR*, vol. 6, no. 1, pp. 1–10, Jul. 1995.

[94] A. Smelter, M. Astra, and H. N. B. Moseley, “A fast and efficient python library for interfacing with the Biological Magnetic Resonance Data Bank,” *BMC Bioinformatics*, vol. 18, no. 1, p. 175, Dec. 2017.

[95] W. Feng, R. Tejero, D. E. Zimmerman, M. Inouye, and G. T. Montelione, “Solution NMR structure and backbone dynamics of the major cold-shock protein (CspA) from Escherichia coli: Evidence for conformational dynamics in the single-stranded RNA-binding site,” *Biochemistry*, vol. 37, no. 31, pp. 10881–10896, Aug. 1998.

[96] J. M. Aramini, J. L. Mills, T. B. Acton, M. J. Wu, T. Szyperski, and G. T. Montelione, “Resonance assignments for the hypothetical protein yggU from Escherichia coli,” *J. Biomol. NMR*, 2003.

[97] F. J. Moy, a P. Seddon, E. B. Campbell, P. Böhlen, and R. Powers, “1H, 15N, 13C and 13CO assignments and secondary structure determination of basic fibroblast growth factor using 3D heteronuclear NMR spectroscopy.,” *J. Biomol. NMR*, vol.

6, no. 3, pp. 245–254, 1995.

[98] J. M. Aramini, Y. J. Huang, J. R. Cort, S. Goldsmith-Fischman, R. Xiao, L. Y. Shih, C. K. Ho, J. Liu, B. Rost, B. Honig, M. A. Kennedy, T. B. Acton, and G. T. Montelione, “Solution NMR structure of the 30S ribosomal protein S28E from Pyrococcus horikoshii,” *Protein Sci*, vol. 12, no. 12, pp. 2823–2830, 2003.

[99] C. Chien, R. Tejero, Y. Huang, D. E. Zimmerman, C. B. Ríos, R. M. Krug, and G. T. Montelione, “A novel RNA-binding motif in influenza A virus non-structural protein 1,” *Nat. Struct. Biol.*, vol. 4, no. 11, pp. 891–895, Nov. 1997.

[100] S. Shimotakahara, C. B. Ríos, J. H. Laity, D. E. Zimmerman, H. A. Scheraga, and G. T. Montelione, “NMR Structural Analysis of an Analog of an Intermediate Formed in the Rate-Determining Step of One Pathway in the Oxidative Folding of Bovine Pancreatic Ribonuclease A: Automated Analysis of 1H, 13C, and 15N Resonance Assignments for Wild-Type and Mutant ,” *Biochemistry*, vol. 36, no. 23, pp. 6915–6929, Jun. 1997.

[101] D. Zheng, J. M. Aramini, and G. T. Montelione, “Validation of helical tilt angles in the solution NMR structure of the Z domain of Staphylococcal protein A by combined analysis of residual dipolar coupling and NOE data.,” *Protein Sci.*, vol. 13, no. 2, pp. 549–554, 2004.

[102] K. A. Mercier, M. Baran, V. Ramanathan, P. Revesz, R. Xiao, G. T. Montelione, and R. Powers, “FAST-NMR: Functional annotation screening technology using NMR spectroscopy,” *J. Am. Chem. Soc.*, vol. 128, no. 47, pp. 15292–15299, Nov. 2006.

[103] W. T. Franks, D. H. Zhou, B. J. Wylie, B. G. Money, D. T. Graesser, H. L. Frericks, G. Sahota, and C. M. Rienstra, “Magic-angle spinning solid-state NMR spectroscopy of the beta1 immunoglobulin binding domain of protein G (GB1): 15N and 13C chemical shift assignments and conformational analysis.,” *J. Am. Chem. Soc.*, vol. 127, no. 35, pp. 12291–12305, 2005.

[104] M. Tang, A. E. Nesbitt, L. J. Sperling, D. A. Berthold, C. D. Schwieters, R. B. Gennis, and C. M. Rienstra, “Structure of the Disulfide Bond Generating Membrane Protein DsbB in the Lipid Bilayer.,” *J. Mol. Biol.*, 2013.

[105] S. Yan, G. Hou, C. D. Schwieters, S. Ahmed, J. C. Williams, and T. Polenova, “Three-Dimensional Structure of CAP-Gly Domain of Mammalian Dynactin Determined by Magic Angle Spinning NMR Spectroscopy: Conformational Plasticity and Interactions with End-Binding Protein EB1.,” *J. Mol. Biol.*, 2013.

[106] A. R. Ranade Sanjay, “Point pattern matching by relaxation,” *Pattern Recognit.*, vol. 12, no. 4, pp. 269–275, 1980.

[107] J. Ton and A. K. Jain, “Registering Landsat Images By Point Matching,” *IEEE Trans. Geosci. Remote Sens.*, vol. 27, no. 5, pp. 642–651, 1989.

[108] S. Grzesiek and A. Bax, “Correlating backbone amide and side chain resonances in larger proteins by multiple relayed triple resonance NMR,” *J. Am. Chem. Soc.*, vol. 114, no. 16, pp. 6291–6293, 1992.

[109] W. T. Franks, K. D. Kloepper, B. J. Wylie, and C. M. Rienstra, “Four-dimensional heteronuclear correlation experiments for chemical shift assignment of solid proteins.,” *J. Biomol. NMR*, vol. 39, no. 2, pp. 107–31, Oct. 2007.

[110] R. J. G. B. Campello, D. Moulavi, and J. Sander, “Density-Based Clustering Based on Hierarchical Density Estimates,” *Adv. Knowl. Discov. Data Min.*, pp. 160–172, 2013.

[111] L. McInnes, J. Healy, and S. Astels, “hdbscan: Hierarchical density based clustering,” *J. Open Source Softw.*, vol. 2, no. 11, 2017.

[112] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, “The Maximum Clique Problem,” in *Handbook of Combinatorial Optimization: Supplement Volume A*, D.-Z. Du and P. M. Pardalos, Eds. Boston, MA: Springer US, 1999, pp. 1–74.

APPENDIX A
LIST OF ABBREVIATIONS

AA	Amino Acid
API	Application Programming Interface
BMRB	Biological Magnetic Resonance Bank
BMRBID	Biological Magnetic Resonance Bank identifier
CIF	Crystallographic Information File
DBSCAN	Density-based spatial clustering of applications with noise
FAIR	Findable, Accessible, Interoperable, and Reusable
HDBSCAN	Hierarchical DBSCAN
JSON	JavaScript Object Notation
MAS	Magic-angle spinning
NMR	Nuclear Magnetic Resonance
PDB	Protein Data Bank
PEP	Python Enhancement Proposal
REST	Representational state transfer
SS	Spin System/Sequence Site/Support Set
STAR	Self-defining Text Archive and Retrieval
UML	Unified Modeling Language
XML	Extensible Markup Language

APPENDIX B
SIMULATED PEAK LIST EXAMPLES

Table B1. Simulated CAN(CO)CA peak list (BMRBID 18397).

Peak #	Assignment	w1	w2	w3
1	Q2CA-N-M1CA	56.08998724415861	125.5070237879975	54.525460498759784
2	Y3CA-N-Q2CA	57.20000110173284	123.69704843317199	56.08195962645852
3	K4CA-N-Y3CA	55.19890411103603	122.4040373463508	57.20466677132368
4	L5CA-N-K4CA	53.048002366758276	126.2414736560956	55.200772909507045
5	I6CA-N-L5CA	60.04083023167545	126.08316380727885	53.055687323974645
6	L7CA-N-I6CA	54.843908683820025	126.7484359958205	60.024319587153016
7	N8CA-N-L7CA	50.80477470316926	124.88727263084405	54.84759761329681
8	G9CA-N-N8CA	44.73568034623651	109.48002972043292	50.77510404384722
9	K10CA-N-G9CA	59.376659781971654	120.78517773915708	44.71538168383893
10	T11CA-N-K10CA	62.23183686676209	106.60758954254734	59.40197617756998
11	L12CA-N-T11CA	54.59285604656336	127.34195466486149	62.23881152802562
12	K13CA-N-L12CA	53.55484172132135	123.0135344383692	54.614481170185954
13	G14CA-N-K13CA	45.1031945803211	105.5391973673235	53.565917961170605
14	E15CA-N-G14CA	53.925301336282736	121.2834690051228	45.09794477154259
15	T16CA-N-E15CA	60.26456108375081	115.48172495521821	53.926292640549505
16	T17CA-N-T16CA	60.37993888040222	115.4611235779842	60.28362710201867
17	T18CA-N-T17CA	61.68176980806341	115.98146739315655	60.34683729710857
18	E19CA-N-T18CA	54.315207575711064	125.04026037959156	61.697225780355865
19	A20CA-N-E19CA	50.8075654636244	125.33685015063142	54.33535323274808
20	V21CA-N-A20CA	63.7462903217478	116.46664354199123	50.80310089905041
21	D22CA-N-V21CA	52.467302209239264	115.53885710836218	63.716091315093756
22	A23CA-N-D22CA	54.751402457453715	123.00540186128194	52.47199639441082
23	A24CA-N-A23CA	54.71918961369314	120.61838059750413	54.75535425000724
24	T25CA-N-A24CA	67.54804736653017	117.0207638536372	54.71433863629124
25	A26CA-N-T25CA	55.23169170190556	123.8771133515964	67.56202665719907
26	E27CA-N-A26CA	59.30816012761388	116.1937041980287	55.25769802670506
27	K28CA-N-E27CA	60.49374416024143	117.22561568240506	59.298809306262626
28	V29CA-N-K28CA	66.59752605340188	119.03260437326401	60.49367000849438
29	F30CA-N-V29CA	57.40814705133402	118.3496235793004	66.59403419519356
30	K31CA-N-F30CA	60.30834382115652	120.64412820576668	57.42500048936475
31	Q32CA-N-K31CA	59.04020387048314	121.12276325017254	60.3026059624695
32	Y33CA-N-Q32CA	61.8671419247353	120.7252462887843	59.04275345941449
33	A34CA-N-Y33CA	56.30211439342077	122.51277739864445	61.87668317121093
34	N35CA-N-A34CA	57.26094994588013	118.06872499286543	56.30356175918259
35	D36CA-N-N35CA	56.0792143510658	120.97634138933645	57.268024574771566
36	N37CA-N-D36CA	53.70133445562052	114.75213159778963	56.086334221846656
37	G38CA-N-N37CA	47.03979494212038	108.27566081302902	53.694149384643694
38	V39CA-N-G38CA	61.944180117520915	121.64640813122338	47.031608819815276
39	D40CA-N-V39CA	52.63109904188199	130.6656535021392	61.95519784135377
40	G41CA-N-D40CA	45.25039333567271	108.10334189555662	52.64047347336225
41	E42CA-N-G41CA	54.92049770506791	118.25607966797557	45.234388010907765
42	W43CA-N-E42CA	57.71462979526749	124.70285485231021	54.917648457628715
43	T44CA-N-W43CA	61.176160986159225	109.00706387073404	57.699085841865674
44	Y45CA-N-T44CA	58.055321401596764	118.37314066265702	61.15645822140491
45	D46CA-N-Y45CA	50.84120126126258	126.29620170792327	58.0790406586594
46	D47CA-N-D46CA	54.69631563448615	123.23755520756762	50.837987492121286
47	A48CA-N-D47CA	54.026517417904365	118.39033648414215	54.67854798619943
48	T49CA-N-A48CA	60.44341637877914	104.18612045224553	54.039361628991436
49	K50CA-N-T49CA	55.504187568702015	119.38243748231302	60.47060213396953
50	T51CA-N-K50CA	62.63757501160989	111.9001557495533	55.51019041112089
51	F52CA-N-T51CA	56.684820301932	130.2686330371651	62.65321259602127
52	T53CA-N-F52CA	60.44503015990373	111.92772371737388	56.68763529821447
53	V54CA-N-T53CA	58.6692707688915	118.1164215364167	60.44317072794205
54	T55CA-N-V54CA	61.497516969059	123.7899940409246	58.669400478059025
55	E56CA-N-T54CA	57.55348493758591	131.08470567790837	61.51666806413445

Table B2. Simulated NCACX peak list (BMRBID 18397).

Peak #	Assignment	w1	w2	w3
1	Q2N-CA-C	126.37474657752463	56.63837257155845	174.86629358773598
2	Q2N-CA-CA	126.36848363529755	56.63909646382991	56.09984762511398
3	Q2N-CA-CB	126.34732142151232	56.62611167606748	30.32104339137229
4	Q2N-CA-CG	126.375763777546	56.636808884098016	35.60240123815321
5	Q2N-CA-CD	126.3547395847522	56.6254767894534	180.2821691525966
6	Y3N-CA-C	124.55185938571728	57.75035516123179	174.8713848011184
7	Y3N-CA-CA	124.57779013310311	57.76572988480728	57.19197720651176
8	Y3N-CA-CB	124.58043473114724	57.742548316586834	43.54553001450174
9	K4N-CA-C	123.27310087371445	55.75636535659763	173.31591088008074
10	K4N-CA-CA	123.26760273614376	55.753256079534076	55.20559552046438
11	K4N-CA-CB	123.28584214768209	55.752685544699	36.277874628349196
12	K4N-CA-CG	123.28278447883608	55.7454444953196	25.660111580679896
13	K4N-CA-CD	123.27539503899574	55.7391050161097	29.06301344409681
14	L5N-CA-C	127.10077465242769	53.60825174227536	174.66265808434846
15	L5N-CA-CA	127.11655134416607	53.600223467763094	53.04340680554794
16	I6N-CA-C	126.95090592637777	60.59505728435444	175.07989575651135
17	I6N-CA-CA	126.93753738495685	60.59551959878378	60.02525384609561
18	I6N-CA-CB	126.95111521659469	60.60710529611978	37.875300412596864
19	L7N-CA-C	127.63198815463531	55.391522551687935	174.8866862743816
20	L7N-CA-CA	127.65178907281144	55.398030176630456	54.84601982534066
21	L7N-CA-CB	127.62174700261683	55.40347971244275	42.994465645654
22	L7N-CA-CD1	127.63514102596997	55.391225260944125	26.055416546255508
23	L7N-CA-CD2	127.61853837911471	55.39661458457827	25.084864823507235
24	N8N-CA-C	125.73920814797353	51.325079638782604	176.22662058542198
25	N8N-CA-CA	125.7249632690477	51.33186412395423	50.79191035118426
26	N8N-CA-CB	125.7508700077936	51.33133148601983	38.37360796424758
27	N8N-CA-CG	125.74594466910337	51.35453565659011	176.31264849270488
28	G9N-CA-C	110.38444242372643	45.275378346770864	173.03710755336644
29	G9N-CA-CA	110.35413233446099	45.27493644220507	44.720004400346326
30	K10N-CA-C	121.67235017176056	59.957514252328856	178.9675708873985
31	K10N-CA-CA	121.69263149006557	59.941956406431906	59.40748121574479
32	K10N-CA-CB	121.66236041631812	59.94674673579991	32.85016130525905
33	K10N-CA-CG	121.67811082346287	59.954182814619664	25.636279113972968
34	K10N-CA-CD	121.6739945619845	59.94632387100416	29.291196645542943
35	T11N-CA-C	107.49933636170229	62.78848912695365	173.2002028994154
36	T11N-CA-CA	107.48092284881253	62.7824230570412	62.25504959718718
37	T11N-CA-CB	107.50132872660146	62.797680634756475	69.80697875609283
38	T11N-CA-CG2	107.50123762141583	62.79237956575546	22.698017785627126
39	L12N-CA-C	128.22473669835472	55.1548346700391	173.59638531183595
40	L12N-CA-CA	128.2101522859729	55.13668264439651	54.60752524008299
41	L12N-CA-CB	128.23429973015402	55.16289023081968	43.29448379232769
42	L12N-CA-CD1	128.23582560222306	55.17667176791743	26.132079805118526
43	K13N-CA-C	123.88680512466497	54.09482859703417	175.5894423407792
44	K13N-CA-CA	123.90910204695982	54.112505896538615	53.55635857732763
45	G14N-CA-C	106.40963109971118	45.62730266717992	171.1294485889564
46	G14N-CA-CA	106.38370693106117	45.644198187067964	45.1013947739578
47	E15N-CA-C	122.13534646911859	54.475786135327255	173.74656480364177
48	E15N-CA-CA	122.13182586905461	54.47640950143144	53.94300109565274
49	E15N-CA-CB	122.13681110000044	54.46723453705194	33.47279643652776
50	T16N-CA-C	116.347138459632	60.82539697932505	171.76867666228054
51	T16N-CA-CA	116.36624383735669	60.82192013747659	60.27999033174986
52	T16N-CA-CB	116.34653014785314	60.83419246316472	70.62702785346805
53	T16N-CA-CG2	116.36782097410823	60.83636291218967	20.011123900906842
54	T17N-CA-C	116.33193327058657	60.928013878469734	173.79854449498035
55	T17N-CA-CA	116.32089518646767	60.90909365461843	60.37822834928552
56	T17N-CA-CB	116.31161508068658	60.93116461515987	73.00671073225485
57	T17N-CA-CG2	116.31216355260378	60.92039938739741	21.440154763842948
58	T18N-CA-C	116.85954493995624	62.2140286147248	171.04737482786783
59	T18N-CA-CA	116.84851836092865	62.220921554885784	61.67639854852682
60	T18N-CA-CB	116.85304519526116	62.22361385045373	70.99262953094194
61	T18N-CA-CG2	116.84536991369178	62.233560855712454	18.72884215261144

62	E19N-CA-C	125.91997008802345	54.878857928913604	175.61006913369428
63	E19N-CA-CA	125.89990754162018	54.887540467440424	54.33934971676499
64	A20N-CA-C	126.20370700883457	51.363543827802204	177.616293363257 9
65	A20N-CA-CA	126.19496903387859	51.37050942146125	50.814165289088976
66	A20N-CA-CB	126.22294528702804	51.34072346537154	23.706026727248624
67	V21N-CA-C	117.3289948272376	64.28412509088761	174.7105500957272
68	V21N-CA-CA	117.36343973374419	64.29136590015015	63.733954361846735
69	V21N-CA-CB	117.35669604594011	64.29370801617658	31.96235744172191
70	V21N-CA-CG1	117.33764461189564	64.28084541979402	20.99281070284152
71	D22N-CA-C	116.40237400699012	53.008656273434596	174.90692770091422
72	D22N-CA-CA	116.40101788148584	53.02951005550764	52.47770190969666
73	D22N-CA-CB	116.38942630073623	53.02109094239296	42.661300163449695
74	A23N-CA-C	123.87053163720914	55.308355904032105	179.76830179787635
75	A23N-CA-CA	123.86686902897017	55.309579220810846	54.75719029920559
76	A23N-CA-CB	123.87818981868088	55.30148414422537	18.23870428487891
77	A24N-CA-C	121.46753955594083	55.26274614608674	181.38180928104723
78	A24N-CA-CA	121.46957541346823	55.27360023942305	54.70910222055376
79	A24N-CA-CB	121.47903707039582	55.27448525452084	18.151226171519273
80	T25N-CA-C	117.89375946523226	68.12068459821047	175.64080434343694
81	T25N-CA-CA	117.89075964522748	68.10675773103603	67.55610069355237
82	T25N-CA-CB	117.87796969046943	68.12314506180006	67.59754372030436
83	T25N-CA-CG2	117.88112350141881	68.10358232878185	21.298333369389805
84	A26N-CA-C	124.75638218456216	55.780121558478854	177.25642668156806
85	A26N-CA-CA	124.75874040778872	55.786456593490904	55.24269656537311
86	A26N-CA-CB	124.77235040807447	55.80354643829877	17.5643138044001 1
87	E27N-CA-C	117.05847561947884	59.86190050683349	177.70624092921577
88	E27N-CA-CA	117.05485045183815	59.84840213229669	59.32109096188497
89	E27N-CA-CB	117.05118101982632	59.849682621888014	29.15000421150389
90	K28N-CA-C	118.09505466626433	61.04705204471706	178.7734112065052
91	K28N-CA-CA	118.0939000257617	61.042653603355625	60.509586644256075
92	K28N-CA-CB	118.0916800615009	61.04908473200433	32.783749561137036
93	V29N-CA-C	119.9006081974715	67.15892388061484	178.5822623944007
94	V29N-CA-CA	119.89185406576382	67.14035596986682	66.5879119196996 5
95	V29N-CA-CB	119.90913855826857	67.15506910322482	32.01118932661409
96	V29N-CA-CG1	119.90858666178023	67.17045065446547	22.24092873909696
97	V29N-CA-CG2	119.91438508895064	67.1491728443379	21.109995916599
98	F30N-CA-C	119.23343472195869	57.95686248999264	178.91122383167908
99	F30N-CA-CA	119.20364274454032	57.96902115702179	57.40945460472741
100	K31N-CA-C	121.50660521297785	60.86719274039746	179.57761026422074
101	K31N-CA-CA	121.50855080931062	60.85926700104904	60.30284945992987
102	K31N-CA-CB	121.50923419472835	60.87224979232287	31.855263682892947
103	K31N-CA-CD	121.4826302795287	60.85720820745724	29.2210103426287 7
104	Q32N-CA-C	122.00152088918666	59.60779339759821	177.37285277147396
105	Q32N-CA-CA	122.013457095649	59.60525657633825	59.054204476353924
106	Q32N-CA-CB	121.99686477196575	59.595574828347196	28.97196074214277
107	Q32N-CA-CD	121.99662317590283	59.59504032927243	179.75144234464221
108	Y33N-CA-C	121.60041319684575	62.40477718361685	178.53702321256395
109	Y33N-CA-CA	121.59049181412631	62.42931543709156	61.89149735873813
110	Y33N-CA-CB	121.58811299553751	62.42494826325337	38.89482678981095
111	A34N-CA-C	123.37370035386915	56.839881594327835	179.40955539643974
112	A34N-CA-CA	123.37972630831551	56.840218853595545	56.315132802130584
113	A34N-CA-CB	123.38625431095323	56.84305762772488	17.982836354109317
114	N35N-CA-C	118.9406336251923	57.825531839639886	179.462504273583
115	N35N-CA-CA	118.94740254747359	57.824577097277775	57.288226160597816
116	N35N-CA-CB	118.94711248200271	57.81383718031868	39.41509769381999
117	N35N-CA-CG	118.92418997973732	57.83465801509005	176.00039393758652
118	D36N-CA-C	121.8548370216947	56.63833954114289	175.89137293219727
119	D36N-CA-CA	121.8523981031267	56.61702387426289	56.057900034784986
120	D36N-CA-CB	121.83886993367527	56.62477194354053	38.44322027893197
121	N37N-CA-C	115.61368574050466	54.24398857303175	174.0319639980829
122	N37N-CA-CA	115.61214139468052	54.24561915071859	53.681875502774716
123	N37N-CA-CB	115.61353484717496	54.250528057453096	40.47090742236391
124	N37N-CA-CG	115.62028168956367	54.239985953039664	176.6181834480967
125	G38N-CA-C	109.14548455939055	47.595962539867045	173.8501364888506
126	G38N-CA-CA	109.14739848140712	47.59365971925514	47.02300923613492
127	V39N-CA-C	122.51193383597331	62.51333309268938	174.9787456224277
128	V39N-CA-CA	122.49143044818096	62.50703953588854	61.94843251118124

129	V39N-CA-CB	122.49787695427139	62.48365249826458	31.83299167011932
130	V39N-CA-CG1	122.48683381881233	62.509296847047544	22.018150116703108
131	D40N-CA-C	131.5302166473598	53.19724131117937	174.73988738326102
132	D40N-CA-CA	131.52399153365718	53.188298144800676	52.64001453716309
133	D40N-CA-CB	131.52671133680525	53.184327349893366	41.18262373759988
134	G41N-CA-C	108.99310048490746	45.789239759605266	172.63503043742523
135	G41N-CA-CA	108.98514596752801	45.81594816803027	45.258775102477195
136	E42N-CA-C	119.12415670603133	55.469864652076765	177.77166830099642
137	E42N-CA-CA	119.12169805951353	55.467167471277506	54.9159124708808
138	E42N-CA-CB	119.12648460103468	55.47278387308149	31.066074731666365
139	W43N-CA-C	125.57693265292565	58.2519204142371	177.16169891663364
140	W43N-CA-CA	125.57039761013453	58.27062615097913	57.70369967212549
141	W43N-CA-CB	125.56402469889645	58.248283966462694	33.66699787163827
142	T44N-CA-C	109.86345917142916	61.712178267494956	173.74585990775753
143	T44N-CA-CA	109.88612622336083	61.70938990522632	61.166038910362566
144	T44N-CA-CG2	109.89042219397506	61.71269310091192	20.90478627656454
145	Y45N-CA-C	119.26542199210138	58.615932158556525	171.76075789735305
146	Y45N-CA-CA	119.25138375738929	58.61158612871409	58.06754244771449
147	D46N-CA-C	127.17219745395516	51.3997599552715	175.75364322992743
148	D46N-CA-CA	127.18145585289383	51.39403084491661	50.851169152758104
149	D46N-CA-CB	127.16742287011871	51.37419221456706	42.08345374475197
150	D47N-CA-C	124.1319952728446	55.24686686310918	177.01120423743282
151	D47N-CA-CA	124.09369997045992	55.24227652966274	54.7045059564102
152	D47N-CA-CB	124.1107202640821	55.25202038902626	42.94369452314282
153	A48N-CA-C	119.24074820975306	54.58603759525219	179.3610321505171
154	A48N-CA-CA	119.26159102786877	54.58200003103815	54.02961923736523
155	A48N-CA-CB	119.26022101642715	54.559479867792405	19.067257851245046
156	T49N-CA-C	105.05528316359172	61.00315567598269	175.70218181747381
157	T49N-CA-CA	105.08407379610355	61.0033259579515	60.45380112030202
158	T49N-CA-CB	105.06716949084783	61.01347744424112	69.89089292977744
159	T49N-CA-CG2	105.0567881832911	61.00359578933136	21.597138806238142
160	K50N-CA-C	120.24627385758113	56.05459168231778	175.23185933646604
161	K50N-CA-CA	120.25416097643715	56.04556113359048	55.50547676857063
162	T51N-CA-C	112.74039365109172	63.21868144313733	174.12394975571473
163	T51N-CA-CA	112.76528581100708	63.21923940676241	62.66243410403021
164	T51N-CA-CB	112.73970269569332	63.20858254122815	71.82358285881497
165	T51N-CA-CG2	112.75991885950224	63.20587990749114	21.002262912858594
166	F52N-CA-C	131.13125492508357	57.21320870587695	175.58142320920783
167	F52N-CA-CA	131.1314438227842	57.229474832302316	56.65597922952022
168	F52N-CA-CB	131.12706491027492	57.231286541428275	43.475757775045196
169	T53N-CA-C	112.78996177971143	60.97901833393373	171.89594336767078
170	T53N-CA-CA	112.79081329410026	60.98305975548514	60.43837648544439
171	T53N-CA-CG2	112.78904130462386	60.992470926590634	20.973646358302172
172	V54N-CA-C	118.99275773585974	59.207268606992834	172.45136390133092
173	V54N-CA-CA	118.99834647577225	59.21412762175015	58.65922963840883
174	V54N-CA-CB	118.98670003930283	59.20111949657525	32.583696560336314
175	V54N-CA-CG1	119.00355191569214	59.216122769631696	21.884922175165705
176	T55N-CA-C	124.67322515827635	62.06218204601004	174.05241664395675
177	T55N-CA-CA	124.66321191520731	62.064664085015686	61.50458244677173
178	T55N-CA-CB	124.66565955596148	62.06012250164029	72.30408457680252
179	T55N-CA-CG2	124.66733375803453	62.05426872860013	21.281331374517737
180	E56N-CA-C	131.9504484180766	58.09501631293763	180.10593984109164
181	E56N-CA-CA	131.9448435612036	58.11078502563426	57.56516414719059
182	E56N-CA-CB	131.95917123459583	58.105543240951	33.17020972688592

CURRICULUM VITAE

Andrey Smelter
2405 Sherry Rd • Louisville, KY 40217
andrey.smelter@gmail.com • 502-403-0973

EDUCATION

2013-2017 **University of Louisville, Louisville, KY 40292**
Department of Computer Engineering and Computer Science
Ph.D. candidate in Interdisciplinary Studies: Concentration in Bioinformatics

2010-2013 **University of Louisville, Louisville, KY 40292**
Department of Chemistry
M.S. Chemistry

2005-2010 **Perm State University, Perm, Russia, 614990**
Department of Chemistry
Specialist Degree in Chemistry

RESEARCH EXPERIENCE

2013-present Ph.D. Candidate

Development of software tools for analysis of protein Nuclear Magnetic Resonance spectral data. Developing and refining the grouping, typing, linking and mapping algorithms and data structures needed to automate the protein resonance assignment of solution and solid-state NMR spectral data. Development of software libraries to facilitate access and manipulation of protein NMR data deposited in Biological Magnetic Resonance Data Bank.

2011-2013 M.S. Chemistry

Developed data structures to model and automate resonance assignment of protein Nuclear Magnetic Resonance spectral data.

2009-2010 Specialist Degree in Chemistry

Studied chemical kinetics of aluminum reduction from alkylbenzene aluminizing electrolytes using polarization and impedance methods.

SOFTWARE ENGINEERING

- **Programming Languages:** Python, working knowledge of C++, R
- **Python packages:** NumPy, SciPy, Jupyter, Cython, bokeh, matplotlib, PyQt.
- **Software documentation:** sphinx, Doxygen.
- **Software testing:** pytest, unittest.
- **Version control:** Git, Github, Gitlab.

PUBLICATIONS

- Andrey Smelter, Eric C. Rouchka, and Hunter N.B. Moseley. “Detecting and accounting for multiple sources of positional variance in peak list registration and spin system grouping.” Submitted to Journal of Biomolecular NMR.
- Andrey Smelter, Morgan Astra, and Hunter NB Moseley. “A fast and efficient python library for interfacing with the Biological Magnetic Resonance Data Bank.” BMC bioinformatics 18.1 (2017): 175.

POSTER PRESENTATIONS

- Andrey Smelter, Indraneel Reddy, Eric C. Rouchka, Hunter N.B. Moseley (2015) “Automated Assignment of Magic-Angle-Spinning Solid-State Protein NMR Spectra” UT-KBRIN Bioinformatics Summit, Buchanan, TN
- Andrey Smelter, Eric C. Rouchka, Hunter N.B. Moseley (2016) “Automated Assignment of Magic-Angle-Spinning Solid-State Protein NMR Spectra” UT-KBRIN Bioinformatics Summit, Cadiz, KY
- Andrey Smelter, Xi Chen, Eric C. Rouchka, Hunter N.B. Moseley “Registration and grouping algorithms in protein NMR derived peak lists and their application in protein NMR reference correction.” Biophysical Society Meeting (2017), New Orleans, LA
- Andrey Smelter, Xi Chen, Eric C. Rouchka, Hunter N.B. Moseley “Registration and grouping algorithms in protein NMR derived peak lists and their application in protein NMR reference correction” UT-KBRIN Bioinformatics Summit, Burns, TN
- Andrey Smelter, Morgan Astra, Hunter N.B. Moseley “A Fast and Efficient Python Library for Interfacing with the Biological Magnetic Resonance Data Bank” UT-KBRIN Bioinformatics Summit, Burns, TN

TEACHING EXPERIENCE

Teaching Assistant, Department of Chemistry, University of Louisville (2010-2013):

- CHEM 103: Introduction to Chemistry Lab
- CHEM 207: Introduction to Chemical Analysis I Lab
- CHEM 208: Introduction to Chemical Analysis II Lab
- CHEM 343: Organic Chemistry Laboratory I
- CHEM 441: Elements of Physical Chemistry Recitation

HONORS AND AWARDS

- 5-Year Full Tuition Scholarship, Perm State University (2005-2010).

- Awarded with a special scholarship from one of the largest oil companies in Russia (LUKOIL) as one of the best students of Department of Chemistry in Perm State University (2009-2010).