

Abstract

Geometric Studies of RNA and Ribosomes, and Ribosome Crystallization.

Neil R Voss

2006

Traditionally, those interested in the way biomolecules pack have focused their research on proteins. However, RNAs also form tertiary structures characterized by complex packing interactions, but due to the lack of information about RNA structures, no investigations of the way RNAs pack have been reported in the literature. This dissertation addresses the issue of how RNA molecules pack on both small and large scales, especially in ribosomes, and compares RNA packing to protein packing.

Using Voronoi volume calculations on RNA crystal structures, a self-consistent, universal set of atomic volumes was obtained. The atomic volumes obtained from this analysis are consistent across different RNA structures and packing environments. When the volumes of atom types common to both proteins and RNAs were compared, it was found that RNA atoms have smaller volumes, which implies that they are more tightly packed than they would be in protein.

Using the rolling probe method, a methodology was developed for delimiting the exterior of the large ribosomal subunit so that its interior packing could be investigated. The subunit interior contains much more solvent (39% v/v) than the interior of most large protein assemblies. Almost all of the solvent-filled voids in the large subunit are connected both to its exterior and to each other by passages big enough for water molecules to pass.

The polypeptide exit tunnel is a major component of the system of solvent channels that permeates the entire subunit and its geometry was extracted from large ribosomal subunit crystal structure using the rolling probe method. Since water and other small molecules can diffuse into and out of the tunnel along many different trajectories, the large subunit cannot be part of the seal that keeps ions from passing through the ribosome-translocon complex. For objects of the size of nascent peptides, however, it is effectively an unbranched tube connecting the peptidyl transferase center of the large subunit and the site where nascent peptides emerge. At no point is the tunnel big enough to accommodate folded polypeptides larger than alpha-helices.

Ribosomes from different animals were purified and investigated as possible sources for ribosome crystallization to expand our knowledge of eukaryotic ribosome structure.

Geometric Studies of RNA and Ribosomes, and Ribosome Crystallization

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Neil R. Voss

Dissertation Director: Prof. Peter B. Moore

November 2006

© 2006 by Neil R Voss.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with Invariant Sections being the Front-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.



In desperation I asked [Enrico] Fermi whether he was not impressed by the agreement between our calculated numbers and his measured numbers. He replied, “How many arbitrary parameters did you use for your calculations?” I thought for a moment about our cut-off procedures and said, “Four.” He said, “I remember my friend Johnny von Neumann used to say, with four parameters I can fit an elephant, and with five I can make him wiggle his trunk.” With that, the conversation was over.

— Freeman Dyson, January 2004

“A meeting with Enrico Fermi”, *Nature* **427**, p. 297, DOI: 10.1038/427297a

During the course of the symposium a semantic difficulty became apparent. To some of the participants, microsomes mean the ribonucleoprotein particles of the microsome fraction contaminated by other protein and lipid material; to others, the microsomes consist of protein and lipid contaminated by particles. The phrase “microsomal particles” does not seem adequate, and “ribonucleoprotein particles of the microsome fraction” is much too awkward. During the meeting the word “ribosome” was suggested; this seems a very satisfactory name, and it has a pleasant sound. The present confusion would be eliminated if “ribosome” were adopted to designate ribonucleoprotein particles in the size range 20 to 100S.

— Richards B. Roberts, February 1958

“Introduction” in *Microsomal Particles and Protein Synthesis*. New York: Pergamon Press. pp. vii–viii

Acknowledgements

My committee

First, I would like to thank Peter Moore. Peter has been a great advisor. Though my thesis project was eukaryotic ribosome crystallization, he was very open to me exploring computational projects before any results were available. Peter maintains an encyclopedia of ribosome information in his head. Buffer conditions, what experiments were performed, *etc.* Despite this, he and I had several comical communication problems, mainly because I have a tendency to use too many pronouns and leave out key information when I speak. I also tend to have selective listening and typically try to answer questions without hearing all the facts. One memorable example when I bought a book for myself that the lab already owned, Peter inquired, “why did you buy this book, we already have a copy?” to which my reply was, “It’s okay, I bought it.” After a couple more exchanges of the same phrases, I finally replied, “I bought the book with *my own money*,” which was of course fine. Jaime Williamson once called Peter the “scariest man in RNA,” but this could not be further from the truth, because Peter makes all lab members feel like members of his family.

Tom Steitz and I interacted mainly through the bi-weekly presentations called “ribosome meeting.” We also engaged in committee meetings and a couple of brain storming session about my project. It was Tom came up with most of the relevant questions addressed in the exit tunnel chapter along with a good way to approach them. He was also good at making sure Peter, Gerstein, and I concentrated on the biology and did not follow any tangents into silly computer calculations. Tom has a firm grasp of how any results can be interpreted and he was always very careful not over interpret the data. Tom provides tough-love and the few times he has given me praise were memorable experiences with I recall well.

Starting with my first rotation, Mark Gerstein has been almost a second advisor to me. I owe a lot to Mark for helping me with the growing pains of becoming a scientist. The most memorable interaction with Gerstein was the time I was in his office past 1:00 in the morning working on the reviewers’ comments for my first paper (Voss & Gerstein, 2005). During meetings, he always had some new gadget he was trying to figure out when I visited his office, from an electronic pen and a color-changing orb to a machine that scans paper notes and emails you a PDF. Mark is very

energetic and a master of extracting information from existing data. Mark’s lab is poised to make several more important contributions in the future to which I look forward to.

The fourth member of my committee, João Morais-Cabral was a great asset. João and I also worked together for three years teaching the X-ray crystallography course. His lectures were always very entertaining. This best day in X-ray was when the students learned about Patterson and autocorrelation functions. João would bring in two slides with the same delta functions and then he would slide them amongst each other saying “tuk tuk tuk tuk,” presumably the noise graphs make when they slide amongst each other.

Also Steve Harvey who acted as my outside reader gave several suggestions that strengthened the quality of this thesis.

I accepted a post-doctorate job with Bridget Carragher and Clint Potter back in May, it is now December and I thank them for being patient with me. It took a long time to compile all these side projects of mine into the grand document.

The department staff and faculty

When I first came to Yale, I had not run a gel or even used a pipette since high school and had no clue what a “buffer” was. Therefore, I would like to thank Fred Sigworth and his researcher Dave Chester for teaching the basics of biochemistry lab work during my second rotation. I would also like to thank the members of the Jennifer Doudna lab, specifically Dan Battle, Jeff Kieft, Angie Grech, Rob Rambo, and Rob Batey for furthering my training during my third rotation.

I would also like to thank Vinzenz Unger. Vinzenz was not formally on my committee, but I turned to him several times for advice. Vinzenz was also pivotal in helping me pass my qualifying exam. Nessie Stewart and Florine Dombroski the ladies who run the MB&B department were always extremely helpful through the years. Several other professors have helped me along the way. Scott Strobel and Rachel Green (from Johns Hopkins) were fun to converse with about RNA at RNA Society meetings. The PI’s of our neighboring labs of Pat Loria and Kurt Zilm have been memorable. Several of the new junior faculty have also offered positive encouragement and advice throughout my time at Yale.

The greater Moore lab

I would like to thank the Moore lab members of the past. Josh Warren’s interesting research caused me to do a fourth rotation in the lab just before he left. Huijing Shi initially trained me in

ribosome purification. Betty Freeborn was knew everything about of ribosome biochemistry and I tried to extract as much information from her as I could before she left. Pramodh Vallurupalli, who was so fed up with me asking computer questions gave me root access early on to my benefit. I always enjoyed working with Pramodh, who was an extremely quiet individual, but past 2:00 in the morning where he would come alive as he intensely followed his favorite cricket team in India. Szilvia Szep taught me the basics of crystallography. Cathy Filotto-Turner provided me with all the department gossip and was a pleasure to talk with. Helen Merianos is probably the hardest worker I have ever met and provided much of the lab energy and helped make the lab more efficient. Susan Schroeder had several great ideas for my project that I appreciated until later. Daqi Tu and his ugly exit tunnel slides gave me the idea to come up with a better way of visualizing the exit tunnel. Tiantian Gu, though only here for a short period of time, was a good friend. Our dishwasher, Marge Barilli was always fun to chat with at 5:00 in the morning. I would also like to thank the undergraduates: Pete Okolo, Yvonne Lai, Sammy Levy, and Sam Crayton for providing tons of entertainment. My rotation student James Lipchock was a pleasure to work with and helped me get off the ground again on my final wet lab project. Elizabeth Pollock always listened to my crazy ideas and was a great friend. Elizabeth's mastery of sarcasm provided hours of entertainment. And last, but not least, I would like to thank Joshua Young who my closest friend throughout my years in the lab. Josh and I typically got lunch together, argued about music, and always clarified what I was trying to say so that Peter could understand. I think Josh and I together would like to thank Les and his "comfort foods" whose knowledge of spiciness pushed the limits of the human taste bud.

I am also leaving behind four more members of the lab: Young-jin Cho, Guliz Gurel, Hong Jin, and Larysa Vasylenko. Hong and I started in the lab about a month apart and finished approximately at the same time together. When Hong first joined the lab she still had difficulties with English, but you would never know it, because she barely had an accent and would always agree with you when she did not understand. Today, she obviously has a firm grasp of English and I hope I've contributed to her slang vocabulary. Larysa joined the lab as our research assistant shortly after I did. Larysa was first to break ground with purifying ribosomes from *Artemia* removing any pitfalls that I would surely fall in. Guliz was a woman of many interests and it was a pleasure to work with her and it looks like she will have a sucessful career. Young-jin was new to the lab, but it was a pleasure working with them during this short time. Young-jin is also taking over my root duties in the computer lab, so I wish him good-luck.

The Steitz and Gerstein labs

During my time, a large group of ribosome people from the Moore and Steitz labs met to present our work. Of these people Gregor Blaha was the most conversational. Gregor always asked me tough questions that were relevant to my project and, in essence, made this work more complete. Daniel Klein was a fellow member who studied the ribosome computationally and had several interesting ideas. Martin Schmeing, Jeff Hansen, Yong Xiong, and Jennifer Kavran also provided assistance to me over the years. Miljan Simonovic, Axel Innis, and Robin Evans were recent additions to the group but brought a ton of energy. Ivan Lomakin was an expert at ribosome preparations and gave me good advice for my subunit preparations. Steitz lab researchers Joe Watson, Bob Grodzicki and Bill Eliason were very helpful when trying to use any of the Steitz lab equipment.

My association with the Gerstein lab was unofficial and very sporadic, but also highly productive. Initially during my rotation with Mark, I worked with Nicholas Luscombe, who helped me get situated to New Haven. Later, Mark placed me in his membrane-helix subgroup where I worked with Ursula Lehnert, Anne Counterman, Yu (Brandon) Xia, and Zhiyun (Eric) Yu, all of whom introduced me to several topics in computational biology. Later, I joined the structure subgroup with Nat Echols, Samuel Flores, Long (Jason) Lu, Prianka Patel, and Alexander (Sasha) Karpikov. The structure subgroup was interested in what could we learn about structures and I found this collaboration very helpful. I also had the pleasure of working with two students from the Gerstein lab: undergraduate Julian Graham modified the libproteingeom code and designed a beautiful webpage in one semester, which still exists today. Kevin Keating a rotation student (now in the Anna Pyle lab) and I always had intelligent discussions about our projects together.

Friends at Yale

Upon coming to Yale, the first years all joined to form a group we called the “MB&B All Stars,” which included Steve Aller, Jesse Cochrane, Angela DeLucia, Catherine Eakin, Bethany Krett, Ben Koo, Stephanie Herring, Jason Ptacek, Ingrid Sprinz, Joshua Weinger, and Haiyuan Yu. The guys especially were very close, we climbed Mt. Katahdin together and played several basketball games. I wish all of them good-luck in the future.

Erik Haghjoo, Robby Watson, Michael Seringhaus, Phillip Fontaine, Peter Korakas, Tara Gianoulis, Tom Royce, Sara Nichols, Jessica Babe, William Bradley, Andy Nickel, Amy Seila,

Rachel Dexter, Joe Bopp, Han Lee, Ben Lacar, Sean Ryder, Peggy Eatherton, Roger Cole, Craig Gibson, Paula Montero, Kevin Rice, Kelvin Tsou, Eric Poolman, Barry Twenter, Art Perlo, Dave Keller, Peter Adams, Olen Stephens, Natalie Tronson, Shannon Gibson, Kim Durniak, Kara Porwancher, Chris Incarvito

I also had many other friends and colleagues that made my time at Yale enjoyable: Peter Adams, Rachel Anderson, Frank Attenello, Jessica Babe, Jeff Barrick, Steve Becker, Isabel Beerman, Andrea Berman, Scott Bidlingmaier, Joe Bopp, Scott Boyle, William Bradley, Jack Brenner, Dylan Burnette, Matt Calabrese, Alex Carroll, Brian Coburn, Roger Cole, Rob Criscuolo, Chris DeFeo, Rachel Dexter, Eric Diken, Kim Durniak, Peggy Eatherton, Edward Eng, Adam Fogel, Phillip Fontaine, Jen Frank, Kendra Frederick, Kyle Friend, John Gehman, Michael Gelfand, Tara Gianoulis, Craig Gibson, Shannon Gibson, Michelle Gill, Matt Glassman, Erik Haghjoo, Bill Hankey, Diane Hannermann, Shayne Harrel, Gillian Hooker, Kevin Huang, Naomi Huang, Chris Incarvito, Thusitha Jayasundera (TJ), Dave Keller, Jeff Knight, Peter Korakas, Mitchell Kundel, Ben Lacar, Rebecca Lackman, Hue Lam, Han Lee, George Lin, Steve Luzietti, Jeff Mankoff, Abby Maranda, Chris Mason, Erin Matthews, Mike McEvoy, Ann Miller, Paula Montero, Steve Nafziger, Ali Nahvi, Matt Neujahr, Sara Nichols, Andy Nickel, Adrian Olivares, Shae Padrick, Eric Paulson, Art Perlo, Eric Poolman, Kara Porwancher, Tyler Radniecki, Kevin Rice, Jesse Rinehart, John Rinn, Tom Royce, Sean Ryder, Jeff Sabina, Aric Sanders, Amy Seila, Michael Seringhaus, Hitesh Sharma, Kelly Sheppard, Iam Simon, Mary Stahley, Brad Stanley, Olen Stephens, Sawako Sugimura, Miguel Talavera, Keith Tanis, Cara Tracewell, Natalie Tronson, Kelvin Tsou, Ben Turek, Barry Twenter, Jamie Vergis, Leven Wadley, Robby Watson, Stephanie West, Ken Wickiser, Jess Williamson, Shaun Wilson, Richard Wing, Lori Yang, and Crystal Zellefrow. I would also like to thank all my fellow basketball teammates and fantasy football players over the years.

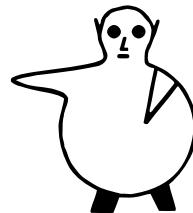
Closest friends

I would like to thank several close friends from Iowa: Nate Adam, Nate & Jodi Brockman, Nate “Brownie” Brown, Joe Bruner, Bobby Hoyer, Josh Kortbein, Travis Olson, Bryan Spring, and Brian “Freksho” Walsh. I would also like to thank the many teachers that helped me get here: Norman Anderson, Douglas Mupasiri, Robert Nelson, Mary Staniger, Kenneth Tesla, Michael Tringides, and Stephen Williams. And also all the people I met at CY-TAG.

My closest friends Corey Morcombe, Jim Schleicher, Jeff Valdez, Bon Koo, Kendrick Jones, Jean Kuan, Dino Gnanamgari, and Josh Young were always very supportive of me and helped me in times of crisis. I would especially like to thank Dino, Josh and Jim for letting me sleep (live) on their couch for the past two months while I finished my dissertation.

My family

Last, I would like to thank my family. My mom, Nancy Thorne flew out to be here on my defense has given me my modesty and love for math. My dad, Gary Voss trained me the basics of computers and gave me my inherent business sense. I would like to thank my grandparents, Donna and Bill Voss who have supported both financially and kept the rest of the family informed about my daily progressions. My grandma, Gertrude Thorne has always been very supportive. My aunt, Kristi, uncles, Steve and Craig, and cousin, Jen have looked upon my career with great support. The many siblings of my grandparents and their offspring, have always been involved with our family. And the many people that I have worked with and for during my first 22 years in Iowa. I would like to thank, my brother, Eric, whom I respect a lot for working full time trying to support his daughter, Keaton all the while trying to get a college degree. Finally, I would like to thank my wife, Heather Volkman, who has spent the last two months in San Diego alone. She has read this dissertation several times helping to fix my numerous typos and has also pushed me to finish this up as fast as I could.



AFDM

Table of Contents

Abstract	i
Title Page	iii
Copyright	iv
Quote	v
Acknowledgements	vi
Table of Contents	xii
List of Figures	xvii
List of Tables	xx
1 Introduction to the Dissertation	1
1.1 The Ribosome	1
1.1.1 Structure	2
1.1.2 The Ribosomal Exit Tunnel	4
1.1.3 Eukaryotic Ribosomes	4
1.2 Geometric Analysis of Biological Structures	5
1.3 References	7
2 Micro-volumes: Calculation of Standard Atomic Volumes for RNA and Comparison with Proteins	9
2.1 INTRODUCTION	9
2.2 FORMALISM AND RESULTS	13
2.2.1 Atomic Radius Calculations	13
2.2.2 Determination of Volumes	21
2.3 DISCUSSION	30
2.3.1 Methodological Effects on Calculations	30
2.3.2 Comparison to Proteins and DNA	37
2.4 CONCLUSIONS	43
2.5 REFERENCES	45
3 Macro-volumes: Ribosome Volume Assessment Using a Rolling Probe Sphere	49
3.1 INTRODUCTION	49
3.2 RESULTS	50
3.2.1 Formalism	50

3.2.2	Defining the Shell	52
3.2.3	Statistical Characteristics of Surface Roughness	58
3.2.4	Volume Distribution of the Ribosome	64
3.2.5	Characterizing the Solvent in the Ribosome	69
3.2.6	Analysis of Other Macromolecules	72
3.3	DISCUSSION	76
3.3.1	Universality of the 10 Å Surface	76
3.3.2	RNA Molecules and Inter-helix Packing	79
3.3.3	Proteins and Inter-domain Packing	79
3.3.4	Partial Specific Volumes	81
3.4	METHODS	81
3.4.1	Atomic Coordinates and VDW Radii	81
3.4.2	External Programs	82
3.4.3	Rolling Probe Method	83
3.5	REFERENCES	86
4	Characterization of the Ribosomal Exit Tunnel	91
4.1	INTRODUCTION	91
4.2	RESULTS	93
4.2.1	Defining the Exit Tunnel	93
4.2.2	Properties of the Canonical Exit Tunnel	97
4.3	DISCUSSION	105
4.3.1	Branching of the Exit Tunnel	105
4.3.2	Protein Folding in the Exit Tunnel	107
4.3.3	Permeability of the Large Ribosomal Subunit to Ions	110
4.3.4	Mechanism of Polyphenylalanine Translation	113
4.3.5	Thermal Fluctuations	115
4.4	METHODS	116
4.4.1	Atomic Coordinates and VDW Radii	116
4.4.2	The Rolling Probe Approach to Determining Volumes	117
4.4.3	Calculation of Linear Dimensions	119
4.4.4	Availability of Software	121
4.5	REFERENCES	122
5	Progress Towards the Crystallization of Ribosomes From Animals	126
5.1	INTRODUCTION	126
5.1.1	The Differences Between Prokaryotic and Eukaryotic Ribosomes	126
5.1.2	Ribosomal Crystallography and Structure	135
5.1.3	Concluding Remarks	138
5.2	RESULTS AND DISCUSSION	138
5.2.1	System Selection	138

5.2.2	Purification	145
5.2.3	Assessing Sample Quality	148
5.2.4	Crystallization	163
5.3	METHODS	173
5.3.1	Materials	173
5.3.2	Buffers	174
5.3.3	Cell Breakage	175
5.3.4	Ribosomes from Cell Lysis	179
5.3.5	Assessing Sample Quality	181
5.3.6	Hidden Breaks	181
5.3.7	Crystallization	181
5.4	REFERENCES	183
A	Rolling Probe Technique for Calculating Excluded Volumes	193
A.1	Analytical technique	193
A.2	Discrete technique	195
A.3	Advantages of the discrete method	199
A.4	Surface area calculation	199
A.5	References	202
B	Alternative Methods to Extract Interior Volumes	203
B.1	Space Overlapping	203
B.2	Active Contours and the Level Set Method	206
B.3	Alpha Shapes	208
B.4	Vector Stepping	208
B.5	References	213
C	Additional Ribosomal Exit Tunnels from Other Structures	214
C.1	Available structures	214
C.2	Volumes of the Exit Tunnels	214
C.3	Rotations	218
C.4	References	221
D	Data Files	222
D.1	Data Files for the Voronoi Programs	222
	atom-defs.dat	225
	atom-vols-prot.dat	226
	atom-vols-rna.dat	227
	bond-lengths.dat	228
	resi-vols.dat	230
	stdvol258.dat	231

stdvol-extended.dat	234
D.2 PDB Files for the Exit Tunnel	235
beta-turn.pdb	236
tungsten-11.pdb	237
E Perl and Shell Scripts	239
E.1 General Perl and Shell Scripts	239
fastnet.pl	240
filetools.pl	241
fixnum.pl	242
E.2 Perl Scripts for the Voronoi Calculations	243
bond-lengths.pl	244
find_min-stdev.pl	245
gen_atomvol.pl	248
gen_basesize.pl	251
gen_histogram.pl	253
gen_stdvol-classic.pl	255
gen_stdvol-extended.pl	256
rm_extreme-pts.pl	258
rm_symmetry.pl	260
run_voronoi.pl	261
superpak8.pl	262
E.3 Perl Scripts for the Solvent and Tunnel Calculations	266
ccp42png.pl	267
get_close-atoms.pl	268
get_near-tunnel.pl	270
get_tunnel-dist.pl	272
pdb2pts.pl	273
rotate-pdb.pl	274
sugar-pucker.pl	275
tunnel-worm.pl	276
xyzr2pts.pl	288
F The Voss Volume Voxelator Program	289
F.1 Library	289
Makefile	290
utils.h	291
utils-main.cpp	292
utils-output.cpp	303
pdb_to_xyzr.sh	309
atmtypenumbers.dat	310

F.2	Example Programs	313
	cavities.cpp	314
	channel.cpp	316
	fsv-calc.cpp	318
	solvent.cpp	320
	tunnel.cpp	322
	vdw.cpp	324
	volume.cpp	325
G	GNU Free Documentation License	326
G.1	APPLICABILITY AND DEFINITIONS	326
G.2	VERBATIM COPYING	327
G.3	COPYING IN QUANTITY	327
G.4	MODIFICATIONS	328
G.5	COMBINING DOCUMENTS	329
G.6	COLLECTIONS OF DOCUMENTS	329
G.7	AGGREGATION WITH INDEPENDENT WORKS	329
G.8	TRANSLATION	329
G.9	TERMINATION	329
G.10	FUTURE REVISIONS OF THIS LICENSE	330

List of Figures

1.1	Side view of the <i>E. coli</i> ribosome from 1976 and 2006	3
2.1	Two-dimensional Voronoi constructs and problems	12
2.2	Venn diagram of protein and RNA atom types	14
2.3	Determining non-bonded VDW radius for the unassigned <i>P4H0u</i> atoms	18
2.4	Distributions of atom type volumes	20
2.5	Determining non-bonded VDW radii for the unassigned big and small <i>O2H0</i> atoms	22
2.6	Plane position methods and vertex error	24
2.7	Distribution of RNA atoms within PDB set	27
2.8	Graphical display of different atomic packing measurements	33
2.9	Effects of crystal symmetry	34
2.10	Comparison of atomic volumes from RNA to protein	40
3.1	Richards' rolling sphere definitions	51
3.2	Convex hull of the ribosomal large subunit	54
3.3	Cavity volume as a function of probe radius	56
3.4	The 10 Å excluded surface of the large subunit compared to its 100 Å surface	57
3.5	Volume and surface area as a function of probe radius	59
3.6	Effective spheres as a function of probe radius	61
3.7	The two probe method to extract the solvent volume from macromolecules	71
3.8	Slices through the interior of the ribosome and selected proteins showing interior solvent volumes	73
3.9	Fractional solvent volume as a function of trimming radius	75
3.10	Histogram of the fractional solvent volume for proteins	78
3.11	Cavity volume from multi-domain proteins as a function of probe radius	80
4.1	The dependence of volume accessible from the exit tunnel on probe radius	95
4.2	Excluded volumes of the ribosomal exit tunnel at different probe radii	96
4.3	The effects of trimming back the shell surface on the exit tunnel volume and shape .	98
4.4	Stereo diagram of the polypeptide exit tunnel showing the positions of landmarks .	100
4.5	The width of the ribosomal polypeptide exit tunnel	101
4.6	Tightness of the exit tunnel around the peptidyl-transferase center	102
4.7	The three major branches of the exit tunnel	106

4.8	Stereo projections of space-filling models of molecules inside the polypeptide exit tunnel	109
4.9	The interior volumes of a chaperonin and the polypeptide exit tunnel compared	111
4.10	Permeability of ions from the exit tunnel	112
4.11	Calculation of linear dimensions for polyalanine chains	120
5.1	Low resolution profiles of the ribosomal subunits	128
5.2	The process of initiation of translation in prokaryotes	133
5.3	The process of initiation of translation in eukaryotes	134
5.4	Image of several whole rabbit pancreases	141
5.5	Pictures of brine shrimp, <i>Artemia</i> , cysts	144
5.6	FPLC profile for <i>Artemia</i> 80S ribosomes	149
5.7	Sucrose gradient profiles for <i>Artemia</i> ribosomes	150
5.8	Acrylamide gel of the <i>Artemia</i> ribosomal proteins	153
5.9	Agarose gel of the <i>Artemia</i> ribosomal RNA	155
5.10	Secondary and tertiary location of expansion segment number 19	157
5.11	28S rRNA sequences of protostomes with known “hidden break” locations aligned to the <i>Artemia</i> 28S rRNA	158
5.12	Location of the “hidden break” in predicted ES19 secondary structures	159
5.13	Sucrose gradient profiles for incubated and chilled ribosomal preparations	161
5.14	Electron micrographs of the <i>Artemia</i> ribosomes using negative stain	162
5.15	Images of crystallization results from HWI for 80S ribosomes	168
5.16	Images of crystallization results from HWI for 60S ribosomal subunits	169
5.17	Images of crystallization results of “the PEGs” buffers for 80S ribosomes	171
5.18	Lysis protocol of rabbit pancreas	176
5.19	Lysis protocol of <i>Artemia</i> cysts	178
5.20	Protocol from cell lysis to purified ribosomes	180
A.1	The van der Waals and solvent accessible contours of a section of the ribonuclease S molecule	194
A.2	Analytical rolling probe method	196
A.3	Discrete rolling probe method	197
A.4	Discrete rolling probe method with small grid spacings	198
A.5	Surface area calculation for discrete voxels	200
B.1	The space overlapping method	204
B.2	Example channels from the space overlapping method	205
B.3	Example channels from the space overlapping method	207
B.4	Defining cavities and channels using alpha shapes	209
B.5	Using alpha shapes to extract interior channel of gramicidin	210
B.6	Extracting the exit tunnel by vector stepping	212

C.1	Volume as a function of probe radius for the different exit tunnels	216
C.2	Images of the different exit tunnels	217
D.1	Distribution of atom type volumes for both plane positioning methods, part 1 . . .	223
D.2	Distribution of atom type volumes for both plane positioning methods, part 2 . . .	224

List of Tables

2.1	Atomic radii for the protein set <i>ProtOr</i> and the composite set <i>NucProt</i>	16
2.2	Summary of structure sets	28
2.3	Base volumes across several different structure sets	31
2.4	Atom volumes for each nucleotide	36
2.5	Summary of effect of other atoms on the packing calculations for the 50S ribosomal subunit	38
2.6	Summary of atom types in protein and RNA	41
3.1	Sphericity of some common shapes and ribosomal surfaces	65
3.2	Volumes of the large ribosomal subunit and its components	66
3.3	Partial specific volumes of the ribosomal components	70
3.4	Solvent properties for several different macromolecules	77
4.1	Volumes of the polypeptide exit tunnel	104
4.2	Stokes radii of selected ions	114
5.1	Molecular composition of ribosomes from a prokaryote and several eukaryotes . .	130
5.2	Progress of three-dimensional eukaryotic ribosome structures obtained by cryo-electron microscopy	137
5.3	Distribution of pancreatic ribonuclease A in vertebrates	139
5.4	Ribosome yield for different cell breaking techniques	146
5.5	Solubility of ribosomes in the presence of the precipitant, polyethylene glycol . .	164
5.6	Crystallization results from Hauptman-Woodward Institute (HWI)	167
5.7	Crystallization results from 80S ribosome sample and “the PEGs” buffers	170
C.1	All 50S subunit structures available and their characteristics	215
C.2	RMSD differences between the PDB files	220

Chapter 1

Introduction to the Dissertation

In this dissertation I use geometric methods to extract information from RNA structures including the largest RNA molecule of them all, the ribosome (described below). By using geometry, I was able to determine how the volume is distributed in the large ribosomal subunit as well as RNA in general and for the first time create a model of the ribosomal polypeptide exit tunnel. I also report on work done to purify eukaryotic ribosomes for crystallization.

The results of this research has been published in two articles: Voss & Gerstein (2005), which in expanded form corresponds to Chapter 2 and Voss *et al.* (2006), which has been expanded into Chapters 3 and 4.

1.1 The Ribosome

Ribosomes were first observed in tissue sections using an electron microscope where they appeared as dense particles or granules (Luria *et al.*, 1943; Palade, 1952, 1955). These granules were initially called “ribonucleoprotein particles of the microsome fraction,” until a more convenient word, “ribosome” was proposed by Roberts (1958) as the name for these particles. While it was known by 1941 that RNA plays an important role in protein synthesis (Caspersson *et al.*, 1941; Brachet, 1942), the discovery that the ribosome is the machine responsible for making proteins in a living cell occurred much later (Littlefield *et al.*, 1955). It is now understood that the ribosome works

by translating the sequences of messenger RNAs (mRNAs) into polypeptide chains that pass through an interior channel in the particle called the “exit tunnel” as they are made. Because ribosomes produce all of the protein in the cell, cell growth is closely coupled to ribosome availability, and thus ribosomes are highly abundant in all growing cells. In bacteria, for example, ribosomes can account for up to 50% of the dry cellular mass (Neidhardt & Magasanik, 1960; Zengel & Lindahl, 1994).

1.1.1 Structure

The structure of the ribosome has been understood in general terms for several decades (Figure 1.1A). In all organisms, it consists of two parts, a large subunit and a small subunit. The small subunit is associated with the decoding of mRNA sequences, and the large subunit catalyzes peptide bond formation and includes the tunnel through which nascent peptide chains pass as they are made. In bacteria, the ribosome is a ribonucleoprotein composed of 3 ribosomal RNA molecules (rRNAs): 23S rRNA (~2900 nucleotides), 16S rRNA (~1540 nucleotides), and 5S rRNA (~120 nucleotides). The three rRNA molecules comprise about two-thirds of the entire particle by mass. The remaining mass is distributed among ~55 ribosomal proteins that range in size from 4–30 kDa with the exception of the atypically large 68 kDa S1 protein.

The first atomic resolution structures of ribosomes and ribosomal subunits appeared only recently. In 2000, the atomic structure of the 50S large ribosomal subunit (Figure 1.1B, right) from the archaea, *Haloarcula marismortui* was determined by Ban *et al.* (2000). Soon after, the structure of the 30S subunit (Figure 1.1B, left) from *Thermus thermophilus* was determined by Schluenzen *et al.* (2000) and Wimberly *et al.* (2000), independently. These structures initiated a new era in ribosomal studies in which investigations depend on knowledge of atomic coordinates. In Chapters 2 and 3, I use the atomic structure to examine the how the atoms within the ribosome are

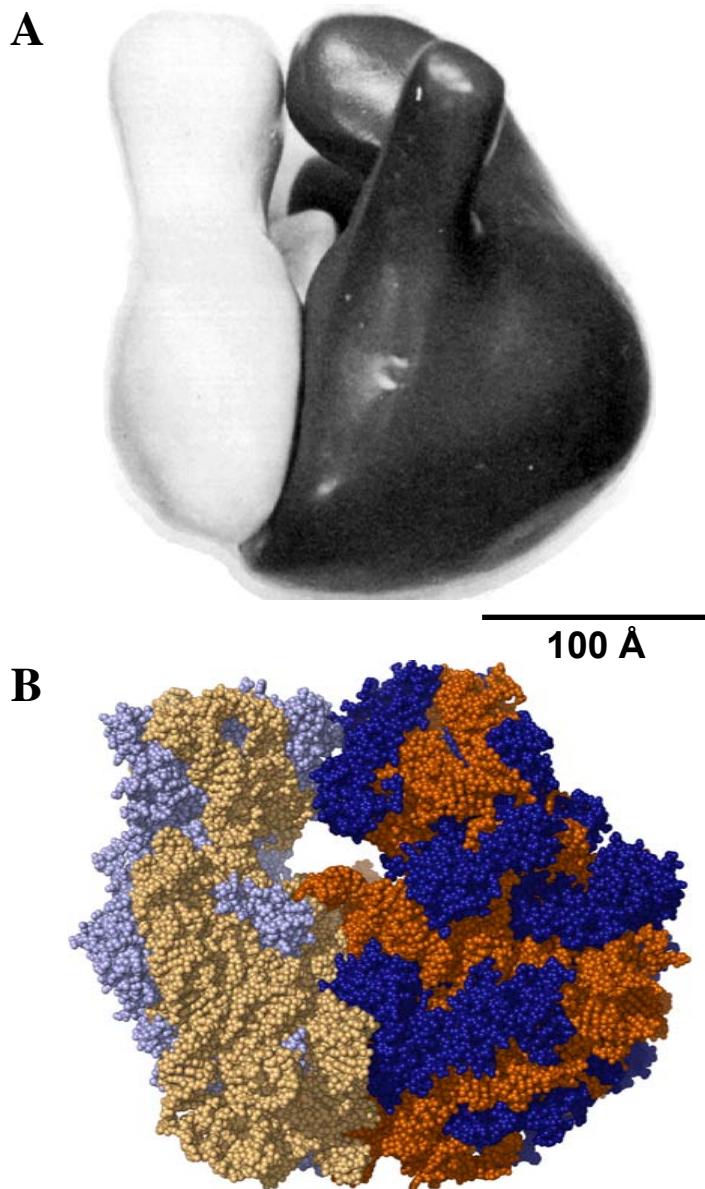


Figure 1.1: Side view of the *E. coli* ribosome from 1976 and 2006. (A) The averaged electron microscope structure of the *E. coli* 70S ribosome determined by Lake (1976). The small subunit is white (left) and the large subunit is black (right). (B) The 70S ribosome X-ray structure from *E. coli*. The small subunit (left) is lightly colored in blue and orange and the large subunit (right) has darker colors. For both subunits proteins are colored in blue and RNA in orange. Image created from PDB ids, 1aw4 and 1avy (Schuwirth *et al.*, 2005) using PyMOL (DeLano, 2002).

packed together. Chapter 2 addresses the packing of individual atoms, while Chapter 3 addresses the packing of the whole ribosomal particle.

1.1.2 The Ribosomal Exit Tunnel

An important structural feature of the ribosome is the large channel through the center of its large subunit called the exit tunnel through which proteins pass as they are synthesized. The possibility that an exit tunnel might exist was recognized over two decades ago by Bernabeu & Lake (1982) and its existence was definitively proven by cryo-electron microscopy by Frank *et al.* (1995). Despite, being heavily studied biochemically little is known about its shape and size and many disputes about its properties have developed. In Chapter 4, I create a model for the exit tunnel and address many of the properties that have mistakenly been ascribed to it, namely that proteins can fold inside it, that it has branches, and that it provides a watertight seal when bound to the translocon in the membrane.

1.1.3 Eukaryotic Ribosomes

The ribosomes of prokaryotes and eukaryotes are generally quite similar, both have two subunits and share the same active site architecture, but their structural differences play an important role in their function. For example, they show differences in antibiotic resistance, which is exploited by doctors to cure bacterial infections. The eukaryotic ribosome (3.9–4.5 MDa) is much larger than its prokaryotic counterpart (2.6 MDa). Prokaryotic ribosomes have a sedimentation coefficient of 70S, each consisting of a small 30S and a large 50S subunit, whereas eukaryotes have 80S ribosomes, each consisting of a small 40S and a large 60S subunit (Tissieres & Watson, 1958). The “additional mass” of the eukaryotic ribosome results from having longer RNA chains, one additional rRNA molecule called the 5.8S rRNA and about 25 more proteins, which shift the mass ratio of RNA to protein from the two-thirds found in bacteria to about three-fifths. In Chapter 5, I discuss the work I have done on ribosomes from higher eukaryotes, including those from rabbit and from brine shrimp, to produce crystals from them.

1.2 Geometric Analysis of Biological Structures

Geometric calculations done on protein structures have led to important insights. Bondi (1964) was first to characterize the size of atoms in protein by assigning them van der Waals (VDW) radii. Later, Lee & Richards (1971) carried out the first systematic geometric analysis of a protein crystal structure. The geometric analysis of proteins developed by Frederic Richards and his colleagues described below largely applies to RNA concepts.

In 1971, Lee & Richards introduced the concept of “accessible surface area” which is the surface area of the protein determined by assigning to atom a radius equal to its VDW radius plus the radius of a water molecule, and computing the intersection of these surfaces. By subtracting the accessible surface area of a folded protein from that of the same polypeptide as an extended protein chain, Lee & Richards could calculate the “buried surface area” of a protein, which is an important determinant of its folding energy. It was found that protein interiors in general contain very few holes or “cavities” big enough to accommodate a water molecule implying that the hydrophobic cores of proteins are densely packed. In Chapter 3, I discuss accessible surface calculations done on the ribosome to extract information about packing of materials in the interior of the ribosome. Unlike proteins, the ribosome does not have a hydrophobic core, because it is made primarily of charged RNA molecules.

Before the packing of atoms in a structure can be understood, precise values are needed for the volumes occupied by its individual atoms. In 1967, Bernal & Finney successfully used Voronoi polyhedra to obtain volumes for atoms in small organic molecules. The Voronoi tessellation is a way of partitioning space amongst a collection of points that completely fills all space and has no overlaps and no gaps (Voronoi, 1908). In 1974, Richards applied the Voronoi technique to protein crystal structures to determine their packing density, which he defined as the VDW volume divided by its Voronoi or occupancy volume for each atom. Richards (1974) quickly realized that not all atoms in a protein are the same size and that the dividing planes should therefore not be positioned

equidistant between large and small atoms as the Voronoi construction requires. So, Richards (1974) developed a second method for positioning the planes, which he called “method B.” In Chapter 2, I use this approach to obtain volumes for the atoms in many different RNA molecules including the ribosome, and provide average atomic volumes for them in the same manner as Richards (1974) first did for proteins.

In 1977, Richards introduced an additional concept namely that of the molecular surface or “excluded surface,” which he determined by rolling a spherical probe over the surface of a protein in the computer. This algorithm was initially done in two-dimensions using a grid, and an analytical algorithm for extending it into three-dimensions was independently later solved by Connolly (1983) and Richmond (1984). Richards (1977) used his molecular surfaces to calculate quantities related to the overall shape of molecules. I explore these same quantities for the ribosome in Chapter 3 to address its packing and shape. In addition, Richards’ rolling probe algorithm was the technique used in Chapters 3 and 4 for characterizing the interior solvent and exit tunnel of the ribosome.

1.3 References

- N. Ban, P. Nissen, J. Hansen, P. B. Moore, and T. A. Steitz (2000). “The complete atomic structure of the large ribosomal subunit at 2.4 Å resolution”. *Science*, **289** (5481): pp. 905–20.
- C. Bernabeu and J. A. Lake (1982). “Nascent polypeptide chains emerge from the exit domain of the large ribosomal subunit: Immune mapping of the nascent chain”. *PNAS*, **79** (10): pp. 3111–5.
- J. D. Bernal and J. L. Finney (1967). “Random close-packed hard-sphere model II. geometry of random packing of hard spheres.” *Disc Faraday Soc*, **43**: pp. 62–69.
- A. Bondi (1964). “Van der Waals volumes and radii”. *J Phys Chem*, **68**: pp. 441–451.
- J. Brachet (1942). “La localization des acides pentosenucleiques dans les tissus animaux et les oeufs d’Amphibiens en voie de développement.” *Arch Biol*, **53**: pp. 207–257.
- T. Caspersson, C. Nystrom, and L. Santesson (1941). “Zytoplasmic nukleotides in tumour cells.” *Naturwissenschaften*, **29**: pp. 29–30.
- M. L. Connolly (1983). “Analytical molecular surface calculation”. *J Appl Cryst*, **16** (5): pp. 548–558. <http://dx.doi.org/10.1107/S0021889883010985>.
- W. L. DeLano (2002). “The PyMOL molecular graphics system”. URL <http://www.pymol.org>.
- J. Frank, J. Zhu, P. Penczek, Y. Li, S. Srivastava, A. Verschoor, M. Radermacher, R. Grassucci, R. K. Lata, and R. K. Agrawal (1995). “A model of protein synthesis based on cryo-electron microscopy of the *E. coli* ribosome”. *Nature*, **376** (6539): pp. 441–4.
- J. A. Lake (1976). “Ribosome structure determined by electron microscopy of *Escherichia coli* small subunits, large subunits and monomeric ribosomes”. *J Mol Biol*, **105** (1): pp. 131–139.
- B. Lee and F. M. Richards (1971). “The interpretation of protein structures: Estimation of static accessibility”. *J Mol Biol*, **55** (3): pp. 379–400. [http://dx.doi.org/10.1016/0022-2836\(71\)90324-X](http://dx.doi.org/10.1016/0022-2836(71)90324-X).
- J. W. Littlefield, E. B. Keller, J. Gross, and P. C. Zamecnik (1955). “Studies on cytoplasmic ribonucleoprotein particles from the liver of the rat”. *J Biol Chem*, **217** (1): pp. 111–123.
- S. E. Luria, M. Delbrück, and T. F. Anderson (1943). “Electron microscope studies of bacterial viruses”. *J Bacteriol*, **46** (1): pp. 57–77.
- F. C. Neidhardt and B. Magasanik (1960). “Studies on the role of ribonucleic acid in the growth of bacteria”. *Biochim Biophys Acta*, **42**: pp. 99–116.
- G. E. Palade (1952). “A study of fixation for electron microscopy”. *J Exp Med*, **95** (3): pp. 285–298.
- G. E. Palade (1955). “A small particulate component of the cytoplasm”. *J Biophys Biochem Cytol*, **1** (1): pp. 59–68.
- F. M. Richards (1974). “The interpretation of protein structures: Total volume, group volume distributions and packing density”. *J Mol Biol*, **82** (1): pp. 1–14. [http://dx.doi.org/10.1016/0022-2836\(74\)90570-1](http://dx.doi.org/10.1016/0022-2836(74)90570-1).

- F. M. Richards (1977). “Areas, volumes, packing and protein structure”. *Annu Rev Biophys Bioeng*, **6**: pp. 151–76. <http://dx.doi.org/10.1146/annurev.bb.06.060177.001055>.
- T. J. Richmond (1984). “Solvent accessible surface area and excluded volume in proteins. Analytical equations for overlapping spheres and implications for the hydrophobic effect”. *J Mol Biol*, **178** (1): pp. 63–89.
- R. B. Roberts (1958). “Introduction”. In R. B. Roberts, ed., “Microsomal Particles and Protein Synthesis.”, pp. vii–viii. Pergamon Press, New York.
- F. Schluenzen, A. Tocilj, R. Zarivach, J. Harms, M. Gluehmann, D. Janell, A. Bashan, H. Bartels, I. Agmon, F. Franceschi, and A. Yonath (2000). “Structure of functionally activated small ribosomal subunit at 3.3 Å resolution”. *Cell*, **102** (5): pp. 615–623.
- B. S. Schuwirth, M. A. Borovinskaya, C. W. Hau, W. Zhang, A. Vila-Sanjurjo, J. M. Holton, and J. H. D. Cate (2005). “Structures of the bacterial ribosome at 3.5 Å resolution”. *Science*, **310** (5749): pp. 827–834.
- A. Tissieres and J. D. Watson (1958). “Ribonucleoprotein particles from *Escherichia coli*”. *Nature*, **182** (4638): pp. 778–780.
- G. F. Voronoi (1908). “Nouvelles applications des paramètres continus à la théorie de formes quadratiques.” *J Reine Angew Math*, **134**: pp. 198–287.
- N. R. Voss and M. Gerstein (2005). “Calculation of standard atomic volumes for RNA and comparison with proteins: RNA is packed more tightly”. *J Mol Biol*, **346** (2): pp. 477–492. <http://dx.doi.org/10.1016/j.jmb.2004.11.072>.
- N. R. Voss, M. Gerstein, T. A. Steitz, and P. B. Moore (2006). “The geometry of the ribosomal polypeptide exit tunnel”. *J Mol Biol*, **360** (4): pp. 893–906. <http://dx.doi.org/10.1016/j.jmb.2006.05.023>.
- B. T. Wimberly, D. E. Brodersen, W. M. J. Clemons, R. J. Morgan-Warren, A. P. Carter, C. Vonrhein, T. Hartsch, and V. Ramakrishnan (2000). “Structure of the 30S ribosomal subunit”. *Nature*, **407** (6802): pp. 327–339.
- J. M. Zengel and L. Lindahl (1994). “Diverse mechanisms for regulating ribosomal protein synthesis in *Escherichia coli*”. *Prog Nucl Acid Res Mol Biol*, **47**: pp. 331–370.

Chapter 2

Micro-volumes: Calculation of Standard Atomic Volumes for RNA and Comparison with Proteins

2.1 INTRODUCTION

Numerous methods have been developed to determine atomic radii and volumes in proteins (Bondi, 1964; Chothia, 1974; Finney, 1975; Harpaz *et al.*, 1994; Li & Nussinov, 1998; Liang *et al.*, 1998a,b; Richards, 1974; Tsai & Gerstein, 2002; Tsai *et al.*, 1999, 2001) and these methods have been applied to DNA (Nadassy *et al.*, 2001). Atomic radii and volumes are necessary for understanding protein structure and particularly for uncovering the relationship between packing and stability. Atomic radii and volumes are used to characterize many protein properties including: folding energies (Chothia, 1975), protein-protein interaction energies (Janin & Chothia, 1990), standard residue volumes (Harpaz *et al.*, 1994), internal core packing derivatives (Janin, 1979; Richards, 1985), packing at the water interface (Gerstein & Chothia, 1996; Gerstein *et al.*, 1995), protein cavities (Hubbard & Argos, 1995; Liang *et al.*, 1998a,b), crystal structure quality (Pontius *et al.*, 1996), dependence of protein volume on amino acid composition (Gerstein, 1998; Gerstein *et al.*, 1994), macromolecular motions (Gerstein & Krebs, 1998; Krebs & Gerstein, 2000), and enzyme-substrate interactions (David, 1988; Finney, 1978). Standard volumes and radii are also important in an

indirect sense for the prediction of side-chain packing (Dunbrack, 1999; Koehl & Delarue, 1997; Lee & Levitt, 1997).

Although a standard atomic volume set has been available for proteins for years (Richards, 1974) and a DNA volume set was recently produced (Nadassy *et al.*, 2001), no attempts have been made to obtain a standard atomic volume set for RNA molecules. This has been due primarily to a lack of crystal structures for RNA other than tRNAs and small oligonucleotides. This lack of structures reflected both the difficulty of producing most RNAs in milligram quantities and the widespread belief that RNAs are hard to crystallize (Doudna & Cate, 1997). However in the past 15–20 years, it has been gradually discovered that RNA molecules are about as easy to crystallize as proteins, and a large number of RNA crystal structures have been obtained including structures for ribosomes (Ban *et al.*, 2000; Wimberly *et al.*, 2000), RNase P (Krasilnikov *et al.*, 2003; Kazantsev *et al.*, 2005) and self-splicing introns (Cate *et al.*, 1996; Adams *et al.*, 2004). Now that there are many crystal structures available, the data needed to address RNA packing exists.

To calculate atomic volumes in RNA, I employed the Voronoi polyhedra method. Use of Voronoi-like diagrams can be traced back to Descartes in 1644, but Voronoi (1908) was first to popularize this approach. The Voronoi polyhedra method is a way of partitioning all space amongst a collection of points by assigning each point a specially constructed polyhedra such that the resulting tessellation completely fills space. In this instance the points are atom centers. Bernal & Finney (1967) were the first to apply this method to molecular systems, and Richards (1974) was the first to use it on proteins. The specific methods used in this work have been previously described in an earlier work with which I am associated (Tsai *et al.*, 2001). Figure 2.1 shows how Voronoi polyhedra are constructed. The Voronoi construction partitions space such that all points within a polyhedron are closer to the atom associated with that polyhedron than to any other. Using the atom typing information, as first described by Richards (1974), Voronoi planes are shifted from the original location, which is equidistant between atoms (Figure 2.1a) to the modified position

(Figure 2.1b) determined by the relative van der Waals (VDW) atomic radii of the two atoms separated by that plane, so that bigger atoms will take up more space in the Voronoi construct than smaller ones. Only atoms whose volumes are well defined (Figure 2.1c) and not those that are loosely packed (Figure 2.1d) or unpacked (Figure 2.1e) are included used for subsequent analysis (Tsai *et al.*, 2001). Unpacked and loosely packed atoms are usually surface atoms or atoms near cavities and therefore do not have enough neighbors to pack tightly. The Voronoi method provides a good estimate of the true volume occupied by an atom and thus leads to reliable, self-consistent values for comparing atomic volumes. Atoms are assigned VDW radii based on their atom type (see Results, page 13). The typing follows standard united atom conventions and chemical atom typing. A new technique is described in this study that uses crystal symmetry to help define the Voronoi volumes of surface atoms.

Using Voronoi polyhedra and the techniques alluded to above, standard volumes (and many other statistics) were obtained for all of the different kinds of atoms found in RNA, of which there are 49 in total (see Table 2.4, page 36 for complete list) and for all four nucleotides. The 49 different kinds of RNA atoms can be sorted into 18 atom types, each with its own volume and radius. The atomic volumes and nucleotide volumes obtained are the same for all kinds of RNAs (*e.g.* tRNA, small rRNA, or ribosomes). Further, atoms on the surface of RNA structures are no different from those in the interior when crystal contacts are taken into account. Also, the Voronoi approach identifies atoms in RNA that have incomplete and extended geometries (Figure 2.1d and e). Many of them are backbone atoms because the backbone has a low percentage of well-packed polyhedra. As expected, RNA atomic volumes are similar to the DNA atomic volumes (Nadassy *et al.*, 2001) and to nucleotides volumes determined *in vitro* (Lee & Chalikian, 2001). Inclusion of solvent and proteins has no effect on the volumes found for RNA atoms, but the number of well-packed RNA atoms in any structure decreases significantly when such supporting atoms are not taken into consideration. Finally, atom type volumes are slightly larger in proteins than in RNA, but the percentage of well-

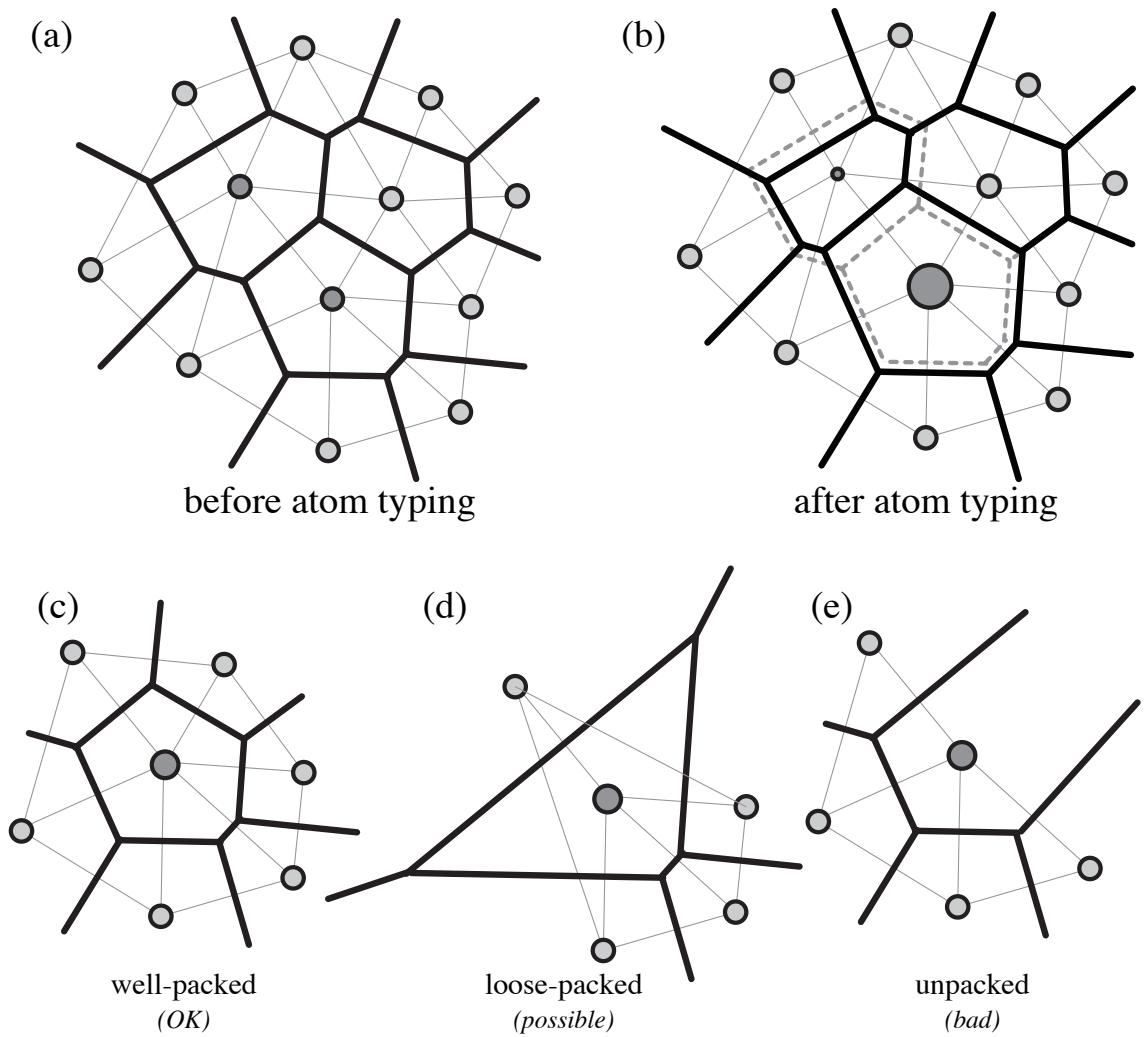


Figure 2.1: Two-dimensional Voronoi constructs and problems. The effect of atom typing on atomic volumes: (a) A two-dimensional example of the Voronoi construction. Planes are drawn equidistant between any two atoms. The planes are then intersected to get a volume. (b) For atoms of different sizes the planes are no longer placed equidistant between the atoms, but rather as a ratio function of the van der Waals radius of the atoms. So, large atoms are assigned a larger volume and small atoms are assigned a smaller volume. There are three major types of voronoi packing: (c) Well-packed: polyhedron is closed and surface area is less than a predetermined cutoff value. (d) Loose-packed: polyhedron is closed, but due to lack of neighbors the polyhedron has a large surface area above the cutoff value. (e) Unpacked: Voronoi polyhedron is open and no volume can be calculated. Only well-packed atoms are used to determine atomic volumes.

packed atoms is lower in RNA than it is in protein. These RNA atomic volumes along with the protein from the *ProtOr* set (Tsai *et al.*, 2001) form a new superset of nucleic acid and protein atomic volumes that others can use. I report this set of volumes, which I call the *NucProt* parameter set. The measured values are consistent across the many different RNA structures and packing environments.

2.2 FORMALISM AND RESULTS

2.2.1 Atomic Radius Calculations

Atom type nomenclature

The atoms in RNA molecules are given names of the form $XnHmS$, where X is the chemical symbol of the atom; n , the number of bonds it makes with its neighbors, which, in most cases, is equivalent to stating that the hybridization state is sp , sp^2 , or sp^3 ; Hm , the number, m of hydrogen atoms attached to the atom where the letter H does not change and only acts as a label for the number, m ; and S the subclassification for the atom type, which is one of the following symbols: b (big), s (small), t (tiny) or u (unique). When there are no subclasses for an atom type, u (unique) is the subclassification used. When the atom type needs to be divided into two separate sub-types the type with the larger volume is designated as b (big) and the smaller volume s (small). In one case (*C3H1*), the atom type requires the addition of a new classification from the two subclasses defined previously in proteins (Tsai *et al.*, 2001). Since the new subclass is smaller than both of the existing b (big) and s (small) subclasses, its subclass is designated as t (tiny). Figure 2.2 summarizes the 24 different atom types involved in RNA as well as protein and shows which types are common to both.

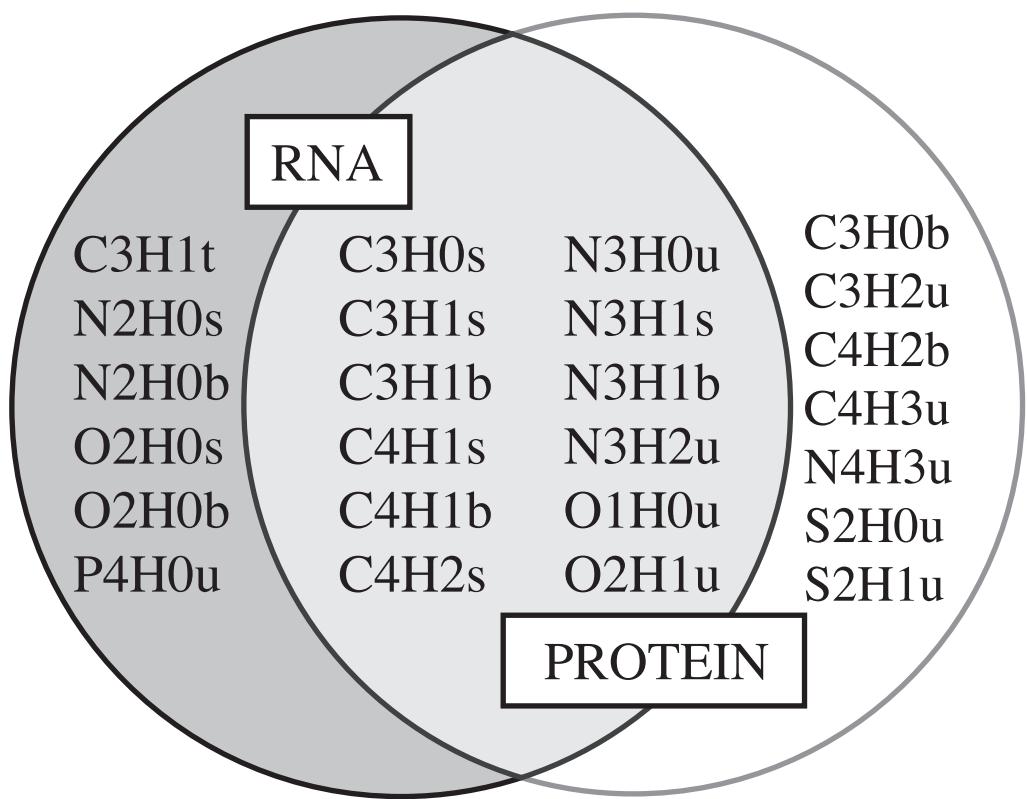


Figure 2.2: Venn diagram of protein and RNA atom types. Diagramed are all atom types found in RNA and protein. Types on the far left are only involved in RNA while types on the far right are only found in proteins. The central 12 types exist in both RNA and protein.

Importance of atom typing

As is described in more detail in Tsai *et al.* (2001), the distance between atoms and their intersecting planes used for Voronoi volume calculation depends on the VDW radius of the atom type (Figure 2.1b). Due to this dependence on atom radius, it is important to classify atoms accurately. A previous study on the effects of varying the number of atomic classifications came to the conclusion that the atom typing system described by the *XnHmS* nomenclature was the best balance between over and under fitting for accurate measurements of atomic volumes (Tsai *et al.*, 2001).

VDW radii taken from protein set

Several of the atom types found in RNA molecules are also found in proteins (Figure 2.2, center), and several papers have been published on the VDW radii of protein atoms (Harpaz *et al.*, 1994; Tsai *et al.*, 1999). For these overlapping types, the non-bonded VDW radii assigned RNA atom types were simply transferred from the corresponding protein atom types using data provided by *ProtOr* (Tsai *et al.*, 1999; Table 2.1). Whenever there is a small and big subclassification for a protein atom type (*e.g.* *N3H1* breaks into *N3H1s* and *N3H1b*), RNA atoms are typed by comparing their volume to those of protein atom types. For example, the guanine *N1* atom, of type *N3H1*, is more similar in volume to *N3H1s* than to *N3H1b* in proteins, so it is classified as *N3H1s*. Despite the vast differences in chemical structure, RNA molecules contained only three atom types not found in proteins: *O2H0*, *N2H0* and *P4H0*. If the atom types do not have unimodal distributions, they will need to be divided into small and big atom types and more than three atom types would exist. Assignment of all the pre-existing RNA atoms to types was for the most part straightforward; the only complication came from assignment of the three remaining missing types, which are described below.

	<i>ProtOr</i>		<i>NucProt</i>		contains RNA atoms	<i>Change bonded</i>
	<i>non-bonded</i>	<i>bonded</i>	<i>non-bonded</i>	<i>bonded</i>		
<i>C3H0s</i>	1.61	0.77	1.61	0.72	YES	0.05
<i>C3H0b</i>	1.61	0.77	1.61	0.72	no	0.05
<i>C3H1t</i>	—	—	1.76	0.68	YES	—
<i>C3H1s</i>	1.76	0.77	1.76	0.68	YES	0.09
<i>C3H1b</i>	1.76	0.77	1.76	0.68	YES	0.09
<i>C3H2u</i>	1.76	0.77	1.76	0.68	no	0.09
<i>C4H1s</i>	1.88	0.77	1.88	0.78	YES	0.01
<i>C4H1b</i>	1.88	0.77	1.88	0.78	YES	0.01
<i>C4H2s</i>	1.88	0.77	1.88	0.76	YES	-0.01
<i>C4H2b</i>	1.88	0.77	1.88	0.76	no	-0.01
<i>C4H3u</i>	1.88	0.77	1.88	0.74	no	-0.03
<i>N2H0s</i>	—	—	1.64	0.64	YES	—
<i>N2H0b</i>	—	—	1.64	0.64	YES	—
<i>N3H0u</i>	1.64	0.70	1.64	0.69	YES	-0.01
<i>N3H1s</i>	1.64	0.70	1.64	0.67	YES	-0.03
<i>N3H1b</i>	1.64	0.70	1.64	0.67	no	-0.03
<i>N3H2u</i>	1.64	0.70	1.64	0.61	YES	-0.09
<i>N4H3u</i>	1.64	0.70	1.64	0.73	no	0.03
<i>O1H0u</i>	1.42	0.66	1.42	0.52	YES	-0.14
<i>O2H0s</i>	—	—	1.50	0.65	YES	—
<i>O2H0b</i>	—	—	1.62	0.65	YES	—
<i>O2H1u</i>	1.46	0.66	1.46	0.65	YES	-0.01
<i>S2H0u</i>	1.77	1.04	1.77	1.04	no	0.00
<i>S2H1u</i>	1.77	1.04	1.77	1.04	no	0.00
<i>P4H0u</i>	—	—	1.82	0.97	YES	—

Table 2.1: Atomic radii for the protein set *ProtOr* and the composite set *NucProt*. The table shows the bonded and non-bonded radii if for both *ProtOr* and *NucProt* atomic parameter sets. Since *NucProt* is an extension of the *ProtOr* set, it includes protein atoms , so whether or not the atom type contains RNA atoms is indicated. The “change bonded” refers to the difference between the bonded radii for the *ProtOr* set and the adjusted bonded radii from the *NucProt* set. All numbers have units of Ångströms.

New atom types for RNA

After all of the RNA atom types that exist in protein were assigned, it was necessary to determine the behavior of the other three new atoms types, $N2H0$, $O2H0$ and $P4H0$. $P4H0$ only applies to one kind of RNA atom because the sugar backbone is averaged over all nucleotides and is not subdivided for each individual base. Therefore, $P4H0$ has to be given the $P4H0u$ designation. The $P4H0u$ atom type produces a tight histogram (Figure 2.3b), confirming that it has a unimodal distribution. The $O2H0$ and $N2H0$ atom types apply to three kinds of atoms and six kinds of atoms in RNA, respectively, of which neither has a simple volume distribution.

The $O2H0$ atom type applies to 3', 4' and 5' sugar oxygen atoms. Individual volume calculations show $O4'$ is significantly smaller than both of its type-equivalents $O3'$ and $O5'$, and the histogram of the $O2H0$ atoms (Figure 2.4f) is bimodal. Hence, I divided the $O2H0$ types into two classes: a big class, $O2H0b$ ($O3'$ and $O5'$), and a small class, $O2H0s$ ($O4'$).

The $N2H0$ atom type is more complex. It comprises six different kinds of atoms: adenine $N1$, $N3$, and $N7$, guanine $N3$ and $N7$, and cytosine $N3$. From Figure 2.4d, it is clear that the $N3$ and $N7$ atoms from both purines have a single unimodal volume distribution, and while the cytosine $N3$ is significantly smaller than all of the other atoms of the $N2H0$ type, it is grouped with the adenine $N1$ because it is only slightly larger. After grouping the atoms in types, a good separation between the small and big $N2H0$ atom types is found (Figure 2.4e). Inspection of $N2H0$'s bimodal volume distribution reveals that each group has the same volume as $N3H1$, which is also bimodal (see Table 2.6 on page 41 for final values). So for this reason, the $N2H0s$ and $N2H0b$ types were given the same atomic radii as the $N3H1s$ and $N3H1b$ types, respectively (Table 2.1; detailed below).

Adjusting the bonded VDW radii

Because this new *NucProt* data set is to encompass all the atoms found in RNA and protein, an investigation into bonded radii was undertaken to adjust the values used so that they would be

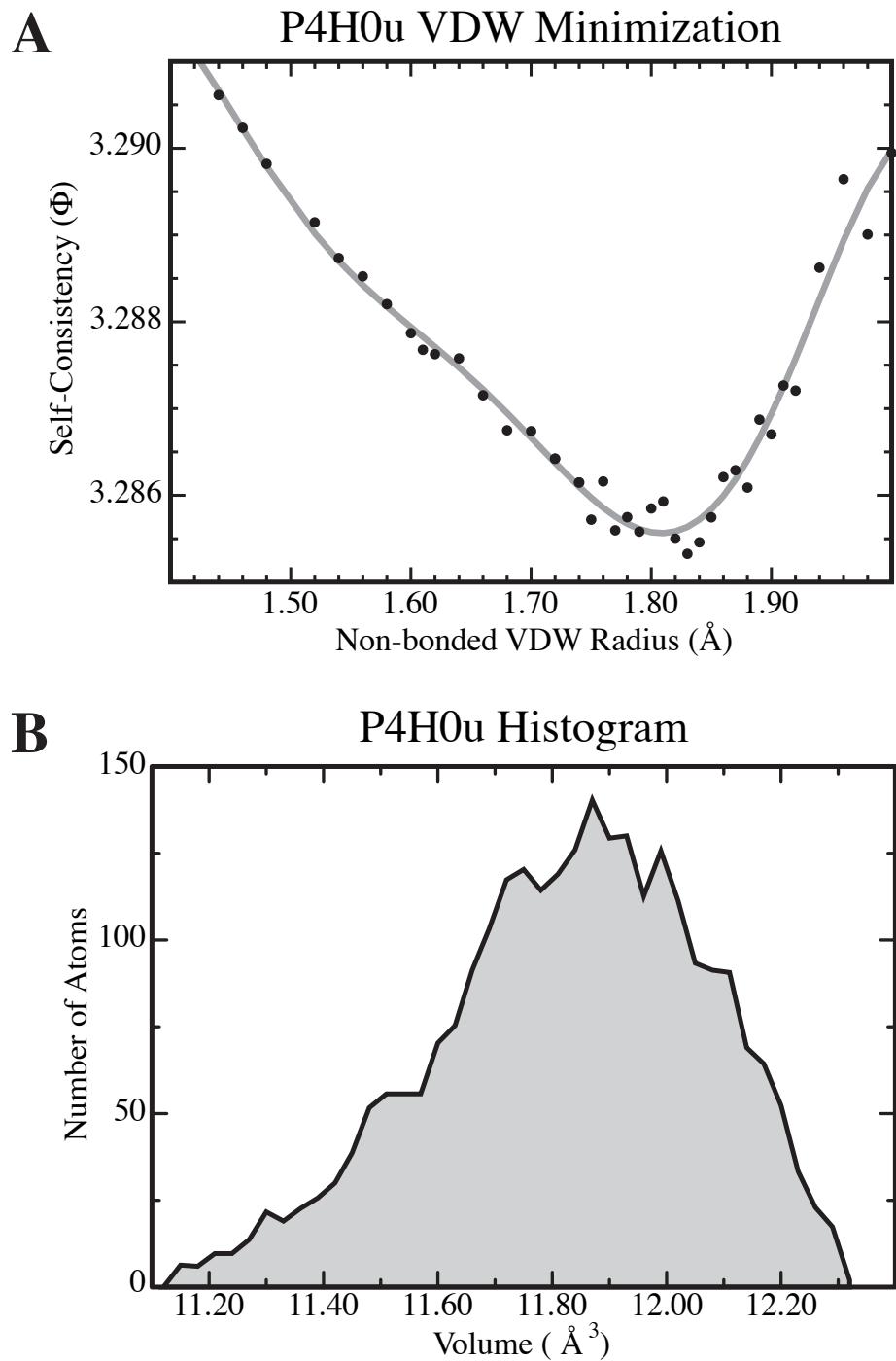


Figure 2.3: Determining non-bonded VDW radius for the unassigned *P4H0u* atoms. (a) The normalized standard deviation of the atomic volumes for all atoms is plotted versus the assigned non-bonded VDW radius for the *P4H0* atoms. The minimum was found to be 1.82 Å. (b) Histogram of the atomic volumes for the *P4H0u* atom using a non-bonded radius of 1.82 Å from the final *NucProt* data set.

optimal for both protein and RNA. Using previously defined bond lengths from Engh & Huber (1991) for protein and Parkinson *et al.* (1996) for RNA, the bonded radii were varied for each atom type (merging any small and big subclasses into one type) to minimize the sum over all squared bond differences (the bond length minus the VDW radius of both atom types involved in a bond) of RNA and protein atoms together. These bonded radii that emerged are not significantly different from those previously published (Tsai *et al.*, 2001), but now account for both protein and RNA chemical bond lengths (Table 2.1). The *O1H0u* bonded radius changes the most (from 0.66 to 0.52 Å, Table 2.1) reflecting a smaller oxygen atom size due to its double-bonded character in RNA. Despite the large change of the oxygen, the average absolute change of all the 19 original types (Figure 2.2, protein) is small, only 0.041 Å (Table 2.1). These new adjusted bonded radii provide a more self-consistent volume data set for both RNA and protein.

Determining non-bonded VDW radius of new types

For the newly created atom types, *O2H0* and *P4H0*, (from above), there are no existing non-bonded VDW radii available. To determine the non-bonded radii systematically, I used the standard deviation as a measure of consistency, a lower standard deviation means the distribution of atomic volumes are more consistently close to the mean value. An atomic volume set with the lowest overall standard deviation would be the most self-consistent set. However, the sum of all standard deviations alone gives an unfair bias to large atoms. This is because atoms with larger volumes have larger standard deviations, so for each kind of RNA atom the standard deviation was divided by the mean. The final equation used to determine the self-consistency (Φ) of the set is:

$$\Phi = 100\% \times \sum_{atom\ kind,\ j} \frac{\sigma_j}{\mu_j} \quad (2.1)$$

where σ_j is the standard deviation and μ_j is the mean for each kind of RNA atom, j (49 in total, see Table 2.4, page 36). By iterating the summed percent standard deviation (Φ) over several

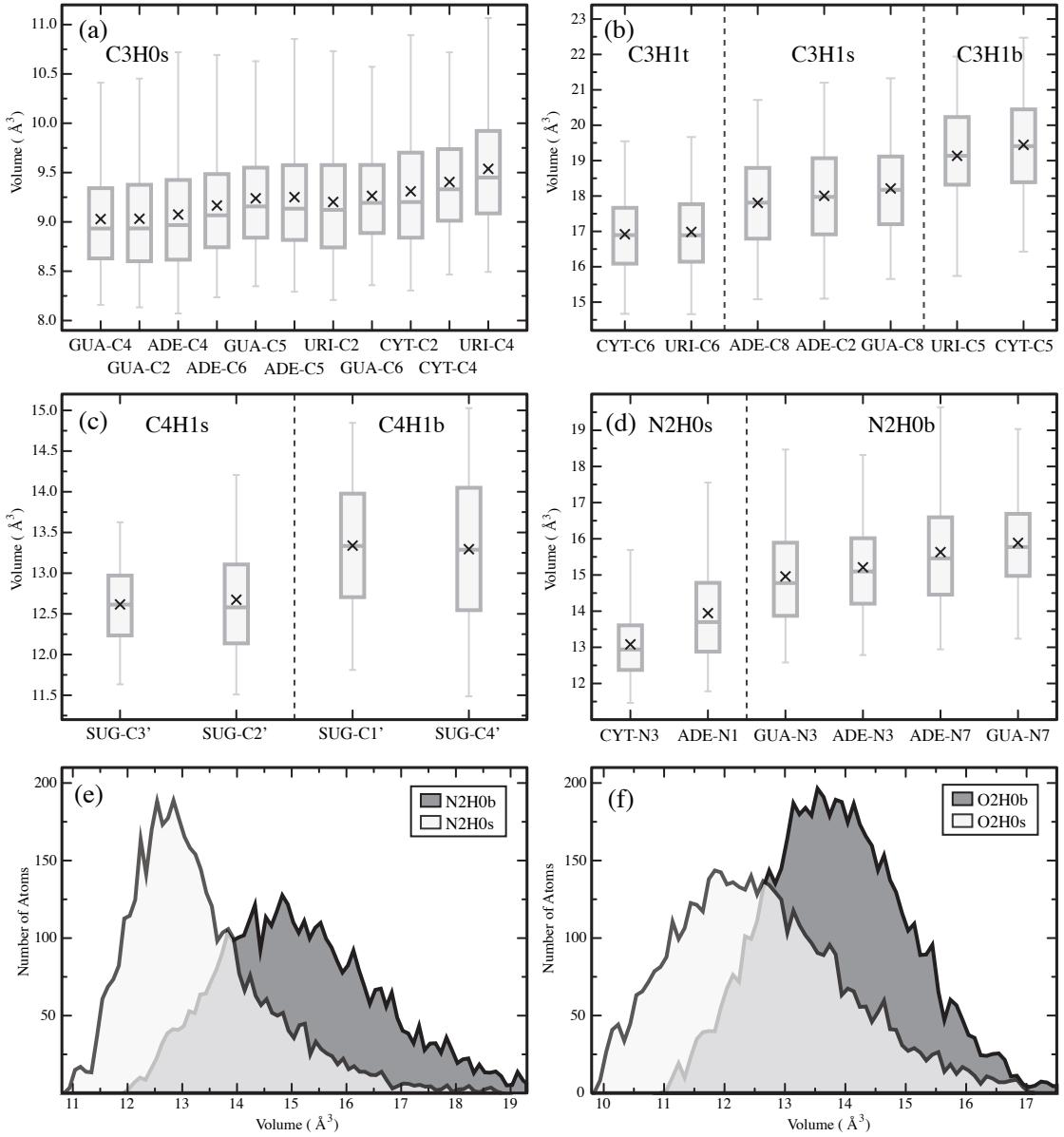


Figure 2.4: Distributions of atom type volumes. (a) Distribution of all atoms composing the $C3H0$ group with one distinct type, $C3H0s$. (b) Distribution of all atoms composing the $C3H1$ group divided into three distinct groups: tiny, small and big. (c) Distribution of all atoms composing the $C4H1$ group divided into two distinct groups: small and big. (d) Distribution of all atoms composing the $N2H0$ group divided into two distinct groups: small and big. (e) Volume distribution of the two $N2H0$ groups, small and big. (f) Volume distribution of the two types of $O2H0$ atoms, small and big. For parts (a) through (d), box plots are shown. The upper line represents the maximum value and the lower line represents the minimum values. For the box, the lower edge is the 25th percentile, the middle is the median value, and the top edge is the 75th percentile. The cross, \times is the mean value.

different atomic radii a minimum can be found (Figure 2.3a). The minimum is then designated as the non-bonded VDW radius for that atom. This method for calculating the missing non-bonded VDW radii results in the most self-consistent set of volumes.

As shown in Figure 2.3a, the standard deviation of the volume for phosphorus atom, *P4H0* volume gave a convex curve when its radius is varied. The curve is then fit to a tenth-degree polynomial (only to smooth out the noise without loss of generality) and the *P4H0u* radius is taken to be the minimum of the polynomial fit, which is 1.82 Å. This final value gave an extremely tight unimodal distribution (Figure 2.3b). Likewise, a two-dimensional optimization is employed for the *O2H0* types due to its bimodal distribution (Figure 2.4e), by simultaneously varying the radius of both subtypes. The global minimum of the percent standard deviation is determined to be 1.50 Å for the *O2H0s* and 1.62 Å for *O2H0b* (Figure 2.5).

2.2.2 Determination of Volumes

A brief description of Voronoi method

The atomic volumes are determined with the Voronoi method previously published (Gerstein *et al.*, 1995; Tsai & Gerstein, 2002; Tsai *et al.*, 1999, 2001). For every pair of atoms, a plane is constructed approximately equidistant from both of the atoms (in actuality the distance is adjusted by the VDW radii of each atom type) and orthogonal to the bond between the atoms (Figure 2.1). The planes are then intersected, leaving an enclosed polyhedron for each atom. While not every atom has a closed polyhedron, the majority of the RNA polyhedra are closed (89.1% in the final set) though not all are well-packed (38.4% in the final set).

Voronoi plane positioning method

Voronoi polyhedra were developed by Voronoi (1908) nearly a century ago. While the Voronoi construction is based on partitioning space amongst a collection of “equal” points, protein atoms

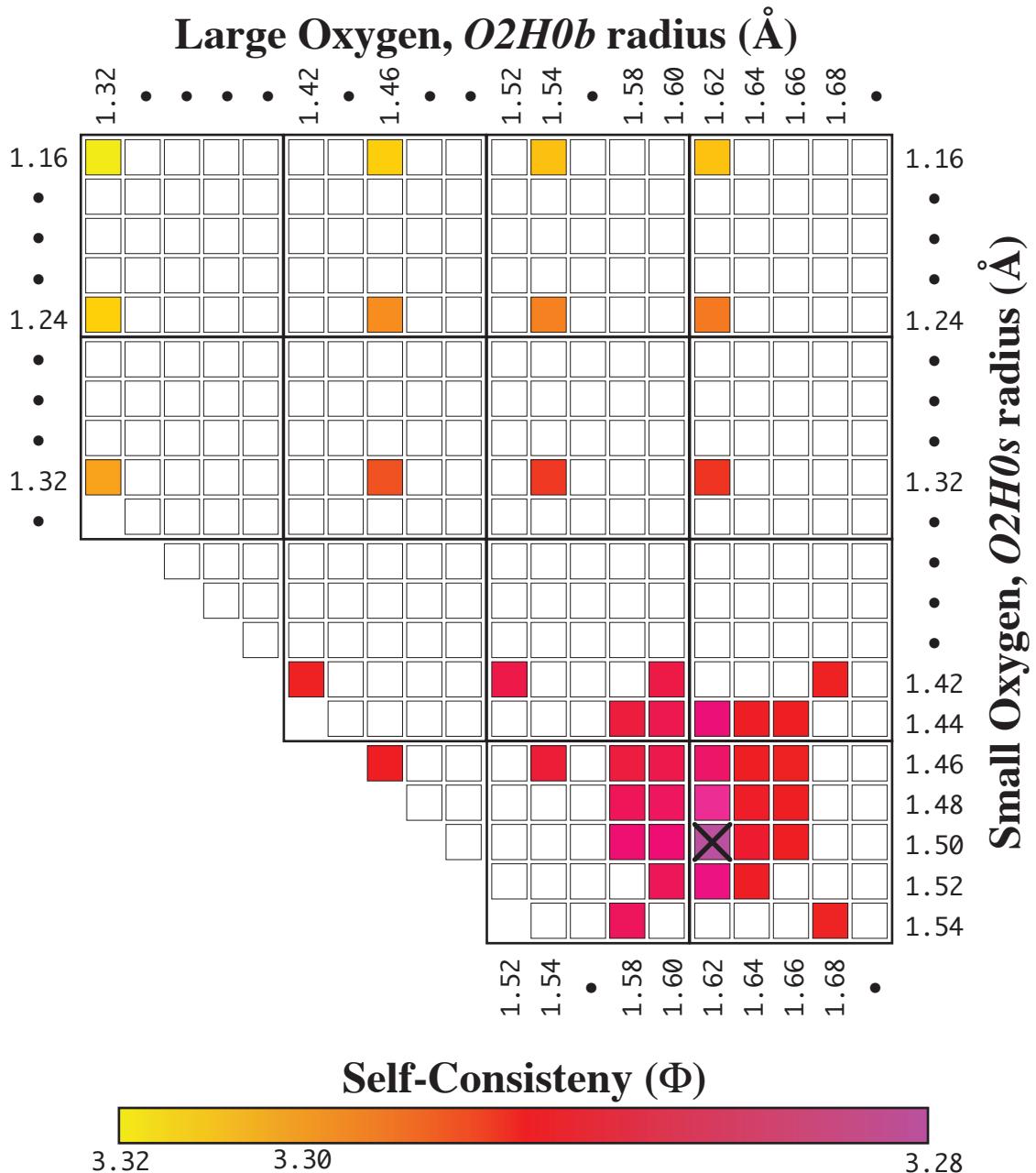


Figure 2.5: Determining non-bonded VDW radii for the unassigned big and small $O2H0$ atoms. The normalized standard deviation for the $O2H0s$ and $O2H0b$ atoms versus its VDW radius. The minimum is found to be 1.50 \AA and 1.62 \AA (marked with the black \times).

are not equal. Carbon is larger than nitrogen, which in turn is larger than oxygen. Richards (1974) was first to propose a solution to this problem, his so-called “method B” (Richards, 1974). The simplified method B (sometimes called the ratio method) divides the plane between the two atoms proportionately according to their covalent radii:

$$d = R + \frac{D - R - r}{2} \quad (2.2)$$

where d is the distance from the atom to the plane, R is the VDW radius of the atom, r is the VDW radius of the neighboring atom and D is the distance between the two atoms (Figure 2.1b). This method has been accepted for a long time, but it was later found that it has a peculiar flaw in it called “vertex error” (see Figure 2.6), which results when the planes created by neighboring atoms do not perfectly intersect at precise vertices. Vertex error becomes a major problem when working with atoms of dramatically different radii. For proteins, however, it has been reported that the volume unaccounted for only adds up to 0.2% of the total volume, primarily because atoms in biological macromolecules have a narrow range (between 1 and 2 Å) of VDW radii (Gerstein *et al.*, 1995). Nonetheless, a method of plane positioning was introduced that solves the problem of vertex error called the radical plane method (Gellatly & Finney, 1982). This new method uses a quadratic equation to divide up the space:

$$d = \frac{D^2 + R^2 - r^2}{2D} \quad (2.3)$$

Because it creates a perfect tessellation of polyhedra, the radical plane method is more precise geometrically than method B. These precise vertices are required for space-dividing constructions such as Delaunay triangulations (Tsai *et al.*, 1997) and alpha shapes (Liang *et al.*, 1998a, Appendix B). There is little difference between the two methods as far as atomic volumes are

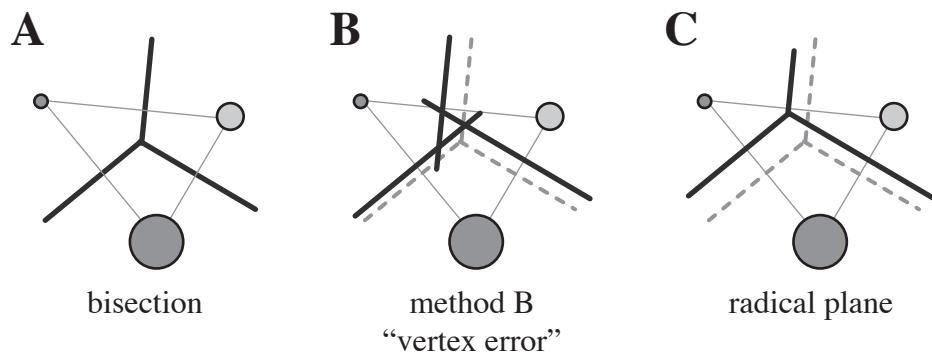


Figure 2.6: Plane position methods and vertex error. (A) Shows the simple “bisection method” where all atoms are treated equally and the planes are placed equidistant between the both atoms. (B) An overly exaggerated construction using the “method B” (Richards, 1974) showing “vertex error.” In fact actual vertex error is usually not visible with the naked eye. (C) The “radical plane method” (Gellatly & Finney, 1982) which always generates perfect vertices, but causes the planes to move much further than “method B.”

concerned. Even though method B suffers from vertex error, it is quite robust for protein calculations and more robust than the radical plane method (Tsai & Gerstein, 2002).

There are two major advantages to using method B. First, method B gives more self-consistent values. Volumes obtained by the radical plane method have higher standard deviations than volumes obtained by method B, suggesting the Voronoi planes are placed in a less consistent manner (Tsai & Gerstein, 2002). Second, method B has been thoroughly tested over the years, while the radical plane method is a more recent approach for macromolecules. The current standard volume set for protein atoms was determined using method B. For these reasons, the same methodology for proteins was used to obtain an RNA atomic volume set to facilitate comparisons. The raw data sets and the histograms for both methods are available in Appendix D (pages 223–224).

Assembling the structure set

Structures were obtained from the Nucleic Acids Database (NDB; Berman *et al.*, 1992) by searching for nucleic acids structures containing RNA, with strand lengths greater than 26 nucleotides—to avoid small, synthetic RNA molecules—and resolutions better than 5 Å. The cutoff value of 5 Å was chosen to include all four ribosomal subunit structures available at the time, including both low and high resolution structures (Ban *et al.*, 2000; Wimberly *et al.*, 2000; Schluenzen *et al.*, 2000; Harms *et al.*, 2001). A high resolution set consisting of structures of 2.0 Å or better was formed to remove any bias from the inclusion of structures with low resolution, but comparison of the data obtained from the high resolution with that extracted from the entire set made no difference. After determining the desired criteria, the search resulted in a raw structure set of 125 RNA structures. Most of the structures found were redundant (*e.g.* 50S ribosomal subunits soaked with various complexes or tRNA with and without synthetases) and some structures contained DNA base-paired with RNA. After removing the DNA hybrid structures and duplicates, taking care to use the most accurate and detailed structure within each redundant set, a final set was created consisting of

50 unique structures. For comparison purposes, the final sets were broken down into five smaller disjoint subsets: (i) *SRP* — RNA structures involved with the Signal Recognition Particle, (ii) *small-ribo* — small ribosomal RNA fragments, such the 5S rRNA structures, (iii) *tRNA* — transfer RNA, with and without synthetases, (iv) *small-RNA* — the other remaining small RNA molecules including ribozymes and self-splicing introns, and (v) *ribosomes* — complete ribosomal subunits (Table 2.2). These structure sets are unfortunately heavily weighted towards ribosomal data (69% of all atoms), because of the immense size of the ribosome, despite their constituting only 4 of the 50 structures in the set (Figure 2.7, Table 2.2). This effect will be addressed later.

Generation of final volume set

Surface atoms (as well as loosely packed interior atoms) sometimes lack closed polyhedra or have extended polyhedra and give rise to indeterminate or inflated volumes, respectively (Figure 2.1, page 12). These two kinds of Voronoi polyhedra need to be removed from the set of atom volumes considered in order to obtain a self-consistent data set. Loosely packed atoms are encountered when the Voronoi shell is heavily extended (Figure 2.1d) and are distinguished from well-packed atoms by the surface area of their Voronoi polyhedra. Atoms above a certain surface area cutoff were characterized as “possible,” meaning that they have a volume, but their surface area is so large that their volume is likely to be irrelevant. Loosely packed atoms were not used in the final *NucProt* data set due to their indefinite character.

If an atom has insufficient neighbors its Voronoi shell will be open-ended and it will have an indeterminate volume (Figure 2.1e). These unclosed polyhedra are easy to identify for they have no volume and are designated as “bad” by the software (Gerstein *et al.*, 1995; Tsai *et al.*, 2001). Consequently, only atoms with closed polyhedra and small surface areas are labeled as “OK” atoms (Figure 2.1c). It should also be noted that all atoms (protein, RNA, ions, water, organic molecules and also modified nucleotides and amino acids) within the PDB file are taken into account for Voronoi plane positioning. There are only 29 pseudo-uridines within my PDB set, and given that at

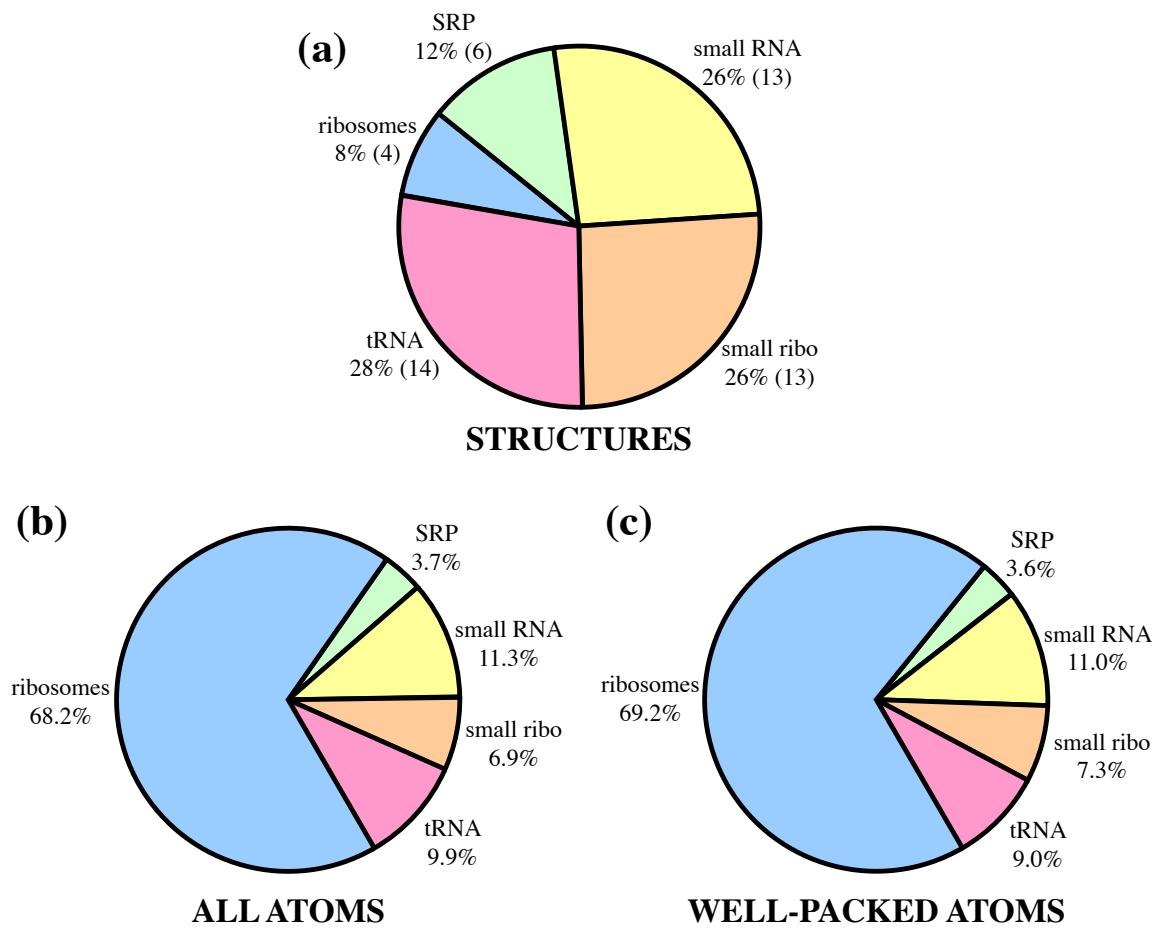


Figure 2.7: Distribution of RNA atoms within PDB set. Pie charts showing how the structure set breaks down into the five major categories: (a) Number of structures for each subset (b) Number of RNA atoms for each subset (c) Number of sufficiently packed or “OK” RNA atoms for each subset.

Set name	Number of PDB files	Number of RNA atoms	%“OK” no symm	%“OK” with symm	% of total RNA atoms	% of total “OK” atoms
<i>Disjoint subsets</i>						
SRP	6	10,137	35.0	37.9	3.7	3.6
Small Ribo	13	19,234	36.6	40.3	6.9	7.3
tRNA	14	27,379	33.2	34.8	9.9	9.0
Small RNA	13	31,438	33.7	37.2	11.3	11.0
Ribosomes	4	188,911	38.9	39.0	68.2	69.2
ALL	50	277,099	37.4	38.4	100.0	100.0
<i>Additional sets</i>						
Hi-Res	9	11,281	49.4	58.7	4.1	6.2
RNA Only	19	33,782	33.7	38.6	12.2	12.2

Table 2.2: Summary of structure sets.

most half the atoms in RNA are well-packed, this is not enough information to provide an average.

Other modified bases volumes are not reported here for the same reason.

Despite applying the standards just described for generating final volume sets, extreme volumes still existed for each kind of RNA atom. To further protect against the effect of these outliers, extreme volumes were removed from the ends of the distributions for each kind of RNA atom so that the average range for each volume distribution dropped in half. The range was dropped by 50% because that percentage provided the best balance between data loss and reduction of range. The range dropping method was effective because it required the removal of only 1.25% of the data from the set. In other words, the central 97.5% of the data had half the range of the complete set of data. The extreme volumes in these distributions that were removed represent both over-packed atoms produced by structure determination error or loosely packed atoms missed by the surface area cutoff criterion for one reason or another. After reducing the range to remove extreme values, only well-packed atoms are left in final volume calculations.

Effect of surface molecules

The treatment of surface atoms plays an important role in calculating Voronoi volumes because Voronoi volumes rely on neighboring atoms to create polyhedra surrounding each atom, and surface atoms generally lack neighbors. To increase the number of neighboring atoms for surface atoms, the effects of crystal symmetry, bound proteins, and solvent atoms on structure were examined. All three factors had little effect on the final volume distributions obtained, but all made a significant contribution to the number of observations for each atom (Table 2.5, page 38). In the final *NucProt* data set, atomic volume information from both high and low resolution structures, both protein-containing and protein-free RNA structures, and the crystal symmetry generated structures are all pooled into one.

Final volumes

Using the of the techniques just described, final volumes were calculated. Since the Voronoi method yields distributions of volumes, *i.e.* the probability of that a given atom type will have a particular volume, both histograms and mean values are provided. Sample distributions are shown in Figure 2.4 and the mean values for the volumes are in Table 2.3. The final volumes for the RNA bases are $146.0 (\pm 4.0) \text{ \AA}^3$ for guanine, $139.3 (\pm 4.0) \text{ \AA}^3$ for adenine, $115.2 (\pm 3.5) \text{ \AA}^3$ for cytosine, and $111.0 (\pm 3.8) \text{ \AA}^3$ for uracil (Table 2.3). All four RNA sugar backbones are approximately the same size and so only one value of $176.2 (\pm 5.2) \text{ \AA}^3$ is reported (Table 2.3, bottom). The nucleotide volumes are $323.0 (\pm 6.5) \text{ \AA}^3$ for guanosine, $315.4 (\pm 6.6) \text{ \AA}^3$ for adenosine, $290.7 (\pm 6.4) \text{ \AA}^3$ for cytosine, and $291.3 (\pm 6.3) \text{ \AA}^3$ for uridine.

2.3 DISCUSSION

2.3.1 Methodological Effects on Calculations

RNA backbone and base packing

From the standard deviations and packing percentages, areas within RNA structure that are not well-packed or are less defined can be located. In the *NucProt* data set, only 26.8% of the sugar-phosphate backbone atoms are packed sufficiently to make a volume measurement possible, which is 20.2% less than the worst base, uracil (at 47.0% well-packed). Further, several of these backbone atoms have low percentages of well-packed Voronoi polyhedra (Figure 2.8b) and high standard deviations (Figure 2.8d). On the other hand, atoms located in the bases are very well-packed. This result suggests that atoms located in the bases “benefit” from (1) the ring structure of purines and pyrimidines providing intrinsic tight-packing neighbors, (2) Watson-Crick base-pairing interactions and (3) base-stacking interactions common in RNA structure which further pack the ring. Supporting this claim, the atoms located in the major groove edge of the RNA

Base Volume (\AA^3)	Standard Disjoint Sets					Additional Sets	
	SRP	ribosome	small ribo	small RNA	tRNA	hi-res	allrna
GUA	141.3	146.4	144.0	144.7	142.7	144.2	144.0
ADE	134.1	139.7	136.4	137.9	136.2	138.7	137.3
CYT	110.2	115.6	111.8	113.4	111.6	113.2	113.1
URI	102.2	111.3	107.4	106.1	107.9	106.6	106.9
<i>Backbone Volume(\AA^3)</i>							
SUG	172.9	176.0	175.8	175.2	174.5	176.8	175.7

Base Volume (\AA^3)	ALL SETS		Other Published Sets			Z-DNA^c
	complete	organic ^a	A-DNA ^c	B-DNA ^c		
GUA	145.9	157.7	145.4	143.8	149.4	
ADE	139.2	148.8	138.3	136.1	92.4	
CYT	115.0	122.1	113.9	113.2	115.2	
URI	110.8	119.2	132.9 ^d	82.7 ^d	132.6 ^d	
<i>Backbone Volume(\AA^3)</i>						
SUG	176.1	133.6 ^b	181.8	174.8	166.6	

Table 2.3: Base volumes across several different structure sets.

^aValues converted from Lee & Chalikian (2001). Volumes require conversion of units from cm per mole to \AA^3 per residue. ^bThis value is from a nucleoside, not a nucleotide, and lacks a phosphate group with a volume of approximately 43 \AA^3 (depending on oxidation state). Calculated sugar volume is averaged over three base volumes subtracted from the nucleoside volumes. ^cValues taken from Nadassy *et al.* (2001). ^dThymine values are used in place of uracil. Thymine should be approximately 27 \AA^3 greater in volume than uracil, based on my atom type volumes for the additional thymine atoms.

bases have high standard deviations, high packing densities and low percentages of well-packed atoms (Figure 2.8b-d), because they do not benefit from any of the three structural interactions just mentioned. Therefore, the backbone-sugar regions and major groove atoms are less packed than the interior base atoms.

Role of crystal symmetry in volume size

Small RNA structures, in general, consist of a single helix and hence lack helix-helix packing, resulting in poorly packed backbones. In an effort to prevent this single helix dilemma, the crystal symmetry information contained in the PDB files was utilized. Crystal symmetry neighbors contribute additional relevant packing interactions. Alas, none of the software found for generating crystal symmetry neighbors is applicable for this purpose. All are either incapable of exporting the information to a file or do not have the capacity to generate all symmetry neighbors within a given distance of the target structure. Fortunately, matrix information on crystal symmetric rotations and translations are contained within most PDB headers (as well as in the online PDB format description, Berman *et al.*, 2000) and once recognized, it was simple to implement a small script to generate additional symmetry neighbors (Appendix E.2, page 262).

As shown in Table 2.5 (page 38), crystal symmetry had little to no effect on the final atomic volumes, but it significantly increased the number of well-packed atoms available for making calculations (Figure 2.9). The different disjoint subcategories of structures show that even though the set of ribosomes saw no benefit from crystal packing on the percentage of well-packed atoms, all other sets (which involve smaller molecules) had dramatic increases in the percentage of well-packed atoms after crystal symmetry was applied (Figure 2.9a, Table 2.2). The number of well-packed non-ribosome atoms increased from 30,272 to 32,816 after symmetry was applied. The high resolution structures saw the highest benefit from the use of crystal symmetry, the number of well-packed atoms increased from 49.4% to 58.7%. The well-packed atoms identified using the

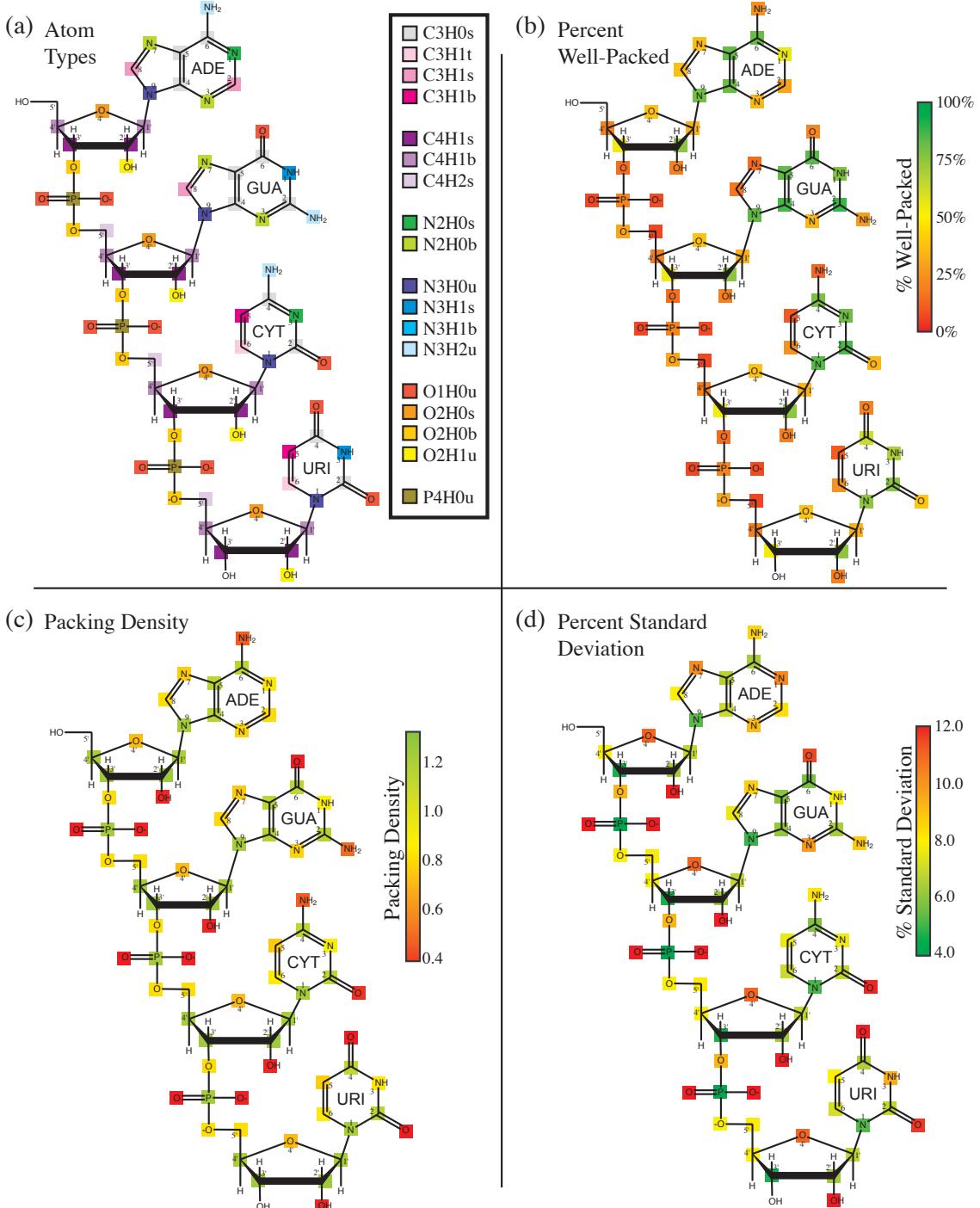


Figure 2.8: Graphical display of different atomic packing measurements. For each atom in RNA: (a) The atom typing of the RNA atoms shows how each atom is classified. (b) The percent well-packed atoms. Green atoms are almost always well-packed and red are almost never well-packed. (c) The packing density is defined as the VDW volume divided by Voronoi volume and measures the how tightly each atoms packs (Richards, 1974). This number is biased by the number of hydrogens bonded to an atom. (d) The percent standard deviation of the volume (standard deviation of the volume divided by the mean volume).

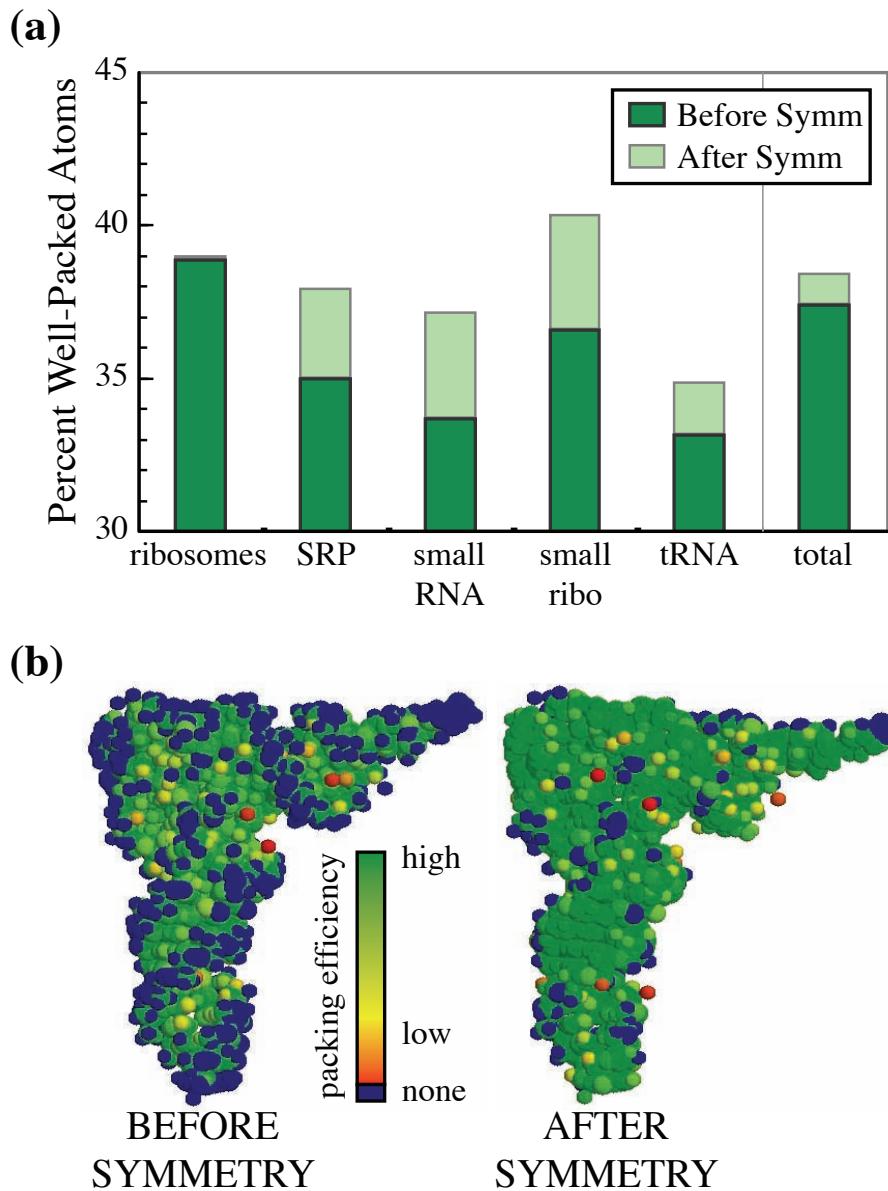


Figure 2.9: Effects of crystal symmetry. (a) Effect of the crystal symmetry on each subset of structures as measured by the number of well-packed atoms. (b) Example of one typical structure (PDB id, 1EHZ; Shi & Moore, 2000) where crystal packing alters the number of well-packed atoms. Shown in color is the packing efficiency, *i.e.* the individual atomic volume divided by the mean atomic volume. Blue represents atoms that have unclosed Voronoi polyhedra. Packing before crystal symmetry (left) and after (right).

symmetry neighbors not only provided more data for analysis, but also increased the amount of information available for backbone atoms that would normally have unclosed polyhedra.

Roles of different RNA structural categories

As noted, 69% of the atoms used to generate the *NucProt* data set are ribosomal atoms. *A priori* it is unjustified to assume short double stranded RNAs have the same packing properties as RNAs in large macromolecular complexes such as the ribosome. The ribosome is large enough in all three dimensions to truly have an interior while short double stranded RNA is completely exposed to solvent.

This issue is explored in Tables 2.3 and 2.4. The volume properties of RNA atoms in the ribosome are effectively the same as those of RNA atoms in small oligonucleotides. Table 2.3 clearly shows that the base size, sugar backbone, and entire nucleotide volumes differ by less than 9 Å³ from the smallest to largest values of the different categories, which is within the standard deviation. Ribosomal RNA volumes run slightly larger than the other structural categories. This could be due to the larger size of the complexes and their inherent problem of packing helices against other helices. Table 2.4 shows that atom type volumes are similarly insensitive. Despite slight variations between the different structural categories of RNAs analyzed , the final outcome of the volume calculations suggests that all RNA molecules pack in a universal way.

Role of solvent and proteins in RNA structures

To test the role solvent and protein atoms play in RNA structures, the largest RNA structure, the refined *Haloarcula marismortui* 50S ribosomal subunit from Klein *et al.* (2001) was used to conduct packing tests by systematically removing each kind of atom (Table 2.5). Crystal symmetry plays a small role in the 50S subunit, because there is more interior than surface. On the other hand, when the solvent is removed (leaving the RNA and protein), the percentage of well-packed atoms drops by 16.7% from the original value. Similarly, when the protein is removed (leaving the RNA and

Uracil (URI)									
atom	type	count	%OK	mode	mean	sd	%SD	min	max
N1	N3H0u	1641	74.7	8.71	8.80	0.46	5.24	7.82	10.11
C2	C3H0s	1649	75.1	9.01	9.20	0.63	6.84	7.99	11.14
O2	O1H0u	849	38.6	16.15	16.61	1.94	11.70	12.89	23.17
N3	N3H1s	1519	69.1	13.18	13.92	1.31	9.45	11.61	18.39
C4	C3H0s	1375	62.6	9.35	9.54	0.62	6.53	8.35	11.41
O4	O1H0u	439	20.0	15.81	16.83	2.07	12.30	12.98	23.69
C5	C3H1b	237	10.8	18.89	19.14	1.54	8.02	15.43	22.86
C6	C3H1t	545	24.8	16.42	16.98	1.22	7.17	14.24	20.30
TOTAL		8254	47.0	107.50	111.00	3.82	3.44	91.31	141.06
Cytosine (CYT)									
atom	type	count	%OK	mode	mean	sd	%SD	min	max
N1	N3H0u	2871	83.8	8.68	8.81	0.45	5.15	7.83	10.12
C2	C3H0s	2922	85.3	9.01	9.31	0.64	6.85	8.13	11.26
O2	O1H0u	1174	34.3	14.28	15.74	1.88	11.90	12.51	21.60
N3	N2H0s	2765	80.7	12.78	13.08	1.00	7.65	11.06	16.66
C4	C3H0s	2688	78.4	9.26	9.41	0.55	5.83	8.24	11.14
N4	N3H2u	438	12.8	21.78	22.48	1.85	8.22	18.32	27.28
C5	C3H1b	349	10.2	20.12	19.45	1.48	7.59	15.83	22.72
C6	C3H1t	842	24.6	16.96	16.92	1.18	6.97	14.04	20.20
TOTAL		14049	51.2	112.90	115.19	3.53	3.06	95.96	140.98
Adenine (ADE)									
atom	type	count	%OK	mode	mean	sd	%SD	min	max
N1	N2H0s	1696	55.4	13.07	13.94	1.43	10.30	11.50	18.82
C2	C3H1s	920	30.0	17.74	18.01	1.51	8.36	14.69	21.64
N3	N2H0b	1063	34.7	15.14	15.21	1.39	9.14	12.33	19.36
C4	C3H0s	2491	81.3	8.85	9.08	0.63	6.98	7.82	11.19
C5	C3H0s	2510	81.9	9.01	9.25	0.61	6.61	8.12	11.34
C6	C3H0s	2494	81.4	8.93	9.17	0.59	6.42	8.08	11.17
N6	N3H2u	726	23.7	22.45	22.45	1.93	8.58	18.27	27.70
N7	N2H0b	1108	36.2	14.60	15.63	1.59	10.20	12.55	20.85
C8	C3H1s	1114	36.4	18.15	17.81	1.41	7.91	14.57	21.55
N9	N3H0u	2454	80.1	8.68	8.77	0.47	5.34	7.74	10.12
TOTAL		16576	54.1	136.60	139.31	3.98	2.86	115.67	173.73
Guanine (GUA)									
atom	type	count	%OK	mode	mean	sd	%SD	min	max
N1	N3H1s	3292	77.8	13.01	13.50	1.07	7.90	11.59	17.42
C2	C3H0s	3696	87.4	8.78	9.03	0.58	6.41	7.98	10.89
N2	N3H2u	1158	27.4	21.16	21.74	1.91	8.80	17.55	27.34
N3	N2H0b	1490	35.2	14.19	14.96	1.47	9.83	12.16	19.49
C4	C3H0s	3631	85.8	8.86	9.03	0.55	6.12	8.01	10.86
C5	C3H0s	3485	82.4	9.05	9.24	0.55	5.96	8.18	11.04
C6	C3H0s	3538	83.6	9.10	9.27	0.53	5.75	8.21	10.96
O6	O1H0u	895	21.2	15.88	16.41	1.88	11.50	12.96	22.98
N7	N2H0b	624	14.7	15.37	15.89	1.38	8.71	12.67	20.12
C8	C3H1s	732	17.3	18.40	18.21	1.38	7.60	14.65	22.03
N9	N3H0u	3499	82.7	8.69	8.77	0.43	4.96	7.86	10.01
TOTAL		26040	56.0	142.50	146.04	3.97	2.72	121.81	183.13
Sugar Backbone (SUG)									
atom	type	count	%OK	mode	mean	sd	%SD	min	max
C1'	C4H1b	4271	33.1	13.37	13.34	0.91	6.81	11.11	15.57
C2'	C4H1s	9693	75.0	12.49	12.67	0.80	6.31	10.96	15.38
C3'	C4H1s	6861	53.1	12.61	12.62	0.59	4.70	11.06	14.60
C4'	C4H1b	2524	19.5	13.29	13.30	1.06	7.97	10.54	15.76
O2'	O2H1u	2468	19.2	16.69	17.39	2.28	13.10	12.94	24.50
O3'	O2H0b	1903	14.7	13.69	14.02	1.30	9.27	11.19	17.69
O4'	O2H0s	4850	37.5	12.26	12.73	1.40	11.00	10.05	17.09
C5'	C4H2s	790	6.1	22.65	21.74	1.77	8.12	16.98	25.63
O5'	O2H0b	3850	29.8	13.94	13.96	1.12	7.99	11.27	16.97
P	P4H0u	2645	20.5	11.85	11.86	0.23	1.98	11.18	12.33
O1P	O1H0u	793	6.2	15.94	16.10	2.27	14.10	11.42	23.32
O2P	O1H0u	902	7.0	15.19	16.49	2.37	14.40	12.22	23.71
TOTAL		41550	26.8	174.00	176.21	5.19	2.95	140.92	222.55

Table 2.4: Atom volumes for each nucleotide.

solvent), the percentage drops by 6.9%. Further, when only the RNA is used the final percentage drops by 20.9% from the original value. The final difference of 20.9% is very close to the sum (23.6%) of the other differences, suggesting an independence of the two atom classes. In addition, the solvent has a much larger effect on the loss of well-packed atoms for RNA than it does for proteins. This indicates that RNA atoms pack tightly against the solvent, which may be related to electrostatic effects. Despite these major differences in the percentages of well-packed atoms, there is no change in the final RNA volumes from the removal of solvent and protein atoms, reinforcing the idea that the *NucProt* set is self-consistent.

2.3.2 Comparison to Proteins and DNA

Partial specific volumes

To validate the computations just described, atomic volumes were used to calculate partial specific volumes (PSVs), which have often been measured for RNAs and proteins. The PSV is computed by summing the volumes of all the atoms in a molecule and then dividing by the molecular mass. Units are then converted from cubic Ångströms per Dalton ($\text{\AA}^3/\text{Da}$) to milliliters per gram (mL/g) (using conversion factor of $1 \text{\AA}^3/\text{Da} = N_A * 10^{-24} = 0.6022 \text{ mL/g}$). (A new online tool for calculating volumes and PSV for DNA, RNA and protein sequences is available at [http://geometry.molmovdb.org/*NucProt*/](http://geometry.molmovdb.org/NucProt/).)

It was shown more than 60 years ago that the volumes of proteins in solution can be accurately estimated from amino acid composition (Cohn & Edsall, 1943). For a given set of proteins, PSVs calculated from the amino acid composition differ average from experimental PSVs by 0.8% (Harpaz *et al.*, 1994). Using the same set of proteins, it was found that PSVs calculated using the Voronoi method differed from experimental values by only 0.65% on average (Harpaz *et al.*, 1994). Thus the Voronoi method is at least as good a means for PSV estimation as the older amino acid composition method.

Crystal Symm	+	-	-	-	-
Protein	+	+	-	+	-
Solvent	+	+	+	-	-
<i>Base Volumes (Å³)</i>					
GUA	145.9	145.9	145.7	146.4	146.2
ADE	140.0	140.0	139.4	138.9	138.2
CYT	115.5	115.5	115.3	115.6	115.0
URI	110.8	110.8	110.6	110.9	110.2
Sugar	176.1	176.1	175.4	179.2	177.4
<i>Additional Information</i>					
total atoms	33,245	33,204	28,996	23,007	20,409
%OK	54.0	53.9	47.1	37.3	33.1
%Closed	93.5	93.1	91.2	90.7	84.9
Mean %SD	6.89	6.89	6.78	7.10	6.55

Table 2.5: Summary of effect of other atoms on the packing calculations for the 50S ribosomal subunit. The 50S ribosomal subunit from *Haloarcula marismortui* was used in these calculations (Klein *et al.*, 2001; PDB id, 1JJ2).

Using the atomic volumes determined here, RNA is found to have an average calculated PSV of 0.569 mL/g, which implies that it is significantly denser than protein (\sim 0.74 mL/g) as has long been known. This value compares well to the published experimental estimates for RNA (0.54 mL/g; Durchschlag, 1986), but it is clearly not as accurate as the Voronoi-based protein PSV. The average PSV for RNA is about 5.4% greater than its experimental value, much larger than the 0.65% at proteins. This effect could be due to electrostriction of the surrounding solvent, which is not taken into account. It also turns out that RNA bases are more loosely packed than complete nucleotides (average calculated PSV, 0.610 mL/g), and it follows that the sugar and phosphate backbone is more tightly packed (0.544 mL/g).

Atom types

There are 18 atom types in RNA and 19 atom types in protein, but the two species share only 12 atom types in common (Figure 2.2, page 14). Figure 2.10b analyzes the 12 shared types in more detail. Though the volumes for atoms of a given type are similar in protein and RNA, the volume of most protein atoms of a given type (10 of 12) are slightly larger than those atoms of the same type in RNA (Table 2.6, Figure 2.10b). The two exceptions to this rule are *C3H0s* and *N3H1b* which are both larger in RNA. The most dramatic RNA-protein difference is shown by *C3H1s*, which consists of aromatic ring carbons in protein and purine ring carbons in RNA (Table 2.6). For *C3H1s*, the 25th percentile of the protein is almost greater than the 75th percentile of the RNA, suggesting two distinctly different values (Figure 2.10b, left). One reason RNA atom types are smaller in volume than equivalent protein types is their built-in chemical structure. Proteins are chains that have few atoms per residue and pack against one another to achieve tight packing, while RNA contains more than 18 atoms per residue and, therefore has inherent neighboring packing interactions. In fact, the worst packed atoms in RNA are either attached to the phosphorus atom or an extension off the ring structure (e.g. sugar *O2'*, guanine *O6*, and *O2* of purines). In essence, the atom type data shows that RNA is more tightly packed than protein atoms.

A

RNA Bases			Protein Side Chains		
Structure	Volume (\AA^3)	PSV	Structure	Volume (\AA^3)	PSV
	110.8	0.601		93.7	0.696
	115.0	0.629		127.0	0.839
	139.2	0.625		130.8	0.735
	145.9	0.585		162.6	0.752

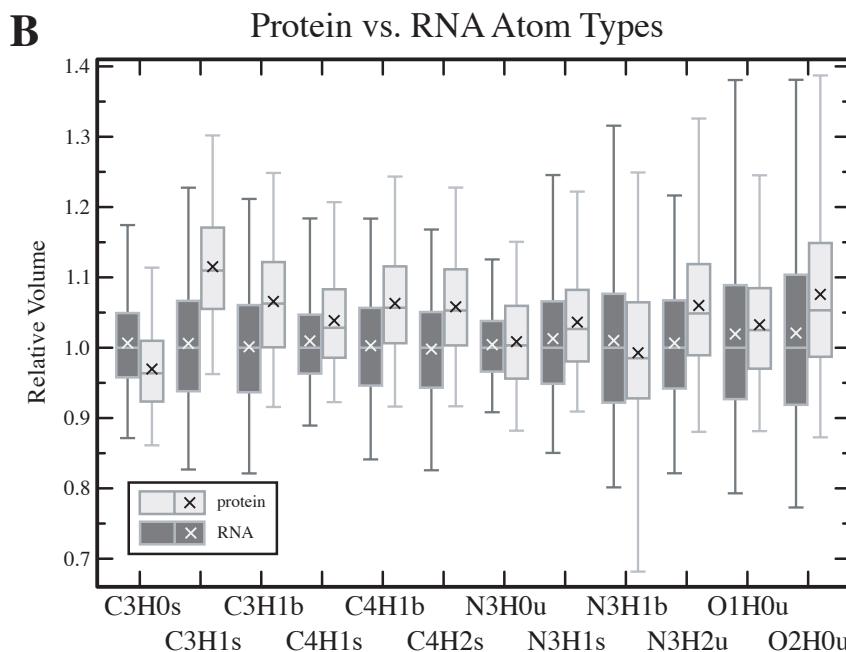


Figure 2.10: Comparison of atomic volumes from RNA to protein. (a) The comparison of aromatic protein residues to RNA bases. (b) Comparing Protein to RNA Atoms using Relative Volume. Relative volume is volume divided by the median RNA volume for that atom type. All 12 atoms in the intersection of the protein and RNA atom types are shown for comparison.

type	RNA NucProt Set						Protein NucProt Set (ProtOr)						%Δ
	num	%OK	count	mean	stdev	%SD	num	%OK	count	mean	stdev	%SD	
C3H0s	11	81.6%	30479	9.09	0.60	6.5%	20	76.1%	12097	8.76	0.63	7.2%	-3.7%
C3H0b							13	49.4%	4418	9.77	0.77	7.9%	-
C3H1t	2	24.7%	1387	16.95	1.19	7.0%							-
C3H1s	3	26.7%	2766	18.56	1.44	8.0%	8	43.5%	1888	20.57	1.81	8.8%	10.8%
C3H1b	2	10.4%	586	20.05	1.51	7.8%	8	55.3%	2181	21.34	1.90	8.9%	6.4%
C4H1s	2	64.1%	16554	12.87	0.72	5.7%	18	53.2%	7227	13.24	1.01	7.6%	2.9%
C4H1b	2	26.3%	6795	13.63	0.97	7.3%	6	54.8%	3747	14.45	1.33	9.2%	6.0%
C4H2s	1	6.1%	790	22.17	1.77	8.1%	20	25.6%	4468	23.50	2.34	10.0%	6.0%
C4H2b							7	27.3%	1137	24.42	2.14	8.8%	-
C4H3u							9	37.5%	3673	36.92	3.25	8.8%	-
N2H0s	2	68.7%	4461	13.41	1.26	9.4%							-
N2H0b	4	29.4%	4285	15.33	1.51	9.8%							-
N3H0u	4	81.0%	10465	8.78	0.45	5.1%	1	74.1%	592	8.81	0.66	7.5%	0.4%
N3H1s	2	74.8%	4811	13.47	1.17	8.6%	20	62.5%	10356	13.78	1.20	8.7%	2.3%
N3H1b	4	29.8%	4352	16.09	1.67	10.8%	4	29.0%	500	15.82	2.21	13.9%	-1.7%
N3H2u	3	21.7%	2322	22.15	1.94	8.8%	4	9.7%	286	23.33	2.77	11.8%	5.3%
N4H3u							1	1.2%	12	21.21	1.85	8.7%	-
O1H0u	6	13.4%	5052	15.92	2.09	12.8%	27	36.1%	8273	16.13	1.59	9.8%	1.3%
O2H0s	1	37.5%	4850	12.73	1.40	11.0%							-
O2H0b	2	22.3%	5753	13.98	1.18	8.4%							-
O2H1u	1	19.1%	2468	17.62	2.28	13.1%	3	20.0%	619	18.57	2.45	13.2%	5.4%
S2H0u							2	50.1%	280	29.17	2.81	9.6%	-
S2H1u							1	51.6%	63	34.60	5.73	16.6%	-
P4H0u	1	20.5%	2645	11.86	0.23	1.9%							-
TOTALS	53	55.4%	110821	11.24			176	52.6%	61817	14.23			

Table 2.6: Summary of atom types in protein and RNA. “num” stands for the number of atoms in RNA or protein that are of that atom type. For example, *O2H0b* contains the *O5'* and *O3'* backbone oxygens, so it has a value of 2. “%SD” is the standard deviation, “stdev,” divided by mean shown as a percentage. “%Δ” is defined as the mean protein volume minus the mean RNA volume divided by the RNA volume taken as a percentage.

Similar protein residues

An interesting question to ask is how the volume of RNA compares to the volume of amino acids of similar chemical structure. For example, how does the volume of an RNA purine compare to tryptophan and how does an RNA pyrimidine compare to phenylalanine, histidine, and tyrosine. Figure 2.10a reports the protein volumes for the side chains (calculated from the residue volume minus the volume of glycine) of tryptophan, tyrosine, phenylalanine, and histidine. While these values for the volume are all relatively close to the RNA base volumes (Table 2.3, page 31), the PSV tells a different story (Figure 2.10a). The average PSV for the RNA bases is 0.609 mL/g while the 4 protein side chains have an average PSV of 0.755 mL/g, suggesting once again that the RNA bases are much denser than protein aromatic side chains (Figure 2.10a). Further, if the total volume is divided by the number of atoms (including the hydrogen atoms), an average volume of 9.83 \AA^3 per atom for RNA bases and 11.25 \AA^3 per atom for protein residues is obtained. Though the volume per atom numbers are biased by the atom type volumes, this also highlights the observation that RNA seems to pack more tightly than protein. These results may be due to nucleotide base rings containing more nitrogen atoms than the amino acid aromatic rings or the fact that RNA rings have more atoms attached to them, creating a large number of inherent neighbors. In addition, RNA duplex base stacking may contribute favorably to achieve this tighter packing. Though it is difficult to directly compare these vastly different chemical structures, I found that the RNA bases are more tightly packed than aromatic protein residues.

DNA volumes

Recently, Nadassy *et al.* (2001) published standard atomic volumes for double stranded DNA. Comparing the volume of RNA in large macromolecular structures to that of A-form DNA (A-DNA) reveals that there are no significant differences between the two structures (Table 2.3, page 31). The volume of RNA bases: adenine, guanine and cytosine are larger by only 0.9 \AA^3 , 0.5 \AA^3 , and 1.1 \AA^3 ,

respectively, than those found for the complementary bases in A-DNA. (Since uracil does not exist in DNA, its volume is compared to the volume for thymidine allowing 27 \AA^3 for its “extra” methyl group.) In RNA structures about half of the base volumes are within a standard deviation of the A-DNA base volumes (Table 2.3), suggesting similar packing of the bases. The sugar-phosphate backbone on the other hand differs in volume by slightly more. The A-DNA sugar plus phosphate reported by Nadassy *et al.* (2001) is 5.7 \AA^3 larger than the RNA backbone reported here (Table 2.3). Though one should expect the backbone atoms of A-DNA to be 8.3 \AA^3 smaller due to the additional volume taken up by the 2' oxygen (based on its atom type volume), this is not the case. Furthermore, in DNA the 2' carbon (atom type, *C4H1s*) volume was reported to be 18.0 \AA^3 , while I found a volume for this atom type to be 12.65 \AA^3 (Table 2.6); this drop is expected because of the loss of the hydrogen. But if the 2' oxygen (atom type, *O2H1u*) volume of 17.39 \AA^3 is taken into account, a total volume of 30.04 \AA^3 remains to fit into the space of 18.0 \AA^3 . This implies that RNA structure must accommodate for this extra space. In summary, the published A-DNA volumes are approximately equal for the bases and differ slightly for the backbone where A-DNA is packed less tightly than RNA.

2.4 CONCLUSIONS

In this study, a careful parameterization of currently available RNA structures was performed to obtain a universal, self-consistent set of volumes, denoted as the *NucProt* parameter set. This composite set can be applied to both RNA and protein. In addition, the aspect of several factors such as crystal symmetry, structural complexity and protein and solvent interactions on the volumes obtained were evaluated and found to be modest. Using two measures, the percentage of well-packed atoms and final volumes, the impact of several factors was assessed on the data. While all the factors affected the percentage of well-packed atoms, none of them had any significant effect on the final volumes. From these volume calculations, it is immediately apparent that in RNA

the backbone is not as tightly packed as its bases as determined by its high standard deviation and its low percentage of well-packed atoms. For RNA, the calculated partial specific volume corresponded well with its experimental value. When compared to proteins, RNA is found to be denser. Comparing common atom types between protein and RNA showed that in 10 of 12 cases, RNA has a smaller volume and is therefore packed tighter. Further, when comparing aromatic protein side chains to the RNA bases, the partial specific volume for RNA bases was again smaller than the protein side chains as well as their average volume per atom. Thus, RNA packs more tightly than protein, but based only on well-packed atoms. A-form DNA, on the other hand, has approximately the same base volumes as RNA, but their backbone volumes differ significantly, which is probably related to tertiary structure formation in RNA. In conclusion, RNA packs more tightly than protein and approximately the same as DNA.

Location of files, programs, scripts and statistics

Visit the *NucProt* website at <http://geometry.molmovdb.org/NucProt> for the parameter sets, additional statistics, perl, and shell scripts, packaged program files, and the raw volume data.

2.5 REFERENCES

- P. L. Adams, M. R. Stahley, A. B. Kosek, J. Wang, and S. A. Strobel (2004). “Crystal structure of a self-splicing group I intron with both exons”. *Nature*, **430** (6995): pp. 45–50.
- N. Ban, P. Nissen, J. Hansen, P. B. Moore, and T. A. Steitz (2000). “The complete atomic structure of the large ribosomal subunit at 2.4 Å resolution”. *Science*, **289** (5481): pp. 905–20.
- H. M. Berman, W. K. Olson, D. L. Beveridge, J. Westbrook, A. Gelbin, T. Demeny, S. H. Hsieh, A. R. Srinivasan, and B. Schneider (1992). “The nucleic acid database. A comprehensive relational database of three-dimensional structures of nucleic acids”. *Biophys J*, **63** (3): pp. 751–9.
- H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne (2000). “The Protein Data Bank”. *Nucleic Acids Res*, **28** (1): pp. 235–42.
- J. D. Bernal and J. L. Finney (1967). “Random close-packed hard-sphere model II. geometry of random packing of hard spheres.” *Disc Faraday Soc*, **43**: pp. 62–69.
- A. Bondi (1964). “van der Waals volumes and radii”. *J Phys Chem*, **68**: pp. 441–451.
- J. H. Cate, A. R. Gooding, E. Podell, K. Zhou, B. L. Golden, C. E. Kundrot, T. R. Cech, and J. A. Doudna (1996). “Crystal structure of a group I ribozyme domain: Principles of RNA packing”. *Science*, **273** (5282): pp. 1678–85.
- C. Chothia (1974). “Hydrophobic bonding and accessible surface area in proteins”. *Nature*, **248** (446): pp. 338–9.
- C. Chothia (1975). “Structural invariants in protein folding”. *Nature*, **254** (5498): pp. 304–8.
- E. J. Cohn and J. T. Edsall (1943). *Proteins, amino acids and peptides as ions and dipolar ions*. Monograph series (American Chemical Society) [No. 90]. Reinhold Publishing Corporation, New York.,
- C. W. David (1988). “Voronoi polyhedra as structure probes in large molecular systems”. *Biopolymers*, **27** (2): pp. 339–44.
- J. A. Doudna and J. H. Cate (1997). “RNA structure: Crystal clear?” *Curr Opin Struct Biol*, **7** (3): pp. 310–6.
- J. Dunbrack, R. L. (1999). “Comparative modeling of CASP3 targets using PSI-BLAST and SCWRL”. *Proteins, Suppl* **3**: pp. 81–7.
- H. Durchschlag (1986). “Specific volumes of biological macromolecules and some other molecules of biological interest”. In H.-J. Hinz, ed., “Thermodynamic data for biochemistry and biotechnology.”, pp. 45–128. Springer, Berlin.
- R. A. Engh and R. Huber (1991). “Accurate bond and angle parameters for X-ray protein-structure refinement”. *Acta Cryst A*, **47**: pp. 392–400.

- J. L. Finney (1975). “Volume occupation, environment and accessibility in proteins. The problem of the protein surface”. *J Mol Biol*, **96** (4): pp. 721–32.
- J. L. Finney (1978). “Volume occupation, environment, and accessibility in proteins. Environment and molecular area of RNase-S”. *J Mol Biol*, **119** (3): pp. 415–41.
- B. J. Gellatly and J. L. Finney (1982). “Calculation of protein volumes: An alternative to the Voronoi procedure”. *J Mol Biol*, **161** (2): pp. 305–22.
- M. Gerstein (1998). “How representative are the known structures of the proteins in a complete genome? A comprehensive structural census”. *Fold Des*, **3** (6): pp. 497–512.
- M. Gerstein and C. Chothia (1996). “Packing at the protein-water interface”. *PNAS*, **93** (19): pp. 10167–72.
- M. Gerstein and W. Krebs (1998). “A database of macromolecular motions”. *Nucleic Acids Res*, **26** (18): pp. 4280–90.
- M. Gerstein, E. L. Sonnhammer, and C. Chothia (1994). “Volume changes in protein evolution”. *J Mol Biol*, **236** (4): pp. 1067–78.
- M. Gerstein, J. Tsai, and M. Levitt (1995). “The volume of atoms on the protein surface: Calculated from simulation, using Voronoi polyhedra”. *J Mol Biol*, **249** (5): pp. 955–66.
- J. Harms, F. Schlüzen, R. Zarivach, A. Bashan, S. Gat, I. Agmon, H. Bartels, F. Franceschi, and A. Yonath (2001). “High resolution structure of the large ribosomal subunit from a mesophilic eubacterium”. *Cell*, **107** (5): pp. 679–88.
- Y. Harpaz, M. Gerstein, and C. Chothia (1994). “Volume changes on protein folding”. *Structure*, **2** (7): pp. 641–9.
- S. J. Hubbard and P. Argos (1995). “Detection of internal cavities in globular proteins”. *Protein Eng*, **8** (10): pp. 1011–5.
- J. Janin (1979). “Surface and inside volumes in globular proteins”. *Nature*, **277** (5696): pp. 491–2.
- J. Janin and C. Chothia (1990). “The structure of protein-protein recognition sites”. *J Biol Chem*, **265** (27): pp. 16027–30.
- A. V. Kazantsev, A. A. Krivenko, D. J. Harrington, S. R. Holbrook, P. D. Adams, and N. R. Pace (2005). “Crystal structure of a bacterial ribonuclease P RNA”. *PNAS*, **102** (38): pp. 13392–7.
- D. J. Klein, T. M. Schmeing, P. B. Moore, and T. A. Steitz (2001). “The kink-turn: A new RNA secondary structure motif”. *Embo J*, **20** (15): pp. 4214–21.
- P. Koehl and M. Delarue (1997). “The native sequence determines sidechain packing in a protein, but does optimal sidechain packing determine the native sequence?” *Pac Symp Biocomput*, **2**: pp. 198–209.
- A. S. Krasilnikov, X. Yang, T. Pan, and A. Mondragon (2003). “Crystal structure of the specificity domain of ribonuclease P”. *Nature*, **421** (6924): pp. 760–4.
- W. G. Krebs and M. Gerstein (2000). “The morph server: A standardized system for analyzing and visualizing macromolecular motions in a database framework”. *Nucleic Acids Res*, **28** (8): pp. 1665–75.

- A. Lee and T. V. Chalikian (2001). “Volumetric characterization of the hydration properties of heterocyclic bases and nucleosides”. *Biophys Chem*, **92** (3): pp. 209–27.
- C. Lee and M. Levitt (1997). “Packing as a structural basis of protein stability: Understanding mutant properties from wildtype structure”. *Pac Symp Biocomput*, **2**: pp. 245–55.
- A. J. Li and R. Nussinov (1998). “A set of van der Waals and coulombic radii of protein atoms for molecular and solvent-accessible surface calculation, packing evaluation, and docking”. *Proteins*, **32** (1): pp. 111–27.
- J. Liang, H. Edelsbrunner, P. Fu, P. V. Sudhakar, and S. Subramaniam (1998a). “Analytical shape computation of macromolecules: I. Molecular area and volume through alpha shape”. *Proteins*, **33** (1): pp. 1–17.
- J. Liang, H. Edelsbrunner, P. Fu, P. V. Sudhakar, and S. Subramaniam (1998b). “Analytical shape computation of macromolecules: II. Inaccessible cavities in proteins”. *Proteins*, **33** (1): pp. 18–29.
- K. Nadassy, I. Tomas-Oliveira, I. Alberts, J. Janin, and S. J. Wodak (2001). “Standard atomic volumes in double-stranded DNA and packing in protein–DNA interfaces”. *Nucleic Acids Res*, **29** (16): pp. 3362–76.
- G. Parkinson, J. Vojtechovsky, L. Clowney, A. T. Brünger, and H. M. Berman (1996). “New parameters for the refinement of nucleic acid-containing structures”. *Acta Cryst D*, **52** (1): pp. 57–64. <http://dx.doi.org/10.1107/S090744995011115>.
- J. Pontius, J. Richelle, and S. J. Wodak (1996). “Deviations from standard atomic volumes as a quality measure for protein crystal structures”. *J Mol Biol*, **264** (1): pp. 121–36.
- F. M. Richards (1974). “The interpretation of protein structures: Total volume, group volume distributions and packing density”. *J Mol Biol*, **82** (1): pp. 1–14.
- F. M. Richards (1985). “Calculation of molecular volumes and areas for structures of known geometry”. *Methods Enzymol*, **115**: pp. 440–64.
- F. Schluelzen, A. Tocilj, R. Zarivach, J. Harms, M. Gluehmann, D. Janell, A. Bashan, H. Bartels, I. Agmon, F. Franceschi, and A. Yonath (2000). “Structure of functionally activated small ribosomal subunit at 3.3 Å resolution”. *Cell*, **102** (5): pp. 615–623.
- H. Shi and P. B. Moore (2000). “The crystal structure of yeast phenylalanine tRNA at 1.93 Å resolution: A classic structure revisited”. *RNA*, **6** (8): pp. 1091–105.
- J. Tsai and M. Gerstein (2002). “Calculations of protein volumes: Sensitivity analysis and parameter database”. *Bioinformatics*, **18** (7): pp. 985–95.
- J. Tsai, M. Gerstein, and M. Levitt (1997). “Simulating the minimum core for hydrophobic collapse in globular proteins”. *Protein Sci*, **6** (12): pp. 2606–16.
- J. Tsai, R. Taylor, C. Chothia, and M. Gerstein (1999). “The packing density in proteins: Standard radii and volumes”. *J Mol Biol*, **290** (1): pp. 253–66.
- J. Tsai, N. Voss, and M. Gerstein (2001). “Determining the minimum number of types necessary to represent the sizes of protein atoms”. *Bioinformatics*, **17** (10): pp. 949–56.

G. F. Voronoi (1908). “Nouvelles applications des paramètres continus à la théorie de formes quadratiques.” *J Reine Angew Math*, **134**: pp. 198–287.

B. T. Wimberly, D. E. Brodersen, W. M. J. Clemons, R. J. Morgan-Warren, A. P. Carter, C. Vonrhein, T. Hartsch, and V. Ramakrishnan (2000). “Structure of the 30S ribosomal subunit”. *Nature*, **407** (6802): pp. 327–339.

Chapter 3

Macro-volumes: Ribosome Volume Assessment Using a Rolling Probe Sphere

3.1 INTRODUCTION

In Chapter 2 it was shown that on average, the volumes occupied by atoms in RNA molecules are slightly smaller than the volumes occupied by chemically similar atoms in proteins. Since the atomic weight of the average RNA atom is greater than the average protein atom, it follows that the partial specific volume of RNA molecules should be substantially smaller than that of protein molecules, and it is. On the other hand, it was realized long ago that proteins are tightly packed aggregates of globular domains and in the interior of the average globular domain there is little room for anything but protein atoms (Lee & Richards, 1971; Shrake & Rupley, 1973). By contrast, the interiors of large RNA-rich structures, and specifically the large ribosomal subunit, are full of gaps large enough to contain solvent molecules (Ban *et al.*, 1999). The following analysis of the volumes inside the large ribosomal subunit was undertaken to provide a quantitative characterization of the way RNA packs in large macromolecular assemblies.

Several software packages have been used to estimate the internal solvent content of macromolecules, especially proteins, but the results obtained on the ribosome using these packages proved largely unsatisfactory (see Appendix B). The ribosome is particularly challenging because it has over 90,000 non-hydrogen atoms and its surface contains large scale concavities making it difficult

to decide which volumes should be considered inside the particle. This problem was solved by defining a surface surrounding the particle called its “shell.” The rolling probe algorithm of Richards (1977), the oldest and simplest of all the methods devised for exploring molecular surfaces, was used to define the shell of the large ribosomal subunit. Once the particle’s shell had been defined, the task of determining how the volume inside the shell is allocated was relatively straightforward.

3.2 RESULTS

3.2.1 Formalism

In the discussion of the geometry of the ribosome that follows, eight kinds of molecular volumes are considered. Their definitions follow:

1. **Convex hull volume:** The convex hull of any object is the smallest convex surface that entirely surrounds the set of van der Waals (VDW) spheres for macromolecules. The convex hull volume of an object is the volume inside its convex hull.
2. **Shell volume:** The “shell” of a macromolecule is the limiting surface used to distinguish the interior of the macromolecule from its exterior. The shell volume is the volume inside that surface. The definition of the shell is discussed in more detail below (see page 52).
3. **Solvent-accessible volume:** The solvent-accessible volume of a macromolecule corresponds to the volume inside the accessible surface defined for it by the rolling probe method using a 1.5 Å radius probe (Figure 3.1, Richards, 1977). However it is not computed that way. Instead, every atom in the structure is treated as a sphere the radius of which is the sum of its non-bonded van der Waals radius plus 1.5 Å, the radius of a water molecule. The solvent-accessible volume is the sum of the volume of all such spheres with regions where spheres overlap counted only once (Lee & Richards, 1971).
4. **Solvent-excluded volume:** The solvent-excluded volume of a molecule is the volume inside the excluded surface of that molecule determined using a spherical probe of 1.5 Å (Figure 3.1).

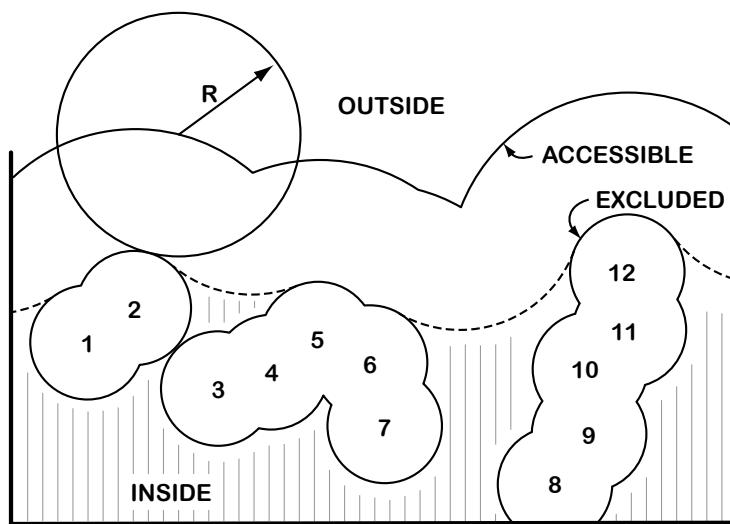


Figure 3.1: Richards' rolling sphere definitions. A sphere of radius R is shown rolling across the surface of a macromolecule defined by the 12 atoms. The excluded surface is the surface defined by the exterior of the sphere as it rolls across the array of atoms (dashed line). The accessible surface is the surface created by the center of the sphere (top solid line) as it rolls across the molecular surface. It is called the accessible surface because the center of a sphere of radius R cannot enter the volume inside the accessible surface (Richards, 1977).

5. **Van der Waals (VDW) volume:** The van der Waals volume of a molecule is the total volume occupied by its atoms. It is computed assuming each atom is a sphere having its non-bonded van der Waals radius, and counting volumes where van der Waals spheres overlap only once.
6. **Empty volume:** The empty volume of a molecule is the difference between its solvent-excluded volume and its van der Waals volume. It corresponds to the sum of the small volumes in between the atoms that are not accessible to a solvent molecule.
7. **Solvent volume:** The solvent volume of a molecule is the difference between its shell volume and its solvent-excluded volume. This corresponds to all points inside the shell volume large enough to accommodate a solvent molecule that do not overlap with the solvent-excluded volume.
8. **Cavity volume:** The cavity volume of a molecule is the sum of the volumes inside its shell that are large enough to accommodate a spherical probe, but are not connected to the molecule's shell by at least one passage traversable by the probe.

3.2.2 Defining the Shell

Essential properties of the shell

All of the volume types above are strictly defined and straightforward to calculate, except the shell volume. As already pointed out, the shell of a molecule is a surface used to determine which spaces are interior to the macromolecule and which are exterior, but it is not so obvious how the shell of a macromolecule can be obtained. In my view, the essential properties of a shell are: (1) that it enclose all the volumes inside the invaginations and other concavities in the surface of a macromolecule, (2) that the volume of exterior solvent captured within it be otherwise minimal, and (3) that its surface be connected. Every point on a connected surface can be reached from every other point by a trajectory lying entirely on that surface.

Initially, I considered the convex hull as a delimiting surface (or shell) for the ribosome; others have used convex hulls as a boundary between the interior and exterior of macromolecules (Laskowski, 1995; Gabashvili *et al.*, 2001). The virtues of the convex hull are (1) it is rigorously

defined, (2) it encloses all the volumes of interest and (3) it always yields a connected surface. But when the convex hull for the ribosome was examined it became evident that large amounts of bulk solvent are included inside it, due to large, lateral stalks of the structure (Figure 3.2).

In the end, I wrote a program that uses the rolling probe algorithm to obtain a shell. The algorithm can generate an infinite number of surfaces for a macromolecule simply by varying the radius of probe used. Below I explore the geometric properties of this family of surfaces and rationalize the use of a specific member of it as the shell of the large ribosomal subunit. Figure 3.1 shows what is meant by the term, excluded surface in this context (Richards, 1977).

Setting a lower bound

The lower bound for the radius of the sphere used to define the shell of the large ribosomal subunit was set by the requirement that the shell be a connected surface. The interiors of space-filling atomic models of macromolecules are full of “empty” regions, some big enough to contain solvent molecules, but others so small they can accommodate nothing. From the point of view of connectivity, the interior voids of concern are cavities. A cavity is any void big enough to accommodate a sphere larger than the largest sphere capable of entering it via a pathway connecting it to the exterior solvent. A rolling sphere surface with cavities will necessarily have a outer surface that encloses smaller surfaces not connected to it. Thus if the rolling sphere excluded surface of a molecule is to be a connected surface, the radius of the sphere must be large enough so that there are no cavities in the macromolecule big enough to accommodate that sphere.

As Figure 3.3 shows, the number of volumes inside a macromolecule identified as cavities by the rolling probe method depends upon probe radius. The function is highly erratic with cavities appearing and disappearing seemingly at random. (Figure 3.3B). The short “life-span” of the cavities results in the apparent fluctuations for the cavity volume as the probe radius increases (Figure 3.3A). No appreciable amount of cavity volume exists for probe radii less than 0.5 Å. The cavity volume reaches a maximum when the probe radius is 3–6 Å. Further, once the probe radius increases above 8 Å there is only one cavity left. It is a void in the L11 stalk (Figure 3.3B, bottom right) that is filled by a protein in the *D. radiodurans* structure (Daniel Klein, personal

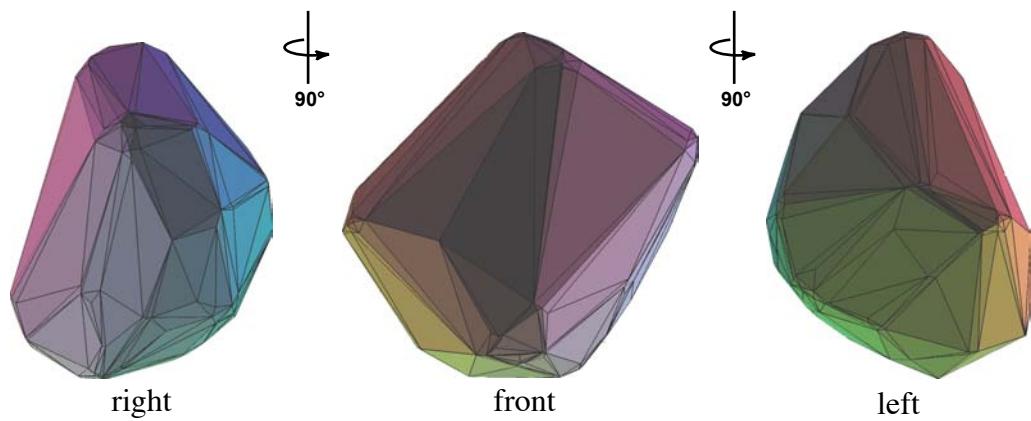


Figure 3.2: Convex hull of the ribosomal large subunit. The convex hull of the ribosome is shown at three different angles. The “front” is the normal “crown view” of the molecule and the side images are 90° rotations in each direction from that view. Figure was generated using Geomview (Phillips *et al.*, 1993).

communication). Also, there is a critical radius beyond which there are no cavities at all. The large ribosomal subunit contains no cavities big enough to enclose a sphere having a radius much greater than 10 Å. Therefore, the excluded surface defined by a 10 Å sphere is connected, as required for it to be a suitable shell surface.

Avoiding bulk solvent inclusion

Given that 10 Å is the radius of the smallest sphere that can be used to generate a satisfactory shell for the large ribosomal subunit, would a surface defined using a larger diameter sphere be just as satisfactory? The answer to the question emerges when the excluded surface of the large ribosomal subunit determined using a 10 Å probe is compared to the excluded surface of the same object obtained using a 100 Å radius probe (Figure 3.4). The large ribosomal subunit is a compact, more or less isometric mass from which three large projections protrude. In addition to having small scale (less than 10 Å) surface concavities due to its molecular granularity, its surface has much larger scale (\sim 100 Å) concavities caused by these projections. Most of the small scale surface concavities of the large ribosomal subunit are not represented in the excluded surface obtained using a 10 Å radius sphere, but its larger scale concavities remain conspicuous. The large scale concavities of the particle are less obvious in the excluded surface obtained using a 100 Å radius sphere because they are partially smoothed, but for the same reason, the surface that emerges includes large amounts of bulk solvent. Thus to keep the amount of extra solvent encompassed by the shell to a minimum, the shell of a macromolecule should be defined using the smallest spherical probe that yields a connected surface.

Surface area and volume flattening

Further insight into the choice of the 10 Å probe radius for determining shells can be obtained by examining the dependence of the area of the excluded surface of the large ribosomal subunit and volume inside that surface on probe radius. Figure 3.5 shows that the excluded surface area of the ribosome drops precipitously with increasing probe radius at less than 10 Å. At higher probe radii, the surface area remains essentially constant. The rapid initial fall in surface area is due to

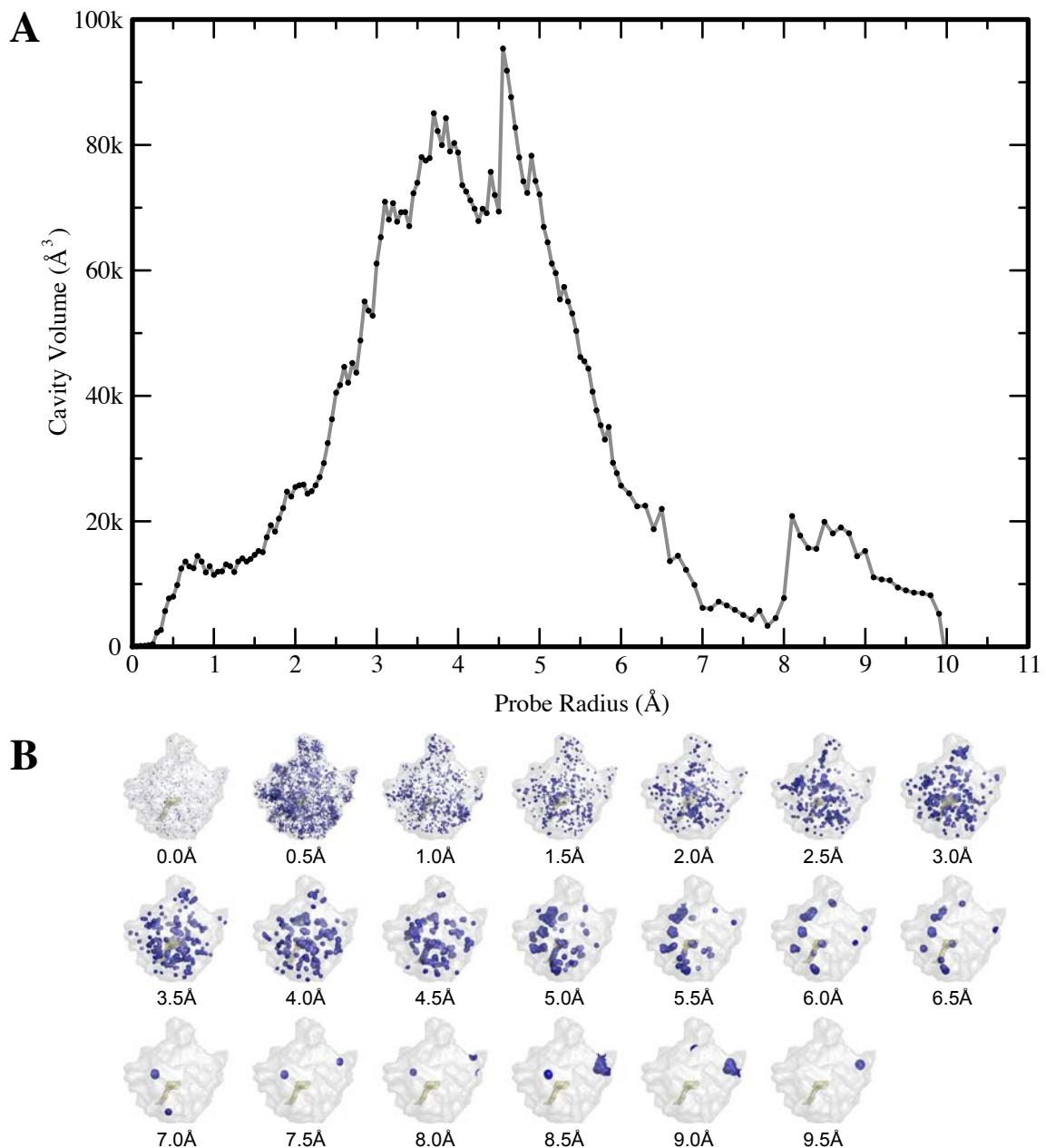


Figure 3.3: Cavity volume as a function of probe radius. (A) The cavity volume of the large subunit of the ribosome is plotted as a function of probe radius. (B) Pictures of the cavity volume at probe radii in increments of 0.5 Å. The 10 Å shell of the ribosome is silhouetted over the cavities and the ribosomal exit tunnel is shown in mesh as a point of reference.

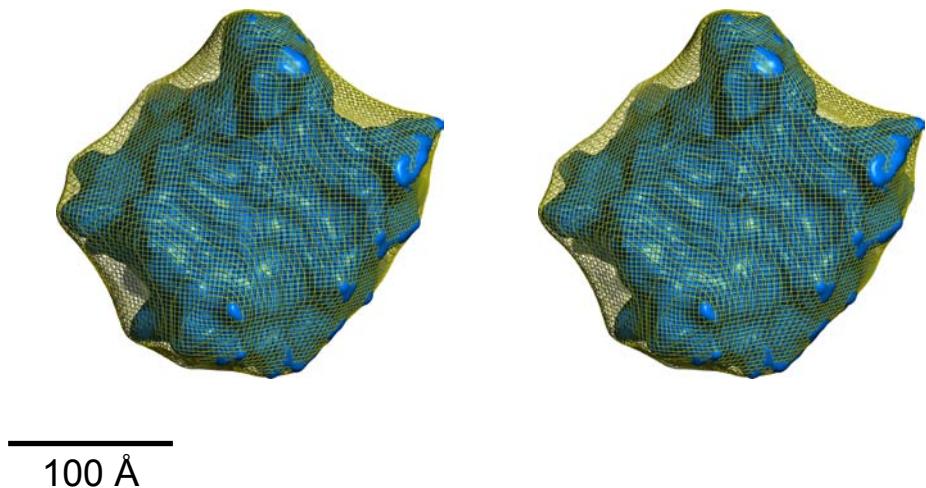


Figure 3.4: The 10 Å excluded surface of the *H. marismortui* large subunit compared to its 100 Å excluded surface. The structure whose excluded surfaces are shown is that described by PDB entry 1JJ2 (Klein *et al.*, 2001). A stereo view is provided of the excluded surface defined for the particle using a sphere of radius 10 Å, *i.e.* its shell surface, (blue solid) with the excluded surface obtained for the same object using a sphere that is 100 Å in radius (yellow mesh) superimposed.

the smoothing of atomic-scale concavities in the subunit surface. Beyond this radius, decreases in surface area produced by local smoothing effects are offset by increases caused by the expansion of the shell that accompanies the smoothing of the subunit's larger scale concavities. In the limit of infinite probe radius the surface area and volume of the shell equals that of the convex hull, as has been noted before (Sanner *et al.*, 1995; Sanner, 1995; Gerstein & Richards, 2001). In contrast, the volume inside the shell increases rapidly with probes smaller than 10 Å and then afterwards slowly approaches the convex hull (Figure 3.5). Since that expansion is associated with growth in the amount of bulk solvent inside the excluded surface, it appears that shells defined using probes having radii around 10 Å are likely to be acceptable. In summary, the 10 Å probe radius works because the surface area has reached its limiting value and the volume is now only filling in outer regions.

3.2.3 Statistical Characteristics of Surface Roughness

The dependence of the volume and surface area of a particle on probe radius provides interesting statistical information about the shape of the particle. For example, using the volume and surface areas resulting from my rolling probe calculations on the ribosome, I was able to address its surface roughness.

Effective radius

Statistical information about the shape of the ribosome can be extracted from the data plotted in Figure 3.5 by considering the quotients obtained by dividing volumes by surface areas. Volume has the dimensions of length-cubed and area has units of length-squared, hence a volume divided by an area has the dimensions of length. To better understand what that length means, it is useful to consider the sphere that has the same ratio of volume to surface area. Its radius, r_{eff} , is computed as follows:

$$\frac{V}{A} = \frac{\frac{4}{3}\pi r_{eff}^3}{4\pi r_{eff}^2} = \frac{r_{eff}^3}{3r_{eff}^2} = \frac{r_{eff}}{3}$$

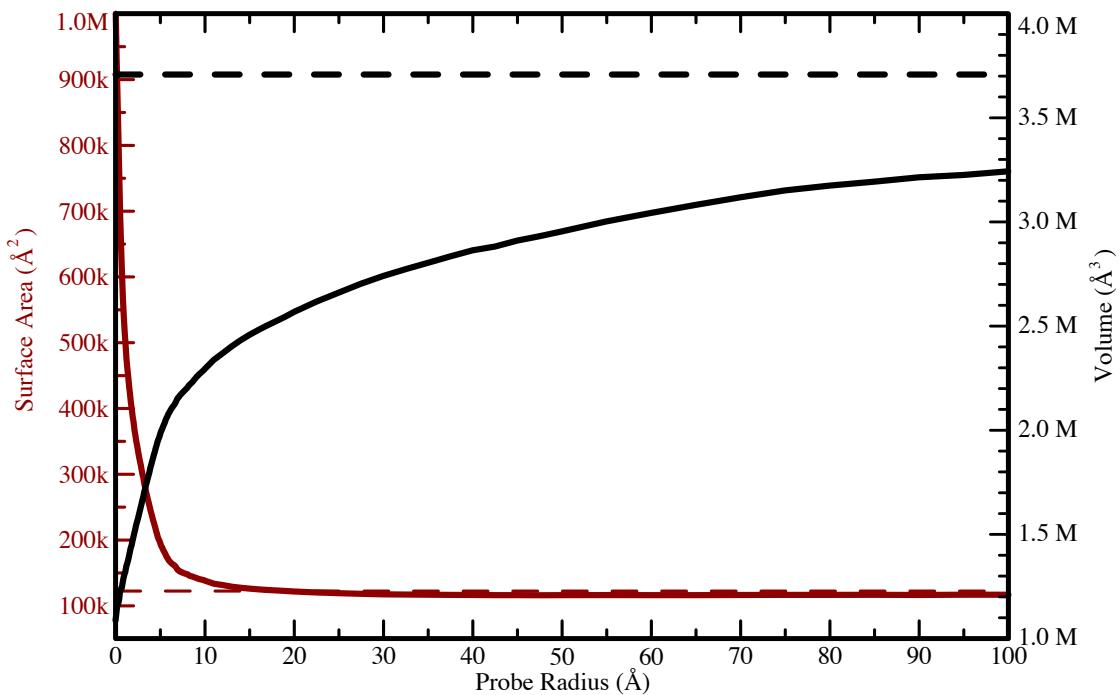


Figure 3.5: Volume and surface area as a function of probe radius. Both the area of the excluded surface of the *H. marismortui* large ribosomal subunit and its interior volume are plotted as a function of the probe radius. The convex hull volume and convex hull surface area are indicated by dashed lines for reference. Areas are plotted in red, and volumes are plotted in black.

where V and A are the volume and surface area of the particle's excluded surface, respectively. Rearranging the variables we get a function for the effective radius in terms of the volume and surface area:

$$r_{eff} = 3 \frac{V_p}{A_p} \quad (3.1)$$

Figure 3.6A shows how the effective radius of the large ribosomal subunit depends upon the probe radius used to define its surface. If the surface of an object includes concave features, the effective sphere radius, r_{eff} , should change as the probe radius increases, and in the limit of infinite probe radius reach the value appropriate for its convex hull. In fact, the effective radius is directly related to the Sauter mean diameter (SMD), \bar{d}_{32} used in the field of atomization and sprays to obtain accurate measurements of drop diameters (Sauter, 1926).

Given the volume and surface area, two additional independent radii can be calculated. The volume radius, r_{vol} , is simply the radius of a sphere with the same volume as the particle's excluded surface V_p :

$$r_{vol} = \left(\frac{3V_p}{4\pi} \right)^{\frac{1}{3}} \quad (3.2)$$

Likewise, the surface radius, r_{surf} , is the radius of a sphere with the same surface area as the particle's excluded surface, A_p :

$$r_{surf} = \left(\frac{A_p}{4\pi} \right)^{\frac{1}{2}} \quad (3.3)$$

These two quantities along with the effective radius are plotted as a function of the probe radius in Figure 3.6A. It is evident that the volume radius, r_{vol} (black line), varies with the probe radius much less than the other two radii, r_{eff} (blue line) and r_{surf} (red line), both of which depend on the surface area that depends strongly on probe radius at small radii. As expected the r_{surf} quantity is large at small probes, because of the many invaginations in the VDW surface of the particle. At large probe radius, the values for r_{surf} , r_{vol} , and r_{eff} approach each other, but never become equal

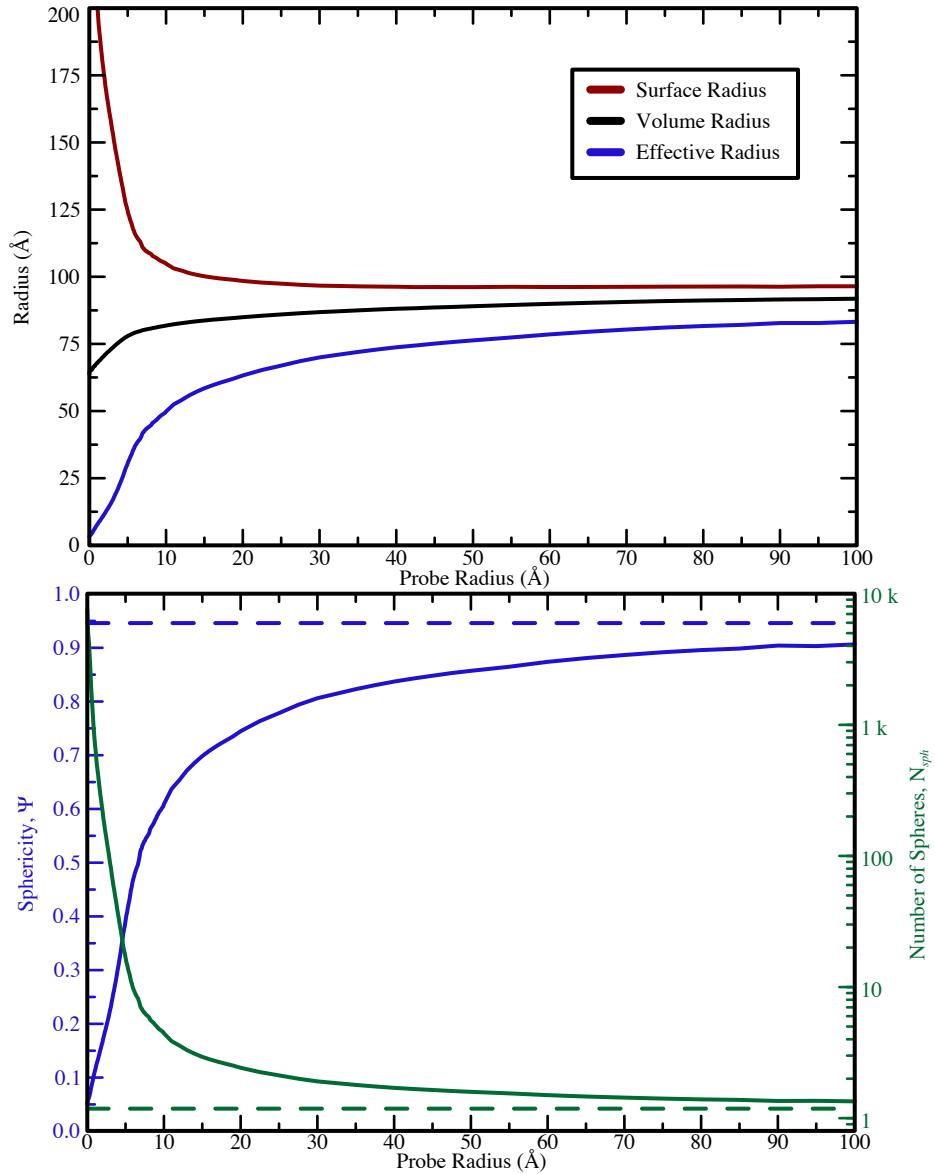


Figure 3.6: Effective spheres as a function of probe radius. (A) The surface radius, r_{surf} (red), volume radius, r_{vol} (black), and effective radius, r_{eff} (blue) of the large ribosomal subunit as a function of probe radius. (B) The number of effective spheres, N_{sph} (green) and Wadell (1932) sphericity, Ψ (blue) of the large subunit plotted against the probe radius. The limiting values for the infinite probe, the convex hull, are shown as dashed lines. *Note:* the vertical scale showing the number of effective spheres, N_{sph} (green) is logarithmic while the sphericity, Ψ (blue) is linear.

because the convex hull is not a perfect sphere. For a perfect sphere all three values are equal to each other and to the radius of that sphere.

Number of effective spheres

One way to evaluate the degree to which the surface surrounding a particle resembles a sphere is to compute the number of effective spheres it would take to account for the volume inside that surface. This number, N_{sph} , can be obtained by computing how many identical spheres it would take to have the same surface area, A_p or the same volume, V_p as the surface surrounding the particle. Thus:

$$N_{sph} = \frac{A_p}{4\pi r_{eff}^2} = \frac{V_p}{\frac{4}{3}\pi r_{eff}^3} \quad (3.4)$$

Using equation 3.1, it is easily shown that:

$$N_{sph} = \frac{A_p^3}{36\pi V_p^2} \quad (3.5)$$

A sphere by definition will always have an N_{sph} value equal to unity.

For the large ribosomal subunit, at very small probe radius, this number should be very large; there are almost 150,000 atoms or VDW spheres (including hydrogens) in the structure. On the other hand, when the probe radius is large, the number of identical spheres required should approach unity. Figure 3.6B shows the dependence of N_{sph} on probe radius as determined by surface roughness. It is clear from Figure 3.6B (green line), that the roughness of the excluded surface of the large subunit falls very rapidly with probe size, and has been largely eliminated by the time the probe radius reaches 10 Å. For the VDW surface (*i.e.* a probe radius of zero), 7,482 effective spheres (Table 3.1) are required because of the large amount of surface area relative to the volume. This N_{sph} value for the VDW surface is much less than the number of atoms in the structure because each atomic sphere has a large overlap with its bonded neighbors.

Sphericity

Surface roughness is also an issue of interest to geologists. While researching sand and dirt particles, Wadell (1932) defined a parameter he called sphericity, Ψ , as the surface area of a sphere of the same volume as some particle divided by the actual surface area of that particle. It is easy to derive a formula for Ψ , first the surface area of the sphere, A_s is written in terms of the volume of the particle, V_p :

$$A_s^3 = (4\pi r^2)^3 = 4^3 \pi^3 r^6 = 4\pi \cdot 3^2 \left(\frac{4^2 \pi^2}{3^2} r^6 \right) = 36\pi \left(\frac{4\pi}{3} r^3 \right)^2 = 36\pi V_p^2$$

Therefore the Wadell sphericity, Ψ is:

$$\Psi = \frac{A_s}{A_p} = \frac{(36\pi V_p^2)^{\frac{1}{3}}}{A_p} = \frac{\pi^{\frac{1}{3}} (6V_p)^{\frac{2}{3}}}{A_p} \quad (3.6)$$

It is now relatively obvious that Ψ is directly related to N_{sph} :

$$N_{sph} = \left(\frac{1}{\Psi} \right)^3 \iff \Psi = \left(\frac{1}{N_{sph}} \right)^{\frac{1}{3}} \quad (3.7)$$

Just as in the number of effective spheres, N_{sph} , a perfect sphere has a value of one. No particle can have $\Psi > 1$ or $N_{sph} < 1$, because the sphere is the minimal surface for any given volume. Second, for a particle of a fixed volume as the surface area goes to infinity or for a fixed surface area as the volume goes to zero (*e.g.* a geometric plane), $\Psi \rightarrow 0$. Thus:

$$0 < \Psi \leq 1 \quad (3.8)$$

This limited range of Ψ makes it easier to work with than the large range of N_{sph} .

The statistical properties of the shell surface show that the surface is relatively smooth. As the probe radius increases, the sphericity goes from 0.227 at 3 Å probe, 0.609 at 10 Å probe, 0.906 at 100 Å probe, and 0.946 at the infinite probe almost approaching the ideal sphere. As Table 3.1 shows, the 10 Å shell surface is less spherical than either a tetrahedron or a cone. A cone may not

look more spherical than the 10 Å probe surface by eye, but it has a higher sphericity because it does not contain the numerous surface invaginations as the ribosomal surface. While the 100 Å probe surface may be much smoother than my choice for the shell surface, it is not ideal, because as previously mentioned, it includes large amounts of bulk solvent.

3.2.4 Volume Distribution of the Ribosome

Now that a limiting surface has been defined for the ribosome (as the 10 Å probe shell surface), the way its interior volume is filled can be analyzed. The ribosome is a ribonucleoprotein and it is a simple matter to examine its RNA and protein parts separately *in silico*. Several interesting architectural aspects, such as how each component fills the space of the ribosome can be investigated this way. In addition, the volumes of the individual components can provide a basis to calculate partial specific volumes and they in turn can be compared to experimentally measured values. All the following analyses will be based upon the volumes defined before (see page 50) which are presented in Table 3.2.

Convex hull volume

The convex hull volume of the large ribosomal subunit is only slightly larger than the convex hull volume of either its protein or RNA components (Table 3.2). The similarity of the convex hull volume of the RNA part of the large subunit to that of the whole particle is not surprising. Pictures comparing the RNA component of the large ribosomal subunit with the whole subunit give the impression that they are the same size and shape (Ban *et al.*, 2000). It is less obvious from such pictures that the convex hull of the protein part of the large subunit should be similar. However, since the proteins of the large subunit tend to decorate its periphery, including its lateral protuberances, and despite the relatively protein-poor surface of the subunit interface, the protein convex hull volume is about the same as that of the whole particle.

Platonic solids	Image	Volume	Area	N_{sph}	Sphericity
tetrahedron (4 faces)		$\frac{\sqrt{2}}{12} s^3$	$\sqrt{3} s^2$	3.308	0.671
hexahedron (cube, 6 faces)		s^3	$6 s^2$	1.910	0.806
octahedron (8 faces)		$\frac{1}{3} \sqrt{2} s^3$	$2 \sqrt{3} s^2$	1.654	0.846
dodecahedron (12 faces)		$\frac{1}{4} (15 + 7\sqrt{5}) s^3$	$3\sqrt{25 + 10\sqrt{5}} s^2$	1.104	0.968
icosahedron (20 faces)		$\frac{5}{12} (3 + \sqrt{5}) s^3$	$5\sqrt{3} s^2$	1.207	0.939

Rounded shapes	Image	Volume	Area	N_{sph}	Sphericity
ideal cone $(h = 2\sqrt{2}r)$		$\frac{1}{3}\pi r^2 h$ $\rightarrow \frac{2\sqrt{2}}{3}\pi r^3$	$\pi r(r + \sqrt{r^2 + h^2})$ $\rightarrow 4\pi r^2$	2.000	0.794
hemisphere		$\frac{2}{3}\pi r^3$	$3\pi r^2$	1.688	0.840
ideal cylinder $(h = 2r)$		$\pi r^2 h$ $\rightarrow 2\pi r^3$	$2\pi r(r + h)$ $\rightarrow 6\pi r^2$	1.500	0.874
ideal torus $(R = r)$		$2\pi^2 R r^2$ $\rightarrow 2\pi^2 r^3$	$4\pi^2 R r$ $\rightarrow 4\pi^2 r^2$	1.396	0.894
the Earth		$1.08 \cdot 10^{12} \text{ km}^3$	$5.10 \cdot 10^8 \text{ km}^2$	$1 + 5.93 \cdot 10^{-6}$	$1 - 1.98 \cdot 10^{-6}$
sphere (minimal surface)		$\frac{4}{3}\pi r^3$	$4\pi r^2$	1.000	1.000

Ribosomal surfaces	Image	Volume	Area	N_{sph}	Sphericity
van der Waals $(p = 0 \text{ \AA})$		1,086,420 \AA ³	999,577 \AA ²	7482	0.051
solvent-excluded $(p = 1.5 \text{ \AA})$		1,393,427 \AA ³	433,434 \AA ²	370.8	0.139
tunnel-excluded $(p = 3 \text{ \AA})$		1,662,860 \AA ³	299,374 \AA ²	85.80	0.227
shell $(p = 10 \text{ \AA})$		2,294,180 \AA ³	138,158 \AA ²	4.430	0.609
bigprobe $(p = 100 \text{ \AA})$		3,243,097 \AA ³	116,916 \AA ²	1.344	0.906
convex hull $(p \rightarrow \infty)$		3,707,784 \AA ³	122,509 \AA ²	1.183	0.946

Table 3.1: Sphericity of some common shapes and ribosomal surfaces. Value for Wadell (1932) Sphericity, Ψ and the number of effective spheres, N_{sph} for all platonic solids and various rounded shapes compared to the ribosomal values. The values for the sphere are exactly one.

	Complete Ribosome	rRNA only	rProtein only	Sum (RNA+Prot)	Overlap Volume
	<i>Mass (Da)</i>				
molecular mass	1,340,664 [†]	931,319	409,345	1,340,664 [†]	—
	<i>Exterior Volumes (Å³)</i>				
convex hull volume	3,707,784	3,166,785	3,282,611	6,449,396	2,741,611
shell volume	2,290,980	1,715,861	825,772	2,541,633	362,624
solvent-accessible volume	2,014,176	1,386,329	797,680	2,184,009	169,841
solvent-excluded volume	1,395,480	867,245	498,025	1,365,270	4,360
VDW volume	1,086,421	698,097	390,339	1,088,435	2,008
Voronoi volume ^a	1,374,540 [†]	879,465	495,075	1,374,540 [†]	—
	<i>Interior Volumes (Å³)</i>				
empty volume ^b	309,059	169,148	107,686	276,834	—
solvent volume ^c	895,500	848,616	—	—	—
fractional solvent volume ^d	39.1%	49.5%	—	—	—
cavity volume	14,641	2,100	0	2,100	—
	<i>Surface Areas (Å²)</i>				
solvent-accessible area	420,245	370,388	214,800	585,189	—
convex hull area	122,509	111,401	114,801	226,202	—
shell area	138,633	128,805	152,193	280,998	—
solvent-excluded area	434,727	340,701	187,095	527,796	—
VDW area	1,003,020	648,424	371,991	1,020,415	—

Table 3.2: Volumes of the large ribosomal subunit and its components. Coordinates from PDB entry 1JJ2 (Klein *et al.*, 2001) was used to calculate all values. The probe radii for the surface are: convex hull is $r \rightarrow \infty$, shell is $r = 10 \text{ \AA}$, solvent-excluded and solvent-accessible are $r = 1.5 \text{ \AA}$, and VDW is $r = 0$. ^aVoronoi volume is calculated based on average RNA atomic volumes reported in Chapter 2, page 36 and protein atomic volumes from Tsai *et al.* (2001). ^bThe empty volume is the solvent-excluded volume minus VDW volume. ^cThe solvent volume is the shell volume minus solvent-excluded volume. ^dThe fractional solvent volume (FSV) is the solvent-excluded volume divided by the shell volume. [†]values for complete ribosome and component sum are equal because of the additive nature of the quantity. Definitions for the volumes are on page 50.

Shell volumes

Comparisons of shell volumes provide a more accurate sense of how the volumes of the components of the large ribosome contribute to its total volume. The shell volume of the whole subunit is much larger than the shell volume of its RNA component, but only 9.9% smaller than the sum of the shell volumes of its protein and RNA parts taken separately (Table 3.2). Clearly, the protein portion of the ribosome does more than fill chinks in the particle's RNA surface.

Two geometric effects act in opposing directions and account for the failure of the sum of the component shell volumes of the particle to add up to the total for the whole particle. On the one hand, the shells of the protein and RNA parts of the structure overlap by a volume of 362,624 Å³. Thus when component shell volumes are summed, these overlapping volumes are counted twice. On the other hand, when the RNA and protein portions of the subunit, are “reassembled” *in silico* to make an intact particle, new invaginations or gaps are “created” in interface regions where the probe used to define the shell surface, cannot enter. Since the sum of the RNA and protein shell volumes exceeds the shell volume of the whole particle by only 250,652 Å³, the amount of new “gap” volume inside the shell of the assembled structure must be 111,971 Å³. These “gap” volumes are included within the shell of the intact assembly, and their presence makes the volume of the intact assembly larger than the sum of the volumes of its components. For shell volumes, the overlap effect is obviously more important than the gap-filling effect, and the sum of component volumes exceeds the volume of the intact assembly.

Solvent-excluded and solvent-accessible volumes

Solvent-excluded surface volumes add better than the convex hull or shell surface volumes. The overlap volume for the solvent-excluded surfaces of the RNA and protein parts of the subunit is small, only 4,360 Å³. This is expected because the smaller size of the probe used to define these surfaces reduces the amount of “padding” they include. In this instance, the sum of the solvent-excluded volumes of the components is actually less than solvent-excluded volume of the complete structure by 30,210 Å³. Therefore, the amount of new solvent-excluded “gap” volume “created” by the assembly of the structure is about 34,570 Å³.

From the way solvent-accessible volumes are defined (see Figure 3.1, page 51), solvent-accessible “gap” volumes do not exist when the components are assembled. It follows that the overlap volume should equal the difference between the sum of the RNA and protein volumes and the volume of the assembled structure, and it does within the error of the grid size used: the overlap volume is 169,841 Å³, and the difference volume is 169,833 Å³.

Surface areas

The surface areas ascribed to ribosomal components by the different methods behave in a simpler manner. The sum of the surface areas of the components of the subunit is larger than the surface area of the intact subunit for all surfaces, as expected (Table 3.2). The reduction in solvent-accessible surface area that accompanies assembly is physically meaningful because the binding energy of the components of a macromolecular complex is known to be roughly proportional to the amount of solvent-accessible surface area buried when they associate (Chothia *et al.*, 1976). When the large ribosomal subunit is assembled *in silico*, the decrease in the solvent-accessible surface area is substantial both in absolute terms (164,944 Å³) and as a percentage of the whole (32%), as was pointed out earlier using a 1.7 Å probe (Klein *et al.*, 2004).

Partial specific volumes

In Chapter 2, I calculated the partial specific volumes (PSV) for RNA molecules using the Voronoi polyhedra. The average PSV for RNA was found to be 0.569 mL/g, which was denser than the average value for proteins (0.728 mL/g; see page 37). This goes against the intuition that proteins have these tightly formed hydrophobic cores, while RNA is charged and cannot pack together as tightly.

Using the solvent-excluded volumes computed for the large ribosomal subunit, I can also obtain estimates of partial specific volumes to compare with those that have been measured experimentally. Table 3.3 shows the solvent-excluded volumes, molecular masses and the partial specific volumes they imply for the large ribosomal subunit and its components. These PSV values compare well with the Voronoi values from Chapter 2 largely because the component Voronoi volumes of the

particle are almost identical to the solvent-excluded volumes (Table 3.2). The PSV has been measured experimentally by van Holde & Hill (1974) for the *E. coli* 50S particle. This experimental value (0.592 mL/g) compares reasonably well with the value implied by my solvent-excluded surface (0.627 mL/g) considering no effort was made to address the net charge of the molecule or electrostriction effects of the solvent on this calculation.

3.2.5 Characterizing the Solvent in the Ribosome

Two probe method to determine amount of solvent

As evident from Table 3.2, the solvent-excluded volume of the large ribosomal subunit is far less than the volume inside its shell. In fact, the difference between these two surfaces is equal to the solvent volume. So, using these defined surfaces, the structure of the solvent-filled volume can be directly extracted from the structure of any macromolecule.

The solvent of a macromolecule is defined using a two-step process. In the first step, the shell volume delimits the entire molecule using 10 Å probe-excluded surface that makes the interior and exterior of the molecule well defined (Figure 3.7A). In the second step, the solvent-excluded volume is calculated using a rolling probe of 1.5 Å, which defines the surface associated with the solvent (Figure 3.7B). Finally, the solvent-excluded volume (Figure 3.7B) is subtracted from the shell volume (Figure 3.7A) resulting in the solvent volume, shown in Figure 3.7C (blue).

Solvent-filled voids are abundant in the large ribosomal subunit

Contrary to the impression conveyed by many published images of the large subunit (for example, see Figure 1.1, page 3), the ribosome is not a solid structure but rather full of volumes large enough to accommodate solvent molecules. Interestingly, when a slice of the subunit's interior is viewed, as in Figure 3.8A, the view is filled with large solvent-filled veins. In fact, the solvent volume fills 39% of the entire shell volume. The most interesting aspect of the slices shown in Figure 3.8A is that the large solvent channels (blue) exist within the RNA (gray), while the proteins (yellow) have no solvent channels running through them. In addition, inside virtually every solvent volume seen in the slice, there is at least one ordered solvent molecule visible crystallographically (Figure 3.8B).

	<i>Molecular Mass</i>	<i>Solvent-Excluded</i>	<i>Partial Specific Volume, \bar{V} (mL/g)</i>		
	(Da)	Volume (\AA^3)	Calculated	Experimental	Voronoi ^d
<i>Ribosome</i>	1,340,664	1,395,480	0.627	0.592 ^a	0.617
<i>RNA</i>	931,319	867,245	0.561	0.540 ^b	0.569
<i>Protein</i>	409,345	498,025	0.733	0.726 ^c	0.728

Table 3.3: Partial specific volumes of the ribosomal components. Experimental values from ^avan Holde & Hill (1974) for the *E. coli* 50S ribosome, ^bDurchschlag (1986) for polynucleotides, and ^cCohn & Edsall (1943) based on amino acid composition. ^dVoronoi partial specific volume is calculated based on average RNA atomic volumes reported in Chapter 2, page 36 and protein atomic volumes from Tsai *et al.* (2001).

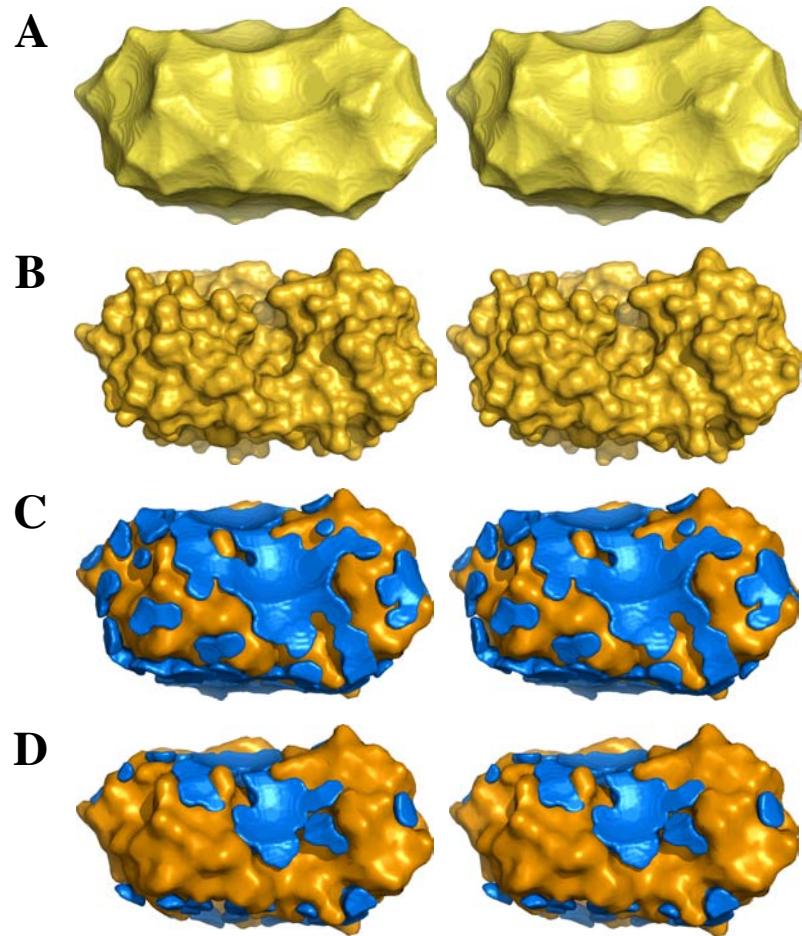


Figure 3.7: The two probe method to extract the solvent volume from macromolecules.
All figures show stereo images of different surfaces for the isomerase protein (PDB id: 1lzo, Parthasarathy *et al.*, 2002). (A) The shell volume of the protein (orange) using a 10 Å probe. (B) the solvent-excluded volume of the protein. (C) the solvent-excluded volume of the protein (orange) overlaid with the solvent volume (blue). (D) The 1 Å trimmed solvent volume (blue) obtained by shrinking the shell volume surface by 1 Å in all directions.

Not only are solvent channels abundant in the ribosome, but solvent channels are surprisingly connected. A solvent probe placed within any large channel inside the ribosome can reach 93% of all the solvent-filled spaces in the interior of the subunit without passing through the exterior. It could be argued in retrospect that high connectivity is likely for almost any structure that has high internal solvent content. Nonetheless, these channels are large enough for solvent molecules to diffuse through and are connected both to the exterior and to each other by several passages.

3.2.6 Analysis of Other Macromolecules

Fractional solvent volume

Any macromolecular structure can be analyzed using the system just described. As shown above, the fraction of the volume inside the shell that is solvent-filled or the fractional solvent volume (FSV) of the large ribosomal subunit is large; just over 39% (see Table 3.2). To understand what this number means it is useful to compare the ribosome to another macromolecular assembly that has an interior cavity: spinach rubisco (PDB id, 1RCX; Taylor & Andersson, 1997). When the rubisco structure is analyzed in the same way, 17% of the volume inside its shell is found to be accessible to solvent. Unlike the ribosome, the interior cavity of rubisco does not have highly reticulated solvent channels, but only one narrow channel down its center (Figure 3.8C, left). The difference between 39% and 17% is fairly large, consistent with the observation that proteins in general contain less solvent than RNA. But upon examination of Figure 3.8C (center), reveals the existence of a thin solvent layer between the exterior surface and the shell that clearly contributes to the amount of solvent assigned to macromolecules. In fact, the thin solvent layer within 3 Å of the shell surface accounts for 56.1% of the entire solvent inside the rubisco shell. When this exterior solvent is removed (Figure 3.8C, right), the solvent volume falls to only 7.6% of the volume inside the reduced shell. In contrast, when the same exterior solvent layer is removed on the large subunit, only a modest drop to 34% is seen because only 12.4% of the solvent inside its shell is located in the corresponding exterior layer. In Figure 3.7D, this same phenomenon is shown for a small isomerase protein where only the outermost 1 Å thick layer has been removed. For both rubisco and isomerase, even this little adjustment makes a large difference in the fractional solvent volume calculation. The

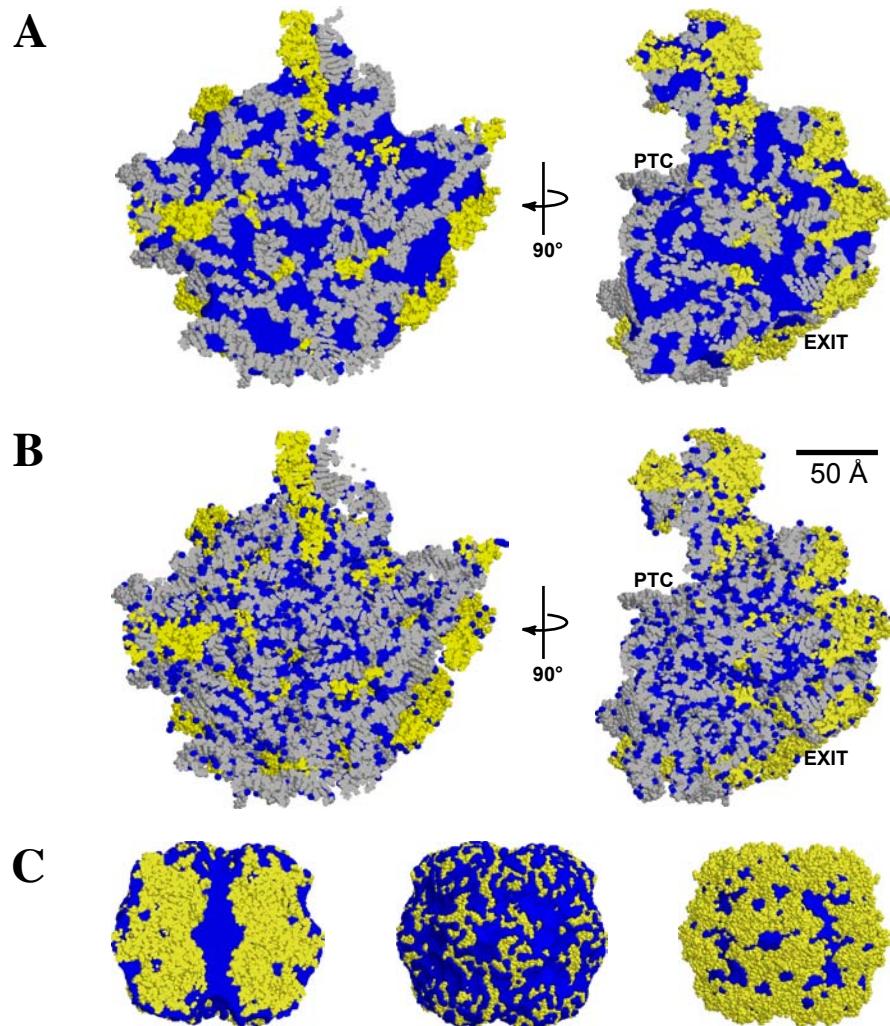


Figure 3.8: Slices through the interior of the ribosome and selected proteins showing interior solvent volumes. The images of the large subunit shown in (A) and (B) were obtained by cutting the particle into two pieces along a plane perpendicular to the line of view, and exposing its interior by removing the portion nearer to the viewer. In the left pair of images, the particle is oriented in the so-called crown view, and in the right pair of images, the particle was rotated by 90° about the vertical axis prior to being sliced. In all images, RNA is gray, protein is yellow, and solvent is blue. (A) The solvent volume as determined by a 1.5 Å probe, represented in blue, of a single slice through the large ribosomal subunit. (B) The locations of solvent molecules (small blue spheres) reported in the crystal structure of the large ribosomal subunit of *H. marismortui* (PDB id, 1JJ2; Klein *et al.*, 2001) are displayed that fall within the space visible in part (A) of this figure. (C) Images of spinach rubisco (PDB id, 1RCX; Thoden *et al.*, 1998) with the solvent volume as determined by a 1.5 Å probe shown in blue. A slice through its central cavity (left), the surface of the protein highlighting the surface solvent (center), the surface of the protein after the shell has been trimmed by 4 Å. All images are drawn to scale.

results obtained by stripping away the surface layer, 34% for the ribosome and 7.6% for rubisco, is more inline with the long-standing observation that the interiors of protein domains contain little or no solvent (Lee & Richards, 1971; Shrake & Rupley, 1973). Thus in order to obtain a more useful measure of the internal solvent for both proteins and RNA, this method needs to be modified so as to remove the surface layer of solvent.

Reduced fractional solvent volume

To further investigate the outer solvent layer seen in rubisco, I analyzed a set of 14 multi-domain protein complexes of about 150-700 kDa in size as well as 87 smaller single-domain proteins with unique fold architecture (Tsai *et al.*, 1999, 2001). First, I need to decide how much of the outer-layer needs to be removed to reduce the surface solvent effect. On one hand, the thicker the layer removed the less the bias caused by the surface solvent. On the other hand, for the single-domain proteins, if the layer removed is too thick, the remaining volume to be analyzed will become very small. Figure 3.9 shows the fraction solvent volume for each structural category changes as the thickness of the surface layer removed increases. Figure 3.9 can also be thought of as a plot of the radial distribution of solvent as distance from the surface in different types of macromolecules. The fractional solvent volume of the ribosome is only moderately reduced as the outer solvent layers are removed (Figure 3.9, black line). In fact, even after removing a 10 Å layer the fractional solvent volume is still above 35%. For both single and multi-domain proteins, however, the effect is less dramatic. On average, proteins reach their minimum value when their outer layer is trimmed by about 4 Å. Note that a few of the proteins analyzed contain a central cavity that is filled with solvent (*e.g.* rubisco) and as surface layers are removed the contribution of the central cavity causes the fractional solvent volume to actually increase, and that effect is visible in the overall average fractional solvent (Figure 3.9, right).

On the basis of these results, I define a new quantity, called the “reduced fractional solvent volume” (rFSV). The rFSV is the fractional volume inside the shell surface where a 4 Å outer layer is trimmed away that is accessible to solvent. The 4 Å cutoff is balance between removing as much surface solvent as possible without removing too much of the protein volume.

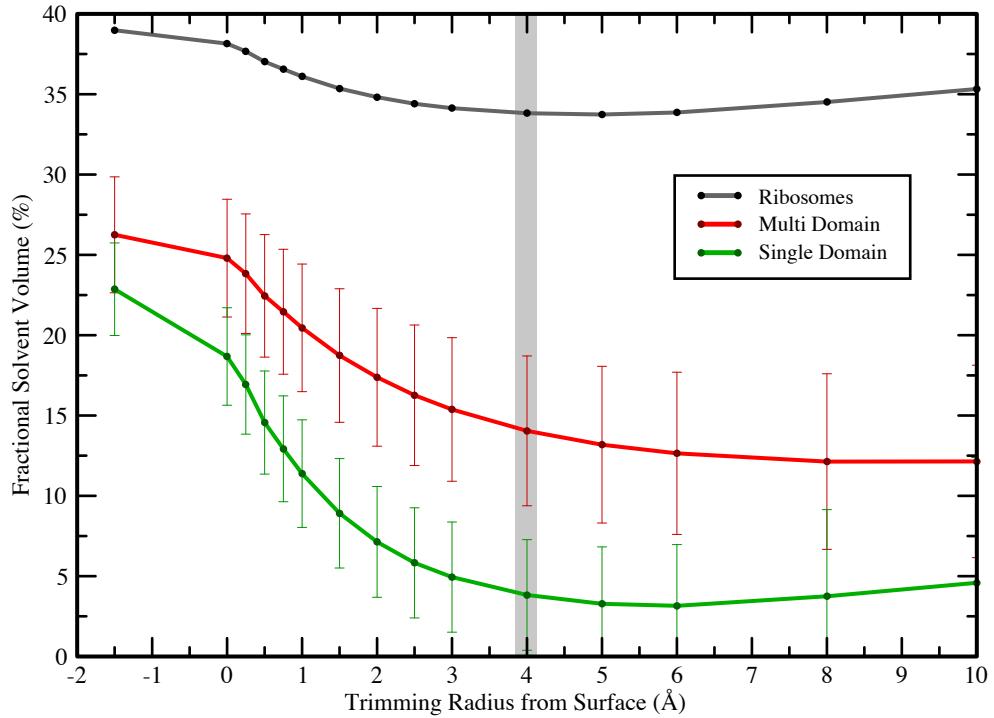


Figure 3.9: Fractional solvent volume as a function of trimming radius. The fractional solvent volume (FSV) is defined as the percentage of solvent in the macromolecule divided by the shell volume. A zero probe is defined as the normal case where the probe center is required to be contained within the shell for it to contribute to the solvent volume. A negative value of the probe radius means that any point of the probe within the shell that is not occupied by the solvent-excluded volume is considered solvent volume. The vertical gray bar represents the ideal solvent removal area.

Solvent content of macromolecules

Using the “reduced fractional solvent volume” (rFSV, defined above), I am now able to systematically address the amount of solvent inside macromolecules. As stated before the normal FSV of the ribosome is 39.0%, but for the rFSV, this drops to 33.8%. This is actually a general trend of RNA structures; other large RNAs including the 30S ribosomal subunit and group I intron both have rFSV values greater than 30% and ribonuclease P has a slightly smaller value of 21.5% due to its larger surface area and its pancake-like shape (Table 3.4). The smaller RNAs see a larger effect when their shell is reduced in size with rFSV values dropping to 9.2%, but they also lack the large-scale helix-helix packing interactions that make the interiors of larger RNAs so “wet.” For the set of single-domain proteins, the final rFSV values are $3.8 (\pm 3.4)\%$ when surface solvent is removed, a significant drop from $22.9 \pm 2.9\%$ (Figure 3.10A). In fact, seven of the proteins have no interior solvent at all and thirty have less than 2%. For multi-domain proteins, the drop is not as significant from $26.2 (\pm 3.6)\%$ to $14.0 (\pm 4.7)\%$, but these values are far less than the nucleotide-containing structures showing that protein are indeed relatively devoid of interior solvent. For even larger protein complexes, such as viruses and chaperones, the results are the same as those obtained with the multi-domain proteins (Table 3.4).

3.3 DISCUSSION

3.3.1 Universality of the 10 Å Surface

While 10 Å excluded surfaces are good for defining the exterior shell for the large ribosomal subunit, it does not perform as well for small proteins. Of the fourteen multi-domain proteins studied, five had no cavities larger than 2.5 Å and all but rubisco had no cavities larger than 8.0 Å. One can assume that single-domain proteins should have even smaller cavities and less of them than the multi-domain proteins. One characteristic observed for these proteins is that after about 3 Å all the cavities disappear and then around 6-7 Å larger scale cavities appear (Figure 3.11). From these results, it is immediately apparent that there exists no probe radius for determining shell surfaces that work for all macromolecules. The problem is not unique to my method, other methods for

Common Name	PDB id	Shell volume	# of atoms	FSV	rFSV
50S ribosome	1jj2	2,288,521	90,418	39.0	33.8
30S ribosome	1i94	1,288,594	44,393	47.9	44.0
RNase P	2a64	169,321	6,383	43.7	21.5
Group I ribozyme	1u6b	132,763	5,543	38.8	31.2
Yeast Phe tRNA	1ehz	36,049	1,652	30.6	9.32
Hammerhead ribozyme	1hmh	22,869	998	35.0	9.07
Nucleosome	1s32	328,468	12,341	39.1	29.5
Polymerase	1ig9	203,768	8,005	27.8	10.9
DNA-protein complex	1b3t	67,108	2,982	19.3	2.04
Simian 40 virus	1sva	26,599,534	958,980	25.0	10.4
Ringspot virus	1a6c	6,424,048	240,960	29.0	8.91
Chaperonin	1a6d	1,699,987	60,656	33.1	14.4
20S Proteasome	1ryp	1,262,080	49,676	26.2	10.8
Multi-domain proteins	14 [†]	596,303 (\pm 294,758)	24,562 (\pm 12,093)	26.2 (\pm 3.6)	14.0 (\pm 4.7)
Single-domain proteins	85 [‡]	41,515 (\pm 35,631)	1,832 (\pm 1,521)	22.9 (\pm 2.9)	3.83 (\pm 3.4)

Table 3.4: Solvent properties for several different macromolecules. Several different macromolecules grouped by those that contain RNA or DNA, only protein, and protein averages are shown. The fractional solvent volume (FSV) is defined as the solvent volume divided by the shell volume. Likewise, the reduced fractional solvent volume (rFSV) is defined as the reduced solvent volume divided by the reduced shell volume. All fractional values are shown as a percentage. For the multi-domain proteins[†], 14 in total, and single-domain proteins[‡], 85 in total, averages with standard deviations are provided. PDB files used are: 50S ribosome: 1JJ2 (Klein *et al.*, 2001), 30S ribosome: 1I94 (Pioletti *et al.*, 2001), RNase P: 2A64 (Kazantsev *et al.*, 2005), group I ribozyme: 1U6B (Adams *et al.*, 2004), yeast phe-tRNA: 1EHZ (Shi & Moore, 2000), hammerhead ribozyme: 1HMH (Pley *et al.*, 1994), nucleosome: 1S32 (Edayathumangalam *et al.*, 2004), polymerase: 1IG9 (Franklin *et al.*, 2001), DNA-protein complex: 1B3T (Bochkarev *et al.*, 1998), simian 40 virus: 1SVA (Stehle *et al.*, 1996), ringspot virus: 1A6C (Chandrasekar & Johnson, 1998), chaperonin: 1A6D (Ditzel *et al.*, 1998), and 20S proteasome: 1RYP (Groll *et al.*, 1997).

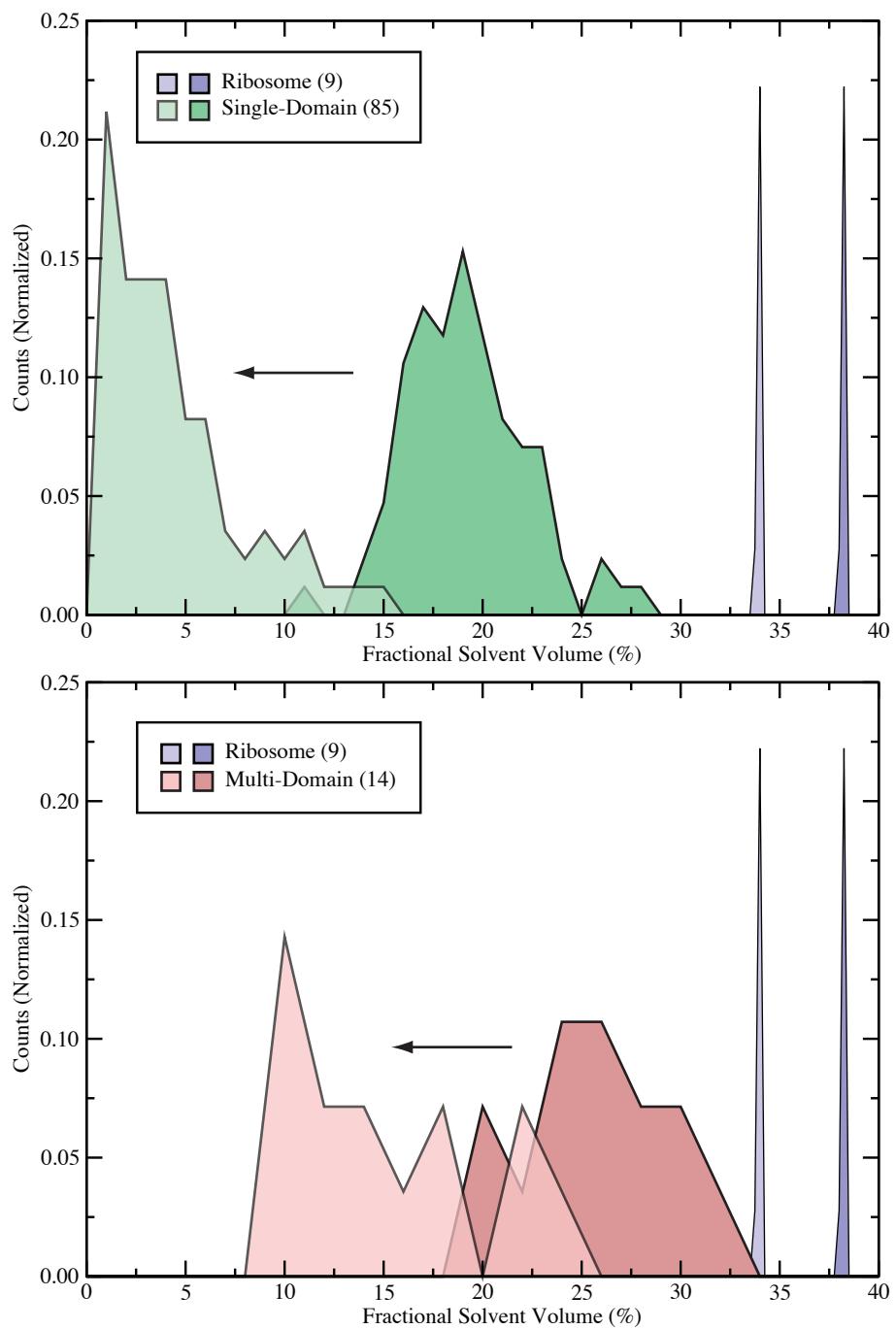


Figure 3.10: Histogram of the fractional solvent volume for proteins. (A) The normal (dark green) and reduced fractional solvent volume (rFSV, light green) for single-domain proteins. (B) The FSV (dark pink) and rFSV (light pink) for multi-domain proteins. The ribosome is shown as a large spike in both (A) and (B). (See page 82 for the list of protein structures.)

characterizing the solvent volumes of macromolecules have same problem of delimiting the exterior (see Appendix B). Thus, the method I have devised is good for sectioning off the exterior of a macromolecule, but the probe radius for the shell needs to be adjusted for each macromolecule.

3.3.2 RNA Molecules and Inter-helix Packing

This study shows that the interior of the ribosome includes a large network of interconnected channels and solvent, which occupies more than one-third of the volume inside the shell of the large subunit. Protein assemblies tend to be much drier in their interiors. There is little or no solvent within protein domains, and only a few of them have internal solvent volumes of substantial size, *e.g.* GroEL. It is easy to understand why the ribosome is so wet. Most of the RNA in the ribosome is helical, and no matter how helices ~ 20 Å in diameter are packed, the gaps between them are bound to be large enough to accommodate solvent. Furthermore, individual RNA helices do not efficiently fill the cylindrical volumes that enclose their structures because their grooves are large. Another reason RNAs pack inefficiently is their packed structures must accommodate the water molecules and counterions required to neutralize their backbone charges. Protein domains, on the other hand, are as densely packed in their interiors as crystals of small organic molecules (Richards, 1974). Other large RNAs also have large amounts of interior solvent, which indicates that the high solvent content of the interior of the ribosome is a general property of RNAs and ribonucleoprotein complexes

3.3.3 Proteins and Inter-domain Packing

Though on a smaller scale than in RNA structures, solvent volumes are seen in large proteins as well. While single-domain proteins have almost no interior solvent ($\sim 4\%$), multi-domain proteins usually have modest amounts ($\sim 14\%$). Presumably the difference in these values is account for by the looseness of the packing between different domains such as that shown in Figure 3.8C. Inter-domain packing is the cause of almost all of the internal solvent seen in the multi-domain complexes with the remaining solvent located with cavities.

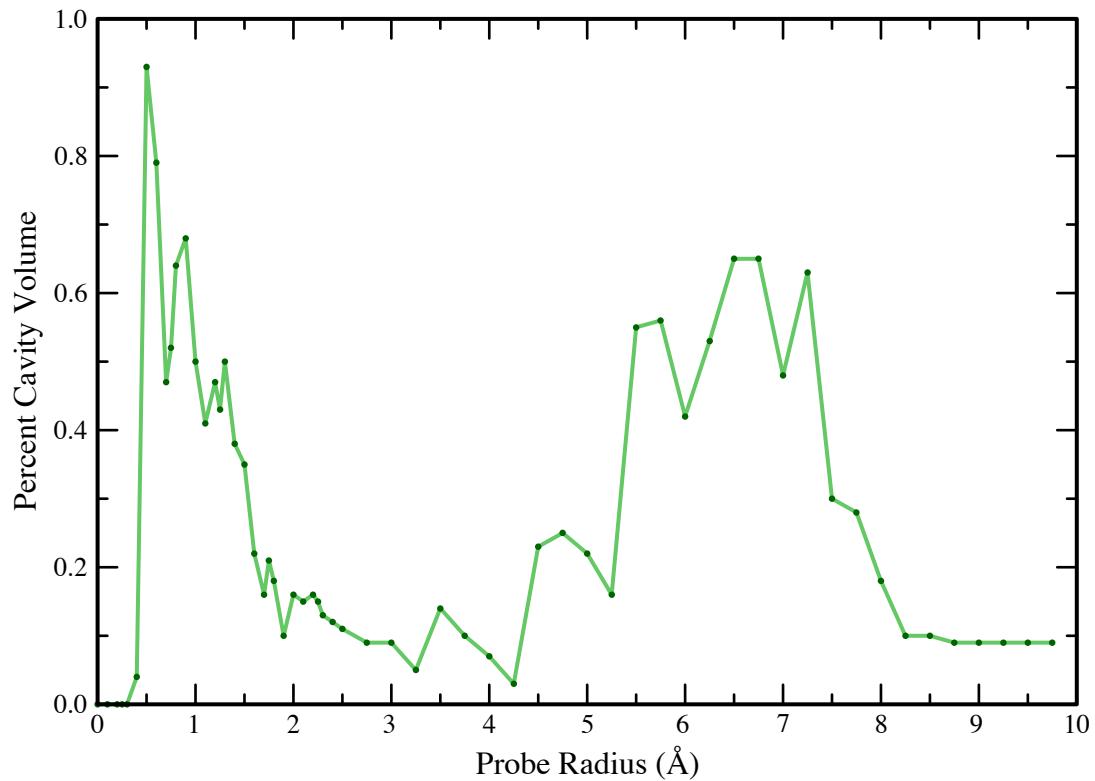


Figure 3.11: Percent cavity volume from multi-domain proteins as a function of probe radius. The percent cavity volume of fourteen different multi-domain proteins is plotted as a function of probe radius. Percent cavity volume is defined as percentage of the cavity volume divided by the shell volume. (See page 82 for the list of protein structures.)

3.3.4 Partial Specific Volumes

The finding that the interior of the ribosome is loosely packed might appear to be in conflict with the assertion that the atoms in RNA molecules have a lower partial specific volume from Chapter 2 and Voss & Gerstein (2005), but this is not the case. First the Voronoi volume of the ribosomal particle is approximately the same as the solvent-excluded volume, the volume of the particle excluding all solvent (Table 3.3). Because these volumes are equal it is reasonable to believe that the Voronoi method does not take into account the solvent within the shell of the particle. Second, the Voronoi volumes for RNA atoms reported in Chapter 2 are only volumes for RNA atoms that are “well-packed,” which means that they are surrounded closely on all sides by other atoms. Only 38.4% of the atoms in the RNA crystal structures analyzed are well-packed, even when crystal contacts are taken into account, and, interestingly, the fraction that is well-packed is about the same for ribosomes as it is for much smaller RNAs, *e.g.* tRNAs. Therefore, on average, the density of packing of RNA secondary structure elements in the ribosome is about the same as the density of packing of much smaller RNAs in crystals, *i.e.* not very tight.

3.4 METHODS

3.4.1 Atomic Coordinates and VDW Radii

All macromolecule files were taken from the Protein Data Bank (PDB; Berman *et al.*, 2000). For all PDB files, hydrogens were added using the program REDUCE with default settings (Word *et al.*, 1999). The van der Waals (VDW) radii used for atoms were taken from MSMS program based on stereochemistry of the RNA atoms (Sanner *et al.*, 1995; Sanner, 1995; Sanner *et al.*, 1996), and merged with coordinate data to create simple XYXR files that specify the coordinates and VDW radius of every atom in a molecule.

Ribosomes and RNA structures

1JJ2 was the main PDB file used for the *H. marismortui* large ribosomal subunit (Klein *et al.*, 2001). Additionally, eight different PDB files of the *H. marismortui* 50S structure were assessed: 1K8A,

1K9M, 1KD1, 1M1K (Hansen *et al.*, 2002), 1K73, 1KC8, 1N8R, and 1NJI (Hansen *et al.*, 2003).

These files were used to determine average values of solvent volume.

For the group I intron the 1U6B (Adams *et al.*, 2004) was the PDB file used. For the 30S subunit, PDB files 1I94 (Pioletti *et al.*, 2001), 1IBL (Ogle *et al.*, 2001), 1N32 (Ogle *et al.*, 2002), and 1XMQ (Murphy *et al.*, 2004) were used, but all four gave the same results. The nucleosome was taken from PDB file 1S32 (Edayathumangalam *et al.*, 2004). The RNase P complex was PDB id, 2A64 (Kazantsev *et al.*, 2005).

Protein structures

Single-domain proteins were taken from a previously defined set of 87 structures selected for a broad representation of different protein environments (Tsai *et al.*, 1999, 2001).

The fourteen multi-domain proteins (PDB id: 1A9X, Thoden *et al.*, 1998, 1EPW Swaminathan & Eswaramoorthy, 2000, 1FO4 Enroth *et al.*, 2000, 1GTE Dobritsch *et al.*, 2002, 1JZ8 Juers *et al.*, 2001, 1K83 Bushnell *et al.*, 2002, 1KEK Chabriere *et al.*, 2001, 1KQF Jormakka *et al.*, 2002, 1LSH Thompson & Banaszak, 2002, 1MUK Tao *et al.*, 2002, 1N2C Schindelin *et al.*, 1997, 1OFD van den Heuvel *et al.*, 2003, 1RCX Taylor & Andersson, 1997, and 2GLS Yamashita *et al.*, 1989) were selected by hand from the Protein Data Bank (PDB; Berman *et al.*, 2000). They were chosen because their protein chains are large and they have minimal symmetry.

3.4.2 External Programs

A comparison of other programs that analyze solvent of macromolecules is covered in Appendix B. Programs used for this chapter are listed below.

Convex hulls

Convex hulls were calculated using the QHULL program (Barber *et al.*, 1996). Since QHULL calculates the convex hulls of sets of points rather than sets of spheres, a perl script was created to read XYZR files, and generate files in a format acceptable to QHULL in which every atom is

represented by 200 points evenly distributed over the surface of its van der Waals sphere (Weisstein, 1999).

Cavity volumes

For Voss *et al.* (2006), cavity volumes were determined using VOIDOO (Kleywegt *et al.*, 2001; Kleywegt & Jones, 1994), but for Figure 3.3, the 3v program (Appendix F) written by me was modified and used. Cavities for the ribosome were calculated using a cubic grid having linear dimensions of 0.25 Å.

3.4.3 Rolling Probe Method

The rolling probe approach to determining volumes of the ribosome

Initially, solvent-excluded surface volumes were calculated using MSMS version 2.5.3 (Sanner *et al.*, 1995; Sanner, 1995; Sanner *et al.*, 1996). (Version 2.5.5, which became available subsequently, does not work with structures containing as many atoms as the ribosome.) However, MSMS is designed for speed at the cost of accuracy, and did not always produce consistent results. For this reason I wrote a program of our own for performing these calculations (Appendix F).

This program determines the solvent-accessible volume of macromolecules by reading their files into memory and assigning to each atom a radius that is its VDW radius plus the radius of the probe (Figure 3.1). The coordinates of the molecule are then superimposed on a finely spaced, cubic grid. The solvent-accessible volume of the molecule is the set of all grid points (voxels) that fall within the augmented atomic sphere of any atom. Next, the set of all “empty” grid points that are adjacent to points within the solvent-accessible volume of the molecule is determined. This set defines the solvent-accessible surface. The solvent-excluded volume is obtained by removing from the set of solvent-accessible points all points that are within the one probe radius from the solvent-accessible points on the surface (Figure 3.1, page 51). The volume is determined multiplying the total number of occupied grid points by the individual grid point volume. Not surprisingly, this algorithm works best if the grid used is very fine. For all calculations reported above, the linear dimensions of the grid were 0.16 Å to 0.22 Å, *i.e.* 244 to 94 grid points per Å³. Figures showing

such surfaces were obtained by writing their grids in EZD electron density file format, converting that file into a CCP4 file using MAPMAN (Kleywegt *et al.*, 2001; Kleywegt & Jones, 1993), and finally rendering them using the PyMOL graphics program (DeLano, 2002). For figures, a 0.3 Å (37 grid points per Å³) grids were calculated, which were then shrunk to 0.6 Å to reduce PyMOL rendering time and prevent crashing.

Extraction of interior volumes

The first step in the determination of the solvent volume was the calculation of the shell volume of the 50S subunit, determined using a 10 Å radius probe, on a fine grid. Second, the accessible surface was determined on the same grid by the rolling probe method (Figure 3.1, page 51) using a smaller probe. Subsequently, the accessible volume grid of the smaller probe was subtracted from the shell volume grid to yield the set of interior grid points that are accessible to the smaller probe center. In order to obtain the excluded volume, the accessible volume subset was convoluted with the probe sphere. Cavity volumes were obtained in the same way as the solvent volume, but all interior points that connect to the outer surface were removed leaving only the cavities.

Surface area calculations

Solvent-accessible and solvent-excluded surface calculations were done using the surface grids generated by the volume estimating software just described. The algorithm used was one devised for processing MRI brain data (Windreich *et al.*, 2003; Mullikin & Verbeek, 1993), which effectively assigns areas to surface voxels (grid points) based on estimates of the local curvature of the surface. The areas that should be assigned grid points in regions having different local curvatures was previously determined by maximum likelihood methods (Mullikin & Verbeek, 1993). Surface areas generated this way are surprisingly close to those obtained by more complicated algorithms of other programs, MSMS (Sanner *et al.*, 1995; Sanner, 1995; Sanner *et al.*, 1996) and AREAIMOL (Collaborative Computational Project, Number 4, 1994).

Availability of software

The volume calculation programs and source code described here are intended to be applicable to similar problems and can be obtained online at <http://geometry.molmovdb.org/3v/> and Appendix F.

3.5 REFERENCES

- P. L. Adams, M. R. Stahley, A. B. Kosek, J. Wang, and S. A. Strobel (2004). “Crystal structure of a self-splicing group I intron with both exons”. *Nature*, **430** (6995): pp. 45–50.
- N. Ban, P. Nissen, J. Hansen, M. Capel, P. B. Moore, and T. A. Steitz (1999). “Placement of protein and RNA structures into a 5 Å-resolution map of the 50S ribosomal subunit”. *Nature*, **400** (6747): pp. 841–7.
- N. Ban, P. Nissen, J. Hansen, P. B. Moore, and T. A. Steitz (2000). “The complete atomic structure of the large ribosomal subunit at 2.4 Å resolution”. *Science*, **289** (5481): pp. 905–20.
- C. B. Barber, D. P. Dobkin, and H. T. Huhdanpaa (1996). “The Quickhull algorithm for convex hulls”. *ACM Trans. on Mathematical Software*, **22** (4): pp. 469–483.
- H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne (2000). “The Protein Data Bank”. *Nucleic Acids Res*, **28** (1): pp. 235–42.
- A. Bochkarev, E. Bochkareva, L. Frappier, and A. M. Edwards (1998). “The 2.2 Å structure of a permanganate-sensitive DNA site bound by the Epstein-Barr virus origin binding protein, EBNA1”. *J Mol Biol*, **284** (5): pp. 1273–1278.
- D. A. Bushnell, P. Cramer, and R. D. Kornberg (2002). “Structural basis of transcription: α -amanitin-RNA polymerase II cocrystal at 2.8 Å resolution”. *PNAS*, **99** (3): pp. 1218–22.
- E. Chabriere, X. Verneude, B. Guigliarelli, M. H. Charon, E. C. Hatchikian, and J. C. Fontecilla-Camps (2001). “Crystal structure of the free radical intermediate of pyruvate: Ferredoxin oxidoreductase”. *Science*, **294** (5551): pp. 2559–63.
- V. Chandrasekar and J. E. Johnson (1998). “The structure of tobacco ringspot virus: A link in the evolution of icosahedral capsids in the picornavirus superfamily”. *Structure*, **6** (2): pp. 157–171.
- C. Chothia, S. Wodak, and J. Janin (1976). “Role of subunit interfaces in the allosteric mechanism of hemoglobin”. *PNAS*, **73** (11): pp. 3793–7.
- E. J. Cohn and J. T. Edsall (1943). *Proteins, amino acids and peptides as ions and dipolar ions*. Reinhold Publishing, New York.
- Collaborative Computational Project, Number 4 (1994). “The CCP4 suite: Programs for protein crystallography”. *Acta Crystallogr D Biol Crystallogr*, **50** (Pt 5): pp. 760–3.
- W. L. DeLano (2002). “The PyMOL molecular graphics system”. URL <http://www.pymol.org>.
- L. Ditzel, J. Lowe, D. Stock, K. O. Stetter, H. Huber, R. Huber, and S. Steinbacher (1998). “Crystal structure of the thermosome, the archaeal chaperonin and homolog of CCT”. *Cell*, **93** (1): pp. 125–38.
- D. Dobritzsch, S. Ricagno, G. Schneider, K. D. Schnackerz, and Y. Lindqvist (2002). “Crystal structure of the productive ternary complex of dihydropyrimidine dehydrogenase with NADPH and 5-iodouracil. Implications for mechanism of inhibition and electron transfer”. *J Biol Chem*, **277** (15): pp. 13155–66.

- H. Durchschlag (1986). “Specific volumes of biological macromolecules and some other molecules of biological interest”. In H.-J. Hinz, ed., “Thermodynamic data for biochemistry and biotechnology.”, pp. 45–128. Springer, Berlin.
- R. S. Edayathumangalam, P. Weyermann, J. M. Gottesfeld, P. B. Dervan, and K. Luger (2004). “Molecular recognition of the nucleosomal *supergroove*”. *PNAS*, **101** (18): pp. 6864–9. <http://dx.doi.org/10.1073/pnas.0402283101>.
- C. Enroth, B. T. Eger, K. Okamoto, T. Nishino, T. Nishino, and E. F. Pai (2000). “Crystal structures of bovine milk xanthine dehydrogenase and xanthine oxidase: Structure-based mechanism of conversion”. *PNAS*, **97** (20): pp. 10723–8.
- M. C. Franklin, J. Wang, and T. A. Steitz (2001). “Structure of the replicating complex of a pol α family DNA polymerase”. *Cell*, **105** (5): pp. 657–667. [http://dx.doi.org/10.1016/S0092-8674\(01\)00367-1](http://dx.doi.org/10.1016/S0092-8674(01)00367-1).
- I. S. Gabashvili, S. T. Gregory, M. Valle, R. Grassucci, M. Worbs, M. C. Wahl, A. E. Dahlberg, and J. Frank (2001). “The polypeptide tunnel system in the ribosome and its gating in erythromycin resistance mutants of L4 and L22”. *Mol Cell*, **8** (1): pp. 181–8.
- M. Gerstein and F. M. Richards (2001). “Protein geometry: Distances, areas, and volumes”. In M. Rossmann and E. Arnold, eds., “International tables for crystallography”, volume F: Crystallography of biological macromolecules, pp. 531–539. Springer, Dordrecht.
- M. Groll, L. Ditzel, J. Lowe, D. Stock, M. Bochtler, H. D. Bartunik, and R. Huber (1997). “Structure of 20S proteasome from yeast at 2.4 Å resolution”. *Nature*, **386** (6624): pp. 463–471.
- J. L. Hansen, J. A. Ippolito, N. Ban, P. Nissen, P. B. Moore, and T. A. Steitz (2002). “The structures of four macrolide antibiotics bound to the large ribosomal subunit”. *Mol Cell*, **10** (1): pp. 117–28.
- J. L. Hansen, P. B. Moore, and T. A. Steitz (2003). “Structures of five antibiotics bound at the peptidyl transferase center of the large ribosomal subunit”. *J Mol Biol*, **330** (5): pp. 1061–75.
- M. Jormakka, S. Tornroth, B. Byrne, and S. Iwata (2002). “Molecular basis of proton motive force generation: Structure of formate dehydrogenase-N”. *Science*, **295** (5561): pp. 1863–8.
- D. H. Juers, T. D. Heightman, A. Vasella, J. D. McCarter, L. Mackenzie, S. G. Withers, and B. W. Matthews (2001). “A structural view of the action of *Escherichia coli* (lacZ) β -galactosidase”. *Biochemistry*, **40** (49): pp. 14781–94.
- A. V. Kazantsev, A. A. Krivenko, D. J. Harrington, S. R. Holbrook, P. D. Adams, and N. R. Pace (2005). “Crystal structure of a bacterial ribonuclease P RNA”. *PNAS*, **102** (38): pp. 13392–7.
- D. J. Klein, P. B. Moore, and T. A. Steitz (2004). “The roles of ribosomal proteins in the structure assembly, and evolution of the large ribosomal subunit”. *J Mol Biol*, **340** (1): pp. 141–77.
- D. J. Klein, T. M. Schmeing, P. B. Moore, and T. A. Steitz (2001). “The kink-turn: A new RNA secondary structure motif”. *Embo J*, **20** (15): pp. 4214–21. <http://dx.doi.org/10.1093/emboj/20.15.4214>.
- G. J. Kleywegt and T. A. Jones (1993). “MAPMAN”. URL <http://xray.bmc.uu.se/usf/rave.html>.
- G. J. Kleywegt and T. A. Jones (1994). “Detection, delineation, measurement and display of cavities in macromolecular structures”. *Acta Cryst D*, **50**: pp. 178–185.

- G. J. Kleywegt, J. Y. Zou, M. Kjeldgaard, and T. A. Jones (2001). “Around O”. In M. G. Rossman and E. Arnold, eds., “International tables for crystallography”, volume F: Crystallography of biological macromolecules, pp. 353–356, 366–367. Kluwer Academic Publishers, The Netherlands.
- R. A. Laskowski (1995). “SURFNET: A program for visualizing molecular-surfaces, cavities, and intermolecular interactions”. *Journal of Molecular Graphics*, **13** (5): pp. 323–330.
- B. Lee and F. M. Richards (1971). “The interpretation of protein structures: Estimation of static accessibility”. *J Mol Biol*, **55** (3): pp. 379–400. [http://dx.doi.org/10.1016/0022-2836\(71\)90324-X](http://dx.doi.org/10.1016/0022-2836(71)90324-X).
- J. C. Mullikin and P. W. Verbeek (1993). “Surface area estimation of digitized planes”. *Bioimaging*, **1** (1): pp. 6–16.
- F. V. t. Murphy, V. Ramakrishnan, A. Malkiewicz, and P. F. Agris (2004). “The role of modifications in codon discrimination by tRNA(Lys)UUU”. *Nat Struct Mol Biol*, **11** (12): pp. 1186–91.
- J. M. Ogle, D. E. Brodersen, W. M. Clemons Jr., M. J. Tarry, A. P. Carter, and V. Ramakrishnan (2001). “Recognition of cognate transfer RNA by the 30S ribosomal subunit”. *Science*, **292** (5518): pp. 897–902.
- J. M. Ogle, F. V. Murphy, M. J. Tarry, and V. Ramakrishnan (2002). “Selection of tRNA by the ribosome requires a transition from an open to a closed form”. *Cell*, **111** (5): pp. 721–32.
- S. Parthasarathy, G. Ravindra, H. Balaram, P. Balaram, and M. R. N. Murthy (2002). “Structure of the Plasmodium falciparum triosephosphate isomerase-phosphoglycolate complex in two crystal forms: Characterization of catalytic loop open and closed conformations in the ligand-bound state”. *Biochemistry*, **41** (44): pp. 13178–13188.
- M. Phillips, S. Levy, and T. Munzner (1993). “Geomview: An interactive geometry viewer.” *Notices of the Amer Math Soc*, **40** (8): pp. 985–988. URL <http://www.geomview.org/>.
- M. Pioletti, F. Schlunzen, J. Harms, R. Zarivach, M. Gluhmann, H. Avila, A. Bashan, H. Bartels, T. Auerbach, C. Jacobi, T. Hartsch, A. Yonath, and F. Franceschi (2001). “Crystal structures of complexes of the small ribosomal subunit with tetracycline, edeine and IF3”. *Embo J*, **20** (8): pp. 1829–39.
- H. W. Pley, K. M. Flaherty, and D. B. McKay (1994). “Three-dimensional structure of a hammerhead ribozyme”. *Nature*, **372** (6501): pp. 68–74.
- F. M. Richards (1974). “The interpretation of protein structures: Total volume, group volume distributions and packing density”. *J Mol Biol*, **82** (1): pp. 1–14. [http://dx.doi.org/10.1016/0022-2836\(74\)90570-1](http://dx.doi.org/10.1016/0022-2836(74)90570-1).
- F. M. Richards (1977). “Areas, volumes, packing and protein structure”. *Annu Rev Biophys Bioeng*, **6**: pp. 151–76. <http://dx.doi.org/10.1146/annurev.bb.06.060177.001055>.
- M. Sanner, A. J. Olson, and J. C. Spehner (1995). “Fast and robust computation of molecular surfaces.” In “Proc. 11th ACM Symp. Comp. Geom”, pp. C6–C7.
- M. F. Sanner (1995). “MSMS: Michel Sanner’s molecular surface”. URL http://www.scripps.edu/~mb/olson/people/sanner/html/msms_home.html.

- M. F. Sanner, A. J. Olson, and J. C. Spehner (1996). “Reduced surface: An efficient way to compute molecular surfaces”. *Biopolymers*, **38** (3): pp. 305–20.
- J. Sauter (1926). “Die Größenbestimmung der im gemischnebel von verbrennungskraftmaschinen vorhandenen brennstoffteilchen”. *VDI Forschungsarbeiten*, **279**: p. 72.
- H. Schindelin, C. Kisker, J. L. Schlessman, J. B. Howard, and D. C. Rees (1997). “Structure of ADP·AlF₄⁻–stabilized nitrogenase complex and its implications for signal transduction”. *Nature*, **387** (6631): pp. 370–6.
- H. Shi and P. B. Moore (2000). “The crystal structure of yeast phenylalanine tRNA at 1.93 Å resolution: A classic structure revisited”. *RNA*, **6** (8): pp. 1091–105.
- A. Shrake and J. A. Rupley (1973). “Environment and exposure to solvent of protein atoms. Lysozyme and insulin”. *J Mol Biol*, **79** (2). [http://dx.doi.org/10.1016/0022-2836\(73\)90011-9](http://dx.doi.org/10.1016/0022-2836(73)90011-9).
- T. Stehle, S. J. Gamblin, Y. Yan, and S. C. Harrison (1996). “The structure of simian virus 40 refined at 3.1 Å resolution”. *Structure*, **4** (2): pp. 165–182.
- S. Swaminathan and S. Eswaramoorthy (2000). “Structural analysis of the catalytic and binding sites of Clostridium botulinum neurotoxin B”. *Nat Struct Biol*, **7** (8): pp. 693–9.
- Y. Tao, D. L. Farsetta, M. L. Nibert, and S. C. Harrison (2002). “RNA synthesis in a cage–structural studies of reovirus polymerase lambda3”. *Cell*, **111** (5): pp. 733–45.
- T. C. Taylor and I. Andersson (1997). “The structure of the complex between rubisco and its natural substrate ribulose 1,5-bisphosphate”. *J Mol Biol*, **265** (4): pp. 432–44.
- J. B. Thoden, S. G. Miran, J. C. Phillips, A. J. Howard, F. M. Raushel, and H. M. Holden (1998). “Carbamoyl phosphate synthetase: Caught in the act of glutamine hydrolysis”. *Biochemistry*, **37** (25): pp. 8825–31.
- J. R. Thompson and L. J. Banaszak (2002). “Lipid-protein interactions in lipovitellin”. *Biochemistry*, **41** (30): pp. 9398–409.
- J. Tsai, R. Taylor, C. Chothia, and M. Gerstein (1999). “The packing density in proteins: Standard radii and volumes”. *J Mol Biol*, **290** (1): pp. 253–66.
- J. Tsai, N. R. Voss, and M. Gerstein (2001). “Determining the minimum number of types necessary to represent the sizes of protein atoms”. *Bioinformatics*, **17** (10): pp. 949–56.
- R. H. van den Heuvel, D. I. Svergun, M. V. Petoukhov, A. Coda, B. Curti, S. Ravasio, M. A. Vanoni, and A. Mattevi (2003). “The active conformation of glutamate synthase and its binding to ferredoxin”. *J Mol Biol*, **330** (1): pp. 113–28.
- K. E. van Holde and W. E. Hill (1974). “General physical properties of ribosomes”. In M. Nomura, A. Tissieres, and P. Lengyel, eds., “Ribosomes.”, pp. 53–91. Cold Spring Harbor Laboratory, Cold Spring Harbor, NY.
- N. R. Voss and M. Gerstein (2005). “Calculation of standard atomic volumes for RNA and comparison with proteins: RNA is packed more tightly”. *J Mol Biol*, **346** (2): pp. 477–492. <http://dx.doi.org/10.1016/j.jmb.2004.11.072>.

- N. R. Voss, M. Gerstein, T. A. Steitz, and P. B. Moore (2006). “The geometry of the ribosomal polypeptide exit tunnel”. *J Mol Biol*, **360** (4): pp. 893–906. <http://dx.doi.org/10.1016/j.jmb.2006.05.023>.
- H. Wadell (1932). “Volume, shape and roundness of rock particles.” *J Geology*, **40**: pp. 443–51.
- E. W. Weisstein (1999). “Sphere point picking”. In “MathWorld: A Wolfram web resource.”, CRC Press LLC.
- G. Windreich, N. Kiryati, and G. Lohmann (2003). “Voxel-based surface area estimation: From theory to practice”. *Pattern Recognition*, **36** (11): pp. 2531–2541.
- J. M. Word, S. C. Lovell, J. S. Richardson, and D. C. Richardson (1999). “Asparagine and glutamine: Using hydrogen atom contacts in the choice of side-chain amide orientation”. *J Mol Biol*, **285** (4): pp. 1735–47.
- M. M. Yamashita, R. J. Almassy, C. A. Janson, D. Cascio, and D. Eisenberg (1989). “Refined atomic model of glutamine synthetase at 3.5 Å resolution”. *J Biol Chem*, **264** (30): pp. 17681–90.

Chapter 4

Characterization of the Ribosomal Exit Tunnel

4.1 INTRODUCTION

The peptidyl transferase center (PTC) is the site where peptide bond formation occurs in the ribosome. It is part of the ribosome's large subunit, and is located in the middle of the subunit face that interacts with the small ribosomal subunit (Frank *et al.*, 1995). In 1982, it was reported that nascent proteins first become accessible to antibodies on the side of the ribosome opposite to the PTC and subunit interface (Bernabeu & Lake, 1982). This surprising observation suggested the large subunit might contain an internal tunnel large enough for nascent peptides to pass through the middle of the ribosome from the PTC to the site where polypeptides emerge, $\sim 100 \text{ \AA}$ away. The existence of such a tunnel, the exit tunnel, was definitively proven by cryo-electron microscopy by Frank *et al.* (1995). It is an obvious feature in all the high resolution crystal structures of 70S ribosomes and large ribosomal subunits published to date (Ban *et al.*, 1999, 2000; Harms *et al.*, 2001; Yusupov *et al.*, 2001).

The physiological properties of the tunnel are not well understood. On the one hand, it is clear that nascent proteins having specific sequences can bind so tightly to the wall of the tunnel that protein synthesis is inhibited (Nakatogawa & Ito, 2002; Nakatogawa *et al.*, 2004; Tenson & Ehrenberg, 2002; Gong & Yanofsky, 2002). On the other hand, it has been long believed that

small macrolides like erythromycin inhibit protein synthesis by blocking the passage of nascent polypeptides down the tunnel sterically because they bind to a specific site on the tunnel wall (Agmon *et al.*, 2003). However, they bind in an indentation in the side of the tunnel where they do not completely obstruct the passage of proteins. Thus, these antibiotics probably inhibit protein synthesis by a mechanism more than just steric obstruction of the tunnel. However, it is not known whether interactions between nascent polypeptides and the tunnel wall play an active role in protein synthesis, *e.g.* by helping determine which nascent proteins will be secreted (Agmon *et al.*, 2003). In this regard, it has been suggested that ribosomal protein L22 might regulate protein synthesis because it has a globular domain that forms part of the rear surface where the translocon binds, and also contains a β -hairpin that stretches over two-thirds of the tunnel wall (Ban *et al.*, 2000; Baram *et al.*, 2005; Mitra *et al.*, 2005). This proposal has been called into question by the finding that cells containing ribosomes from which the loop sequence of L22 has been deleted are still viable (Zengel *et al.*, 2003).

Several other proposals have been made about tunnel functions that have geometric implications. In some electron microscopic reconstructions of the ribosome the tunnel appears to branch nears its exit end (Gabashvili *et al.*, 2001). This observation has led to the suggestion that nascent peptides might leave the ribosome by two different routes, one used by membrane proteins and the other by cytoplasmic proteins. It has also been suggested that nascent polypeptides fold at the tertiary level while traversing the tunnel (Gilbert *et al.*, 2004), a proposal that could relate to data indicating that 23S rRNA has chaperone activity (Pal *et al.*, 1999). Finally, the ribosome-translocon complex promotes the passage of nascent proteins across membranes that have electrochemical potential gradients across them, but protein translocation is not accompanied by passive ion flow. It has been suggested that the ribosomal exit tunnel itself must be part of the “seal” that makes the ribosome-translocon complex ion-tight (Liao *et al.*, 1997).

Using the methods presented in Chapter 3, a geometric analysis of the tunnel in the large ribosomal subunit of *Haloarcula marismortui* was undertaken to illuminate its role in protein synthesis. For objects the size of nascent polypeptides the tunnel is not branched. It is too small to allow nascent proteins to fold, and the ribosome does not contribute to the seal that prevents the passage of ions through the ribosome-translocon complex. In Chapter 3 it was shown that the solvent volume inside the ribosomal particle is a system of connected channels that permeate the entire particle. Here I show that a water molecule located inside the tunnel can diffuse to any other position without leaving the ribosome's interior.

4.2 RESULTS

4.2.1 Defining the Exit Tunnel

While the ribosome is a convex object like an apple, the tunnel is a concave object like the interior of a kitchen sink. So defining the tunnel is not such a straightforward issue. In order to define the volume contained by a sink, you must first decide where the sink ends and open space begins and the same applies to the tunnel. In Chapter 3, I used the rolling probe technique to define the exterior of the ribosome and a second probe to classify its solvent volumes. Here I modify this technique to arrive at a definition of the tunnel surface.

Delimiting the large ribosomal subunit

The tunnel was made discrete by surrounding the entire large ribosomal subunit with a surface called its “shell” (see Chapter 3, page 52 for detailed discussion). The essential properties of the shell of the large ribosomal subunit are: (1) that it enclose all the volumes inside the invaginations and other concavities in the surface of the large ribosomal subunit, (2) that the volume of exterior solvent captured within it be otherwise minimal, and (3) that its surface be connected. By delimiting the surface I was able to convert the tunnel from a concave object into a convex object.

The size and shape of the tunnel depends on probe size

Once the large ribosomal subunit had been enclosed in a satisfactory shell, the surface of its tunnel could be defined by placing a sphere of some radius inside its lumen *in silico* and allowing that sphere to roll around inside it. The excluded surfaces that emerge are determined in part by the atoms forming the wall of the tunnel, and in part by the surrounding shell.

Choosing the appropriate probe radius to define the tunnel wall is not a trivial question because the size and shape of the surface defined for the tunnel this way is extremely sensitive to probe radius. Figure 4.1 (green line) shows how the volume of the tunnel accessible to spherical probes depends on probe radius. For comparison, the dependence of the total accessible volume inside the shell of the ribosome on the radius of the probe sphere is also shown (Figure 4.1, red line). For probes having radii up to 2.5 Å, the volume accessible from the lumen of the tunnel is almost as large as the entire volume inside the shell accessible to a sphere of that radius. However, as the probe radius increases from 2.5 Å to 3.0 Å, the volume assigned to the tunnel plummets both in absolute terms and as a fraction of the total accessible volume inside the shell (Figure 4.1).

This dramatic fall in volume is associated with an equally dramatic change in tunnel morphology (Figure 4.2). Interestingly, when the tunnel surface is determined using a 2.7 Å radius probe, the object so defined is bifurcated at its distal end. However, a slight increase in probe radius eliminates the bifurcation without otherwise affecting the shape of the tunnel surface. When a probe of radius 3.0 Å is used, the tunnel surface that emerges corresponds to a single, cylindrical shape (see Figure 4.2, right on page 96).

The surface of the ribosome is very rough

The exterior surface of the ribosome (and all macromolecules) is very rough. It includes invaginations so large that it is possible for the center of a probe of moderate radius to travel virtually everywhere along the surface while staying inside of the shell (Figure 4.3B, bottom-left). So when

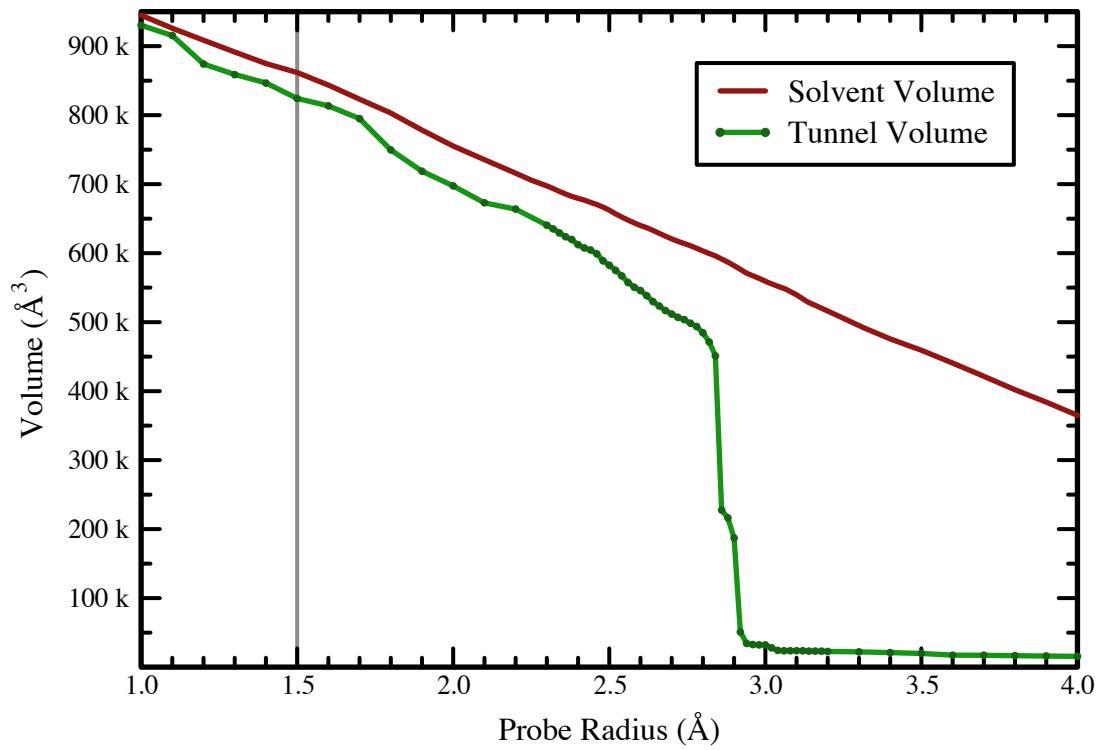


Figure 4.1: The dependence of volume accessible from the exit tunnel on probe radius. The total solvent volume (red) and the volume that is accessible to a probe placed in the tunnel lumen (green) is plotted as a function of probe radius. The gray line highlights the radius of a water-sized probe (1.5 \AA).

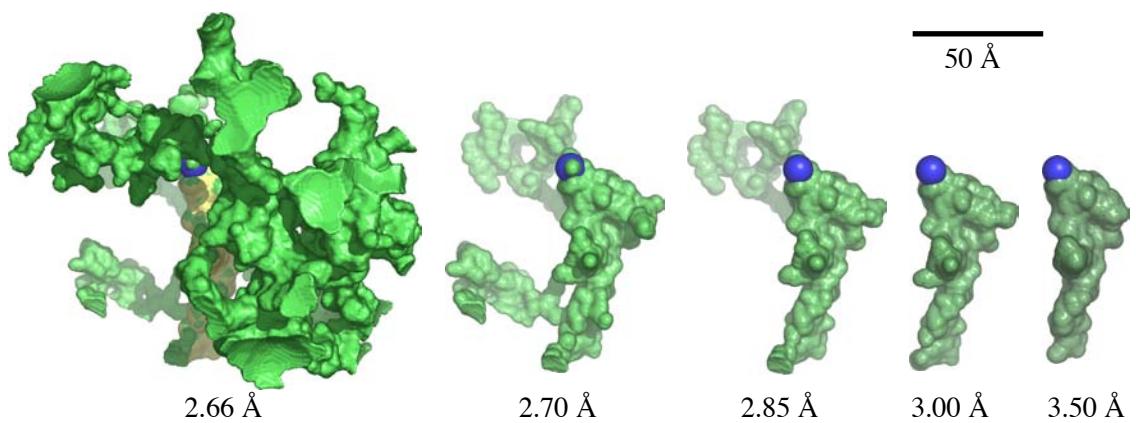


Figure 4.2: Excluded volumes of the ribosomal exit tunnel at different probe radii. The shapes of the excluded volume surfaces that are accessible to probes placed in the tunnel lumen. The location of the active site α -amino (the alpha-nitrogen of the first amino acid when connected to both A-site and P-site tRNAs in an intermediate state) is marked in all pictures with a large blue sphere. In the left-most figure (2.66 Å probe) the portion of surface that corresponds to the 3 Å shell is shown in yellow.

using a probe of moderate radius instead of getting the tunnel back, a large amount of superficial solvent is included, which makes the look as if it has a large “foot” (Figure 4.3B, left two images). To prevent this phenomenon and isolate the tunnel from the surface, I developed a technique in Chapter 3 called trimming. Trimming back the surface is equivalent to shrinking the shell by a defined radius from its outer surface. If the shell surface is trimmed by the same radius as the probe, then the entire volume of probe spheres whose centers lie inside the new surface will be contained inside the original shell, not just the probe center. A trimming radius equal to the probe radius is usually enough to keep superficial solvent from altering the shape of the tunnel. Figure 4.3 shows how a 2.78 Å and 2.94 Å tunnel appear with different trimming radii. For a 2.78 Å probe and no trimming the resulting collection of channels stretches throughout most of the ribosome. But when the 2.78 Å tunnel is trimmed by 6.5 Å, the exit tunnel that emerges is the main tunnel with only two small branches, a vast difference from the non-trimmed model. For a 2.94 Å probe when no trimming is used, the resulting exit tunnel has three large “feet” at the bottom of the tunnel because of the bloated shell. When the shell is then trimmed by 3.0 Å, the emerging volume is much more cylindrical and only retains one of the “feet” that occur without trimming. When 6.5 Å is trimmed from the shell, the exit tunnel is completely cylindrical and corresponds more closely to what most would consider to be the exit tunnel.

4.2.2 Properties of the Canonical Exit Tunnel

There are several factors that go into defining the tunnel (see above) and because they are somewhat arbitrary the size and shape of the tunnel cannot be uniquely defined. Secondly, its effective size and shape is determined by the shapes and sizes of the molecules that must pass through it. An extended polyalanine chain has lateral dimensions of roughly 6 Å (see Methods, page 119), and nascent polypeptides with bulkier side chains are larger than that. Hence for the purposes of analyzing the passage of polypeptides down the tunnel, it is appropriate to think of the tunnel surface as being the

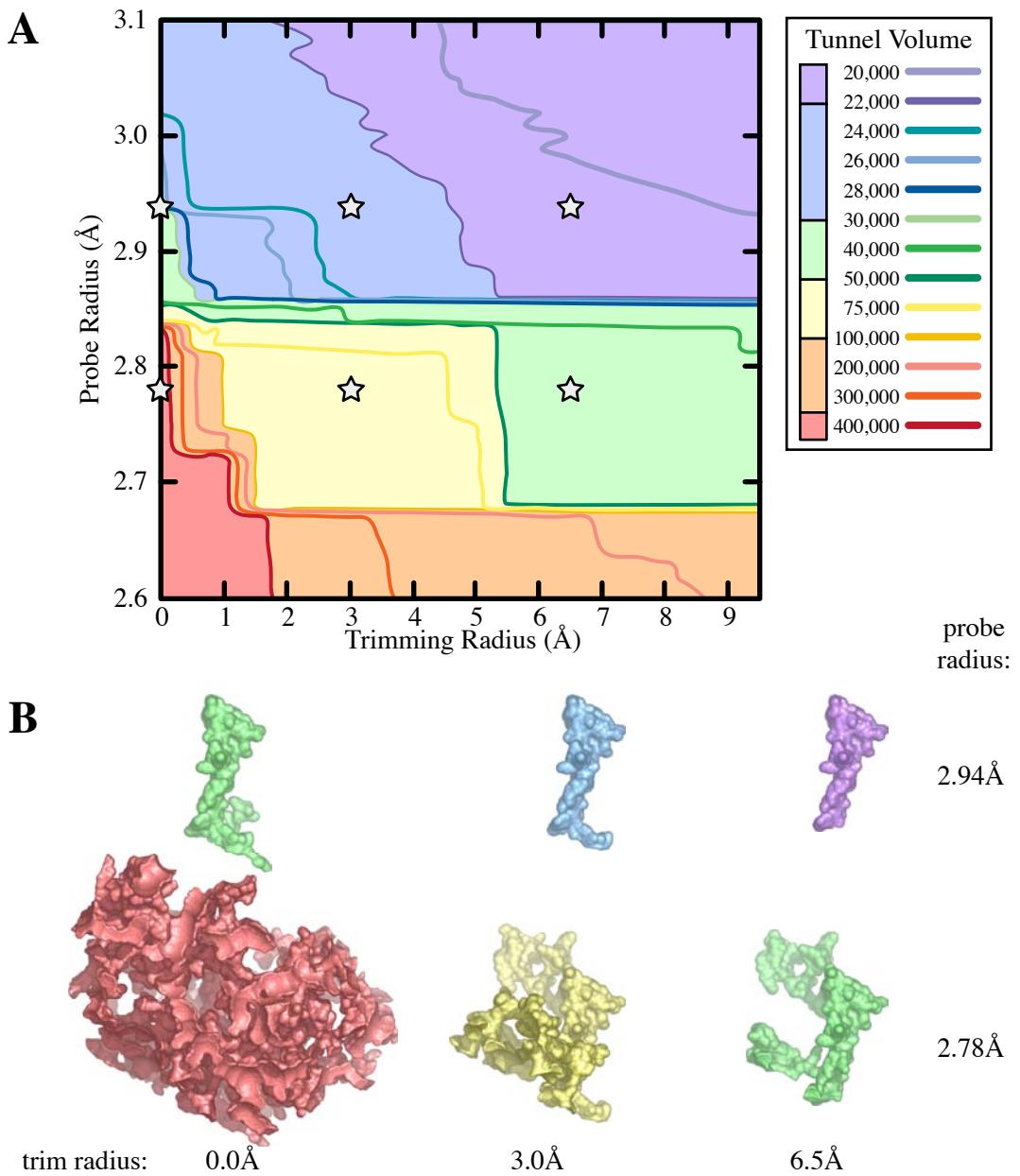


Figure 4.3: The effects of trimming back the shell surface on the exit tunnel volume and shape. (A) A two-dimensional contour plot of the exit tunnel volume as a function of trimming radius and probe radius. The colored lines and solid colors correspond to the volumes (in \AA^3) listed in the legend. A slice of the probe radius at trimming radius equal to zero is the same as Figure 4.1. The solid colors also coordinate to the corresponding images in part B. (B) Pictures of the tunnel for probe radii of 2.78 \AA and 2.94 \AA and trimming radius of 0 \AA , 3.0 \AA and 6.5 \AA which correspond to the gray stars (\star) in part (A).

surface defined using a sphere having a radius of 3.0 Å. A stereo view of that tunnel is provided in Figure 4.4A with some its biochemical landmarks indicated. The wall of the tunnel is RNA-rich; 82% of the atoms that contact its surface are RNA atoms leaving only 18% of them as protein atoms. The volume inside the tunnel surface is about 22,000 Å³, and the distance from the PTC to the distal end of the tunnel is about 80 Å (depending on where one decides its distal end is). Therefore, if the tunnel were a cylinder of uniform diameter, its diameter would be about 20 Å.

The width of the tunnel and tightness of the active site

Now that the exit tunnel has been defined as a three-dimensional collection of points, its width as a function of distance from the active site can be calculated by intersecting a plane with the tunnel surface and determining how much of the plane is contained within the tunnel surface. As Figure 4.5 (black line) shows, the cross-sectional area of the tunnel remains relatively constant over its length (other than minor fluctuations) with an effective radius of 9.10 ± 1.55 Å. It is evident that the tunnel is small around the active site of the peptidyl transferase center and then it opens up into a large space before narrowing slightly at its distal end.

To measure the tightness of the active site region of the tunnel, the bottom of the exit tunnel was blocked off a certain distance from then α -amino (*i.e.* the alpha-nitrogen of the first amino acid when connected to both A-site and P-site tRNAs in an intermediate state) and a probe was allowed to propagate through the remaining tunnel and active site area (Figure 4.6). The results of this computation showed that any probe having a radius larger than 1.7 Å is trapped within the lumen; only probes smaller than 1.68 Å can freely leave the lumen of the tunnel via its PTC end. This radius is to be compared to that of the much larger 2.66 Å probe (Figure 4.2) which can freely pass through the entirety of the solvent volume in the ribosome.

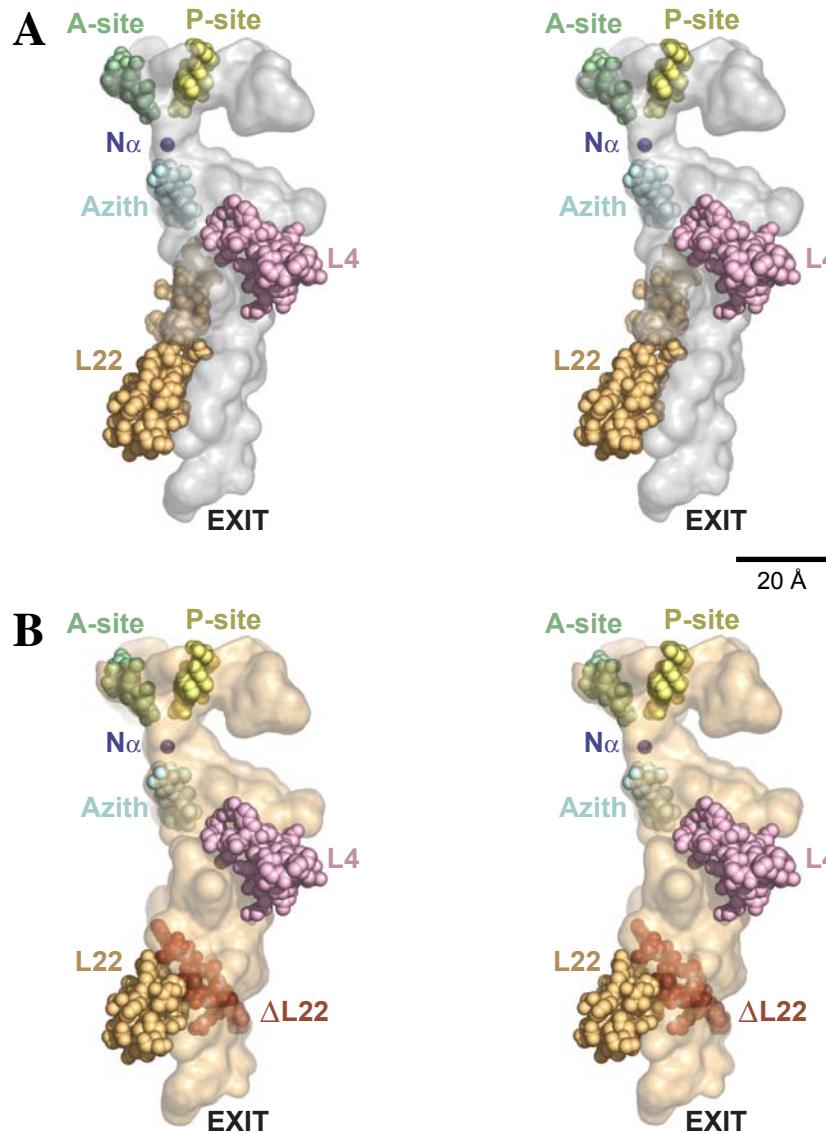


Figure 4.4: Stereo diagram of the polypeptide exit tunnel showing the positions of landmarks. (A) The wild-type *H. marismortui* polypeptide exit tunnel is shown with important landmarks (Klein *et al.*, 2001). (B) The *H. marismortui* polypeptide exit tunnel with mutated L22 protein is shown (Tu *et al.*, 2005). Both figures are oriented with the tRNA binding cleft at the top and the exit at the bottom. L22 is orange, L4 is pink, A-site and P-site CCA tRNA tails are green and yellow, respectively, the active site α -amino is dark blue, azithromycin macrolide is cyan, and the excluded surface for the wild-type and mutant exit tunnel is gray and tan, respectively. The dynamic loop of the mutated L22 protein (Δ L22) is colored in red and was not used in the volume calculation.

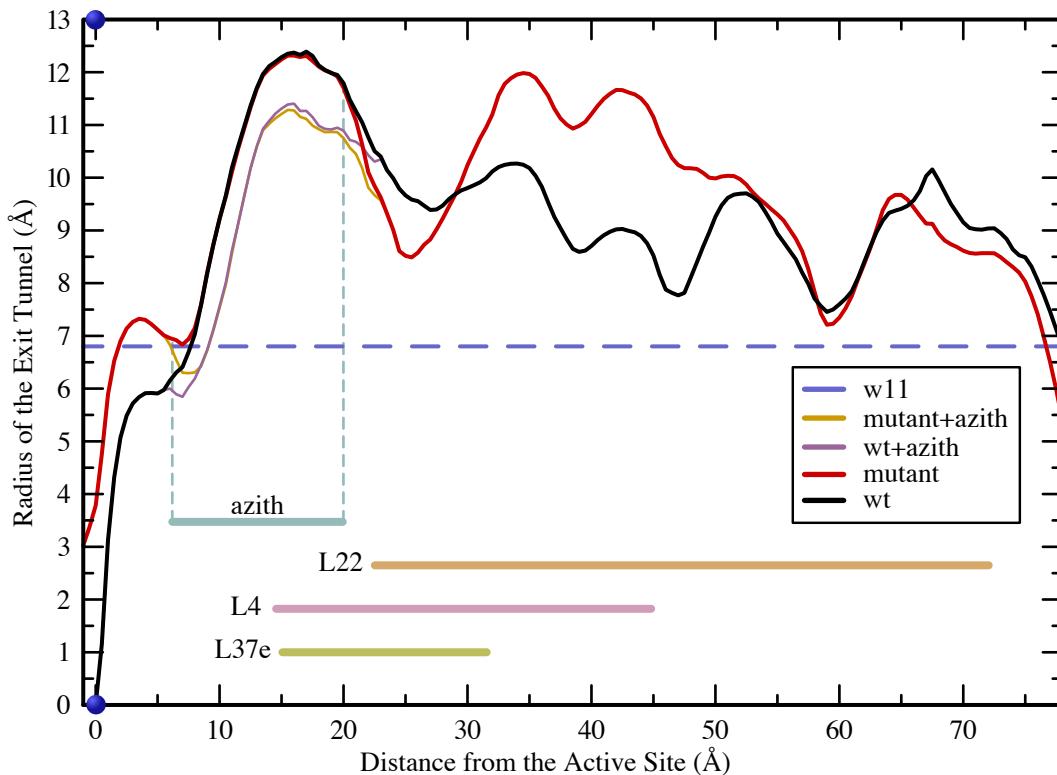


Figure 4.5: The width of the ribosomal polypeptide exit tunnel. The cross-sectional area of the tunnel was found by intersecting a plane with the volume of the canonical tunnel (Figure 4.4). The sectioned area was then converted into a radius by assuming it to be circular (though this is less than ideal). Shown is the wild-type tunnel (black) and the L22 deletion mutant (red). The radii adjusted for azithromycin macrolide are shown for the wild-type (pink) and mutant (orange) exit tunnels. The lines on the bottom of the graph show where the atoms of the proteins L22, L4, and L37e as well as the azithromycin macrolide are located relative to the exit tunnel. Also, the radius of the tungsten-11 cluster (W11) is shown as a dashed purple line for guidance. The zero point corresponds to the α -amino present in other figures.

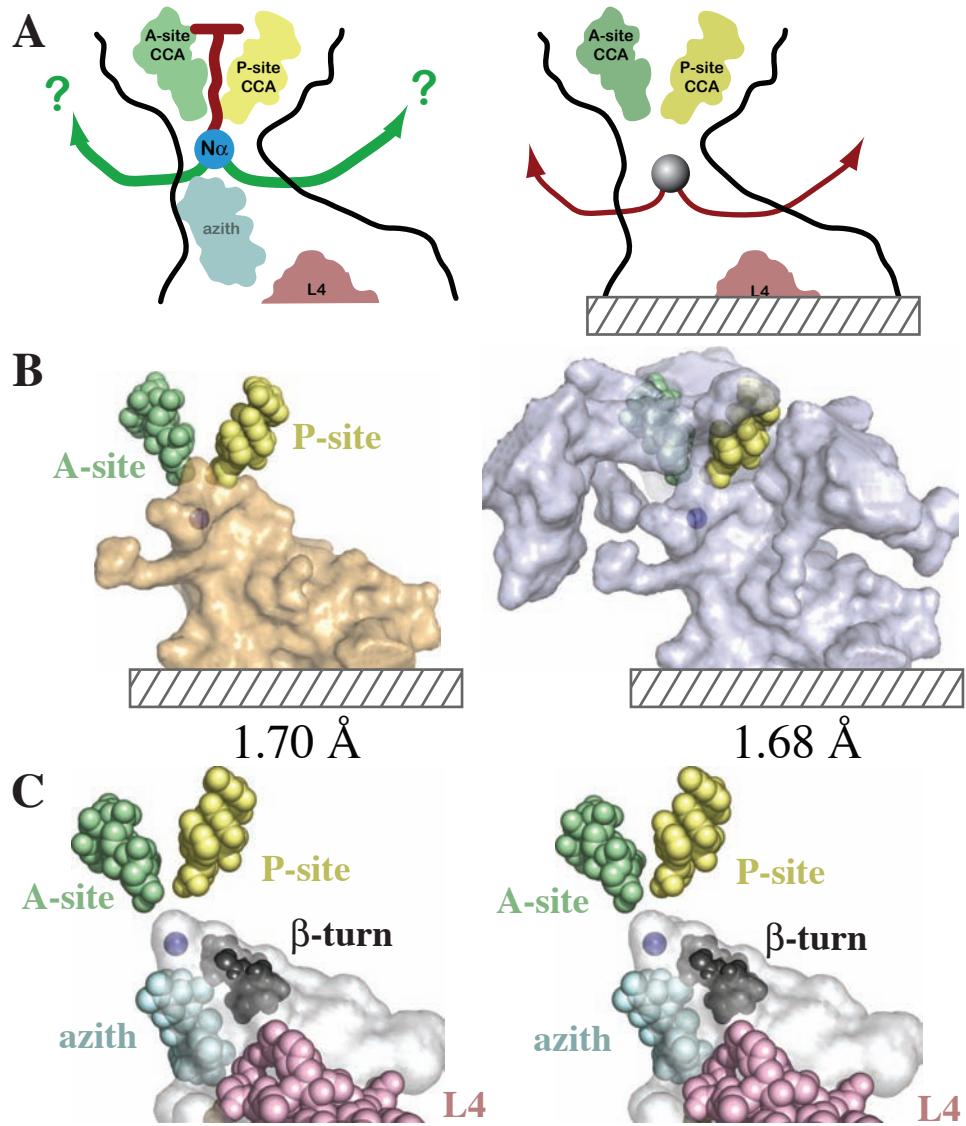


Figure 4.6: Tightness of the exit tunnel around the peptidyl-transferase center. (A) A peptide being expressed by the ribosome is too big to turn around and go back through the tRNA binding area (left, red line), therefore it must escape out through the periphery for this model to work (left, green arrows). A probe is placed inside the lumen of the exit tunnel with its top sealed off by the A-site (green) and P-site (yellow) tRNA CCA tails and its bottom sealed off by creating a barrier *in silico* (right). (B) The top part of the exit tunnel mapped by a probe of radius 1.70 \AA (left) and probe of radius 1.68 \AA (right). (C) A stereogram of a β -turn (black), a simple protein secondary structure motif, is shown above the macrolide (cyan) inside the exit tunnel near the active site.

Macrolides do not plug the tunnel

As shown by Tu *et al.* (2005) and in Figure 4.4A, the macrolide antibiotic, azithromycin does not physically block the tunnel. Further, the width of the tunnel is only moderately affected by its addition (Figure 4.5 red line). The average radius over the first 25 Å of the exit tunnel drops from 9.21 (± 2.9) Å to 8.54 (± 2.6) Å upon the addition of the macrolide. Secondly, Tu *et al.* (2005) created a 3 amino acid deletion mutant of the ribosomal protein L22 that allows ribosome to again translate peptide in the presence of macrolide antibiotics. These mutations caused the whole top part of the protein loop to flip down in the tunnel (Figure 4.4B, red) and open up a wide area in the middle of the tunnel (Figure 4.5, red) between 29 Å and 53 Å from the active site. The average radius over the entire exit tunnel increases from 9.10 Å to 9.37 Å and from 9.23 Å to 10.77 Å in the region of interest.

The tunnel is permeable to water

To determine the permeability of the exit tunnel to water, a water-sized probe (radius 1.5 Å) was placed inside tunnel lumen and the fraction of the solvent volume inside the shell that the probe can reach was calculated. As is evident in Table 4.1 (and Figure 4.1, gray vertical line), the two volumes are nearly identical. In other words, if the tunnel is defined using a 1.5 Å radius probe, then the volume inside its surface is 93% of the total solvent-accessible volume inside the shell of the large ribosomal subunit. Thus effectively, there is no tunnel within the large ribosomal subunit for water-sized objects. Cavities big enough to contain solvent molecules account for only 0.02% of the solvent volume. The remaining solvent volume (~7%) that cannot be reached from the tunnel is solvent “trapped” in concavities at the surface of the particle by the shell.

The tunnel so defined comprises a huge network of interconnected channels that permeates the entire structure of the large ribosomal subunit. Since this network reaches the particle’s shell in many places, molecules the size of water molecules should be able to diffuse into and out of the

<i>Exterior Volumes (\AA^3)</i>			<i>Mass</i>
shell volume 2,290,980	solvent-excluded volume 1,395,480	VDW volume 1,086,421	molecular mass (Da) 1,340,664
<i>Interior Volumes (\AA^3)</i>			<i>Percentage</i>
solvent volume 895,500	empty volume 309,059	solvent-cavity volume 1,578	fractional solvent volume 39.1%
<i>Solvent and Tunnel Volumes (\AA^3)</i>			
1.5 \AA tunnel volume 824,323	1.5 \AA solvent volume 889,475	3.0 \AA tunnel volume 22,773	3.0 \AA solvent volume 551,943
<i>Surface Areas (\AA^2)</i>			
shell area 138,633	solvent-excluded area 434,727	VDW area 1,003,020	

Table 4.1: Volumes of the polypeptide exit tunnel. Volumes are the same as those defined in Chapter 3 (see page 50). The solvent volume is the shell volume minus solvent-excluded volume. The empty volume is the solvent-excluded volume minus VDW volume. The fractional solvent volume is the solvent-excluded volume divided by the shell volume. The tunnel volume is 93% of the solvent volume at 1.5 \AA and 4.4% of the solvent at 3.0 \AA . The solvent-cavity volume is the cavity volume with a probe of 1.5 \AA .

tunnel without difficulty. The tunnel that emerges when a 3.0 Å radius sphere is used, on the other hand, is a much smaller object. Its volume is about 4.4% of the total solvent volume of the particle.

The three major branches of the exit tunnel

There are three major branches in the exit tunnel. The first branch is evident for probes having radius of 2.85 Å or less (Figure 4.7, top). It occurs at the beginning of the exit tunnel and a nascent peptide would have to make a sharp turn (greater than 90°) to enter it. Further the 2.85 Å branch is very narrow throughout its length and not only includes a large cavern with no exit, but also several kinks which the nascent peptide would have difficulty passing through. As the probe radius is lowered, a second branch appears at 2.70 Å toward the exit end of the tunnel (Figure 4.7, middle). This branch, which is only passable by objects smaller than 2.70 Å, also requires a sharp turn to enter. It is very narrow throughout, and it includes a dead-end cavern that could trap the peptide before it reaches the exterior of the subunit. Finally, if the probe radius is reduced to 2.66 Å, then a large amount of internal solvent becomes accessible from the exit tunnel (Figure 4.7, bottom). This 2.66 Å branch, located towards the exit end of the tunnel, also requires a sharp turn to enter and is initially very narrow. What differentiates this branch from the other branches is that once the probe passes into the narrow branch, it opens up to a wide space with numerous passages to the exterior.

4.3 DISCUSSION

4.3.1 Branching of the Exit Tunnel

In some electron microscopic reconstructions of the ribosome, the tunnel branches at its distal end, and these branches appear to be big enough to accommodate nascent peptides (Gabashvili *et al.*, 2001). However, our calculations show that for objects the size of polypeptides the tunnel is not branched. How can this discrepancy be explained? It likely that the branching seen in low resolution electron microscopic images is a Fourier series termination artifact similar to the one that causes

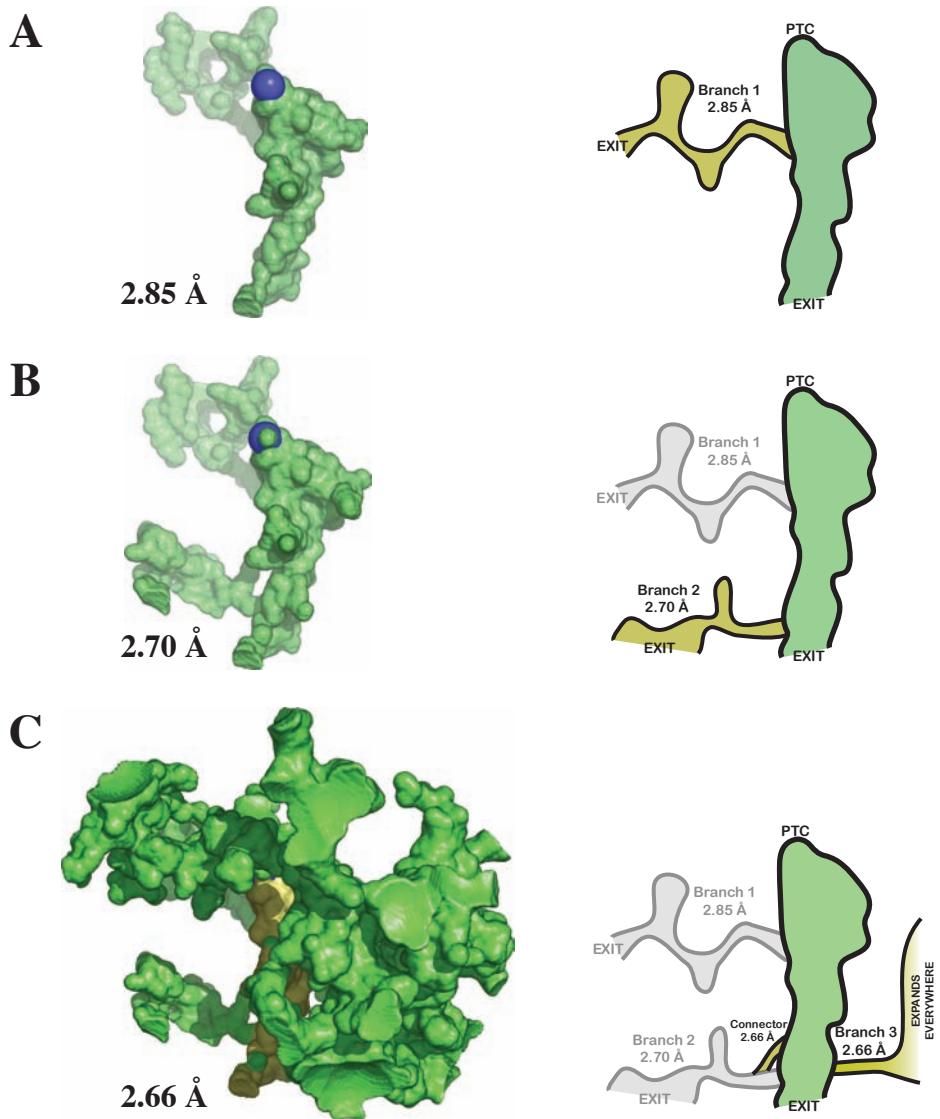


Figure 4.7: The three major branches of the exit tunnel. The excluded-volume of a different-sized probes reachable from the tunnel lumen showing the branches of the exit tunnel. Excluded-volumes from probe radii: (A) 2.85 Å (B) 2.70 Å and (C) 2.66 Å are shown on the left and two-dimensional schematics of the excluded volumes are shown on the right. Again the α -amino is a large blue sphere and the exit is located at the bottom.

α -helices to resemble slender rods in X-ray crystallographic electron density maps calculated at resolutions around 6 Å. The 9 Å resolution crystallographic electron density map published for the large ribosomal subunit looks far more porous than it really is for this same reason (Ban *et al.*, 1998). It should also be noted that the choice of the appropriate contouring level of electron density maps from electron microscopic analyses is not an exact science; it is based on molecular weights and specific densities (Conway *et al.*, 1996). These contouring problems coupled with resolution artifacts could lead to difficulties with interpretation. In this connection, when low resolution EM images are superimposed on the high resolution X-ray crystal structures, it is found that a ribosomal protein, which should have relatively low electron density, occupies the space assigned to the proposed branch (Daniel Klein, personal communication).

4.3.2 Protein Folding in the Exit Tunnel

What is the conformation adopted by nascent proteins as they traverse the exit tunnel? Nascent chains could exist solely as extended chains, form secondary structure, or even fold into native proteins. Important insights can be derived by combining what has been learned about the tunnel crystallographically with biochemical data first reported almost 40 years ago (Malkin & Rich, 1967). In 1967, Rich and colleagues demonstrated that the last (*i.e.* C-terminal) 30 residues of nascent peptides are protected from proteolytic attack by their association with the ribosome. Many such experiments have been performed since using ribosomes from different sources and nascent peptides of different kinds (for review see Hardesty & Kramer, 2001). They have all consistently shown that the length of the peptides protected by the ribosome is 30 to 40 residues. Since the distance from the PTC to the distal end of the tunnel is about 80 Å, the distance of the tunnel traversed by each residue of nascent peptides is, on average, 2.0 to 2.7 Å. This distance is considerably less than the 3.5 Å per residue it would be if nascent proteins were in an extended conformation, but not as short as the 1.5 Å per residue it would be if nascent peptides were entirely α -helical. An increasing body

of experimental data supports the view that nascent peptides are indeed in a α -helical conformation (Lu & Deutsch, 2005; Woolhead *et al.*, 2004). Furthermore, a recent theoretical study by Ziv *et al.* (2005) has shown that confinement of peptide chains in cylindrical cavities the size of the exit tunnel drives them to adopt α -helical conformations entropically.

The observations just described notwithstanding, it has been suggested that nascent peptides might become at least partially folded at the tertiary level inside the tunnel (Gilbert *et al.*, 2004). Is this plausible geometrically? Accidentally, some information was obtained about the bore of the tunnel several years ago from a heavy-atom isomorphous replacement experiment done using a cluster compound containing 11 tungsten atoms (Wei *et al.*, 1997). This irregularly shaped molecule, which has maximum linear dimensions of 13.6 Å, binds to 3 sites inside the tunnel as well as to a single site in the tRNA binding cleft (Figure 4.8C; Ban *et al.*, 1998). As it happens, the diameter of the largest sphere that can fit inside the tunnel is only slightly larger, 13.7 Å, but as reported earlier the average cross-sectional diameter is 18.4 Å. The diameter of a polyalanine α -helix is considerably smaller than this at 9.2 Å, and as expected, α -helices will fit comfortably in the tunnel, provided some bends in the helix axis are permitted (Figure 4.8A). However, small protein domains, such as an IgG domain will not fit (diameter of 25.6 Å, Figure 4.8B). Thus neither the biochemical data available nor the structure of the large ribosomal subunit itself supports the hypothesis that tertiary folding of nascent peptides occurs in the ribosome.

Those who believe that the tunnel does allow tertiary folding might argue that the conformation of the tunnel is not fixed; that it might be able to accommodate large objects the way a boa constrictor swallows a pig. While it is impossible to rule this hypothesis out, it is implausible. Since the tunnel lies in the middle of a huge macromolecular structure, any conformational change that significantly altered its dimensions would require the disruption of a very large number of tertiary interactions. Furthermore, the tunnel has the same size and shape in the crystal structures now available for

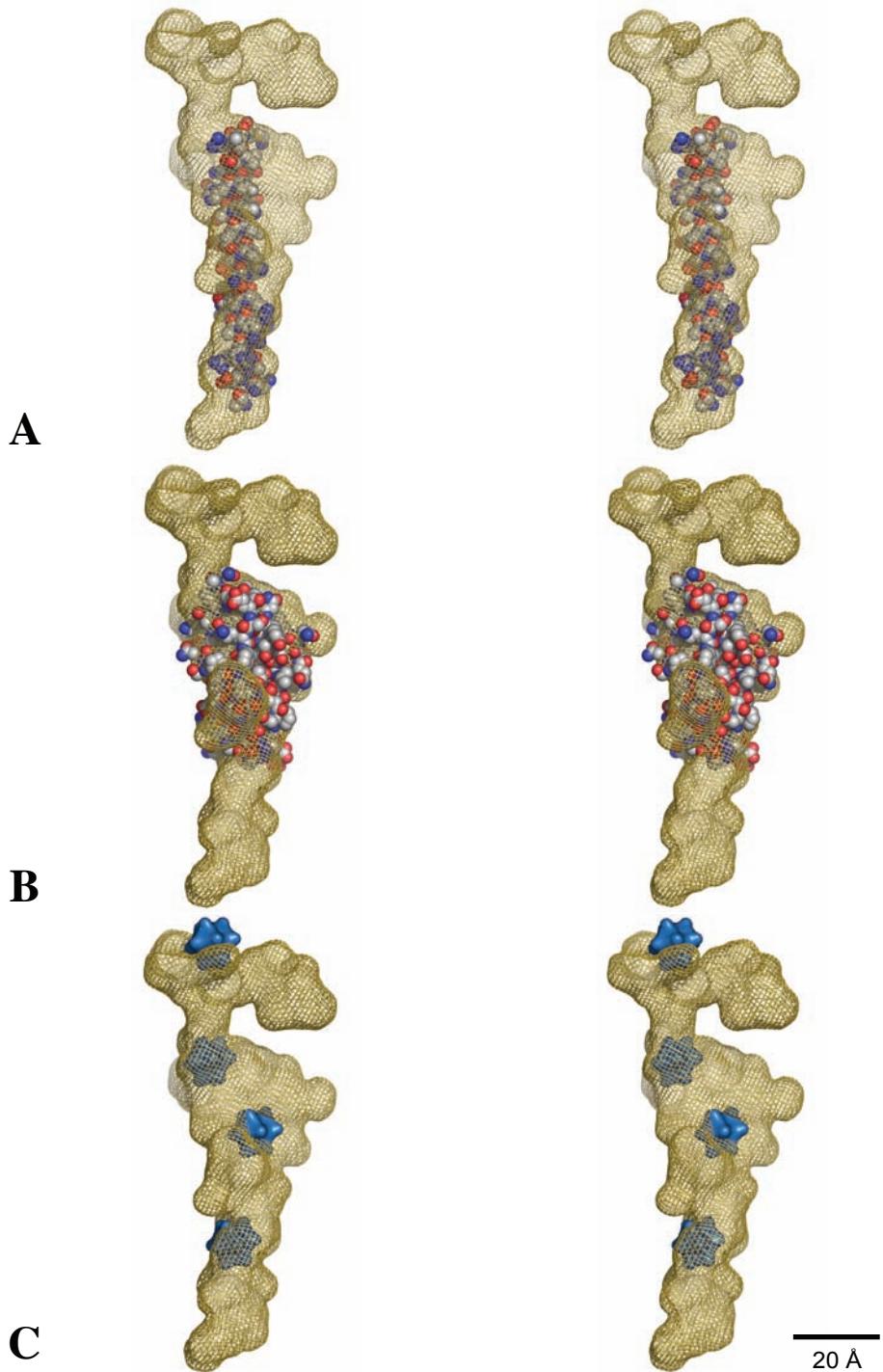


Figure 4.8: Stereo projections of space-filling models of molecules inside the polypeptide exit tunnel. (A) A straight, 41 amino acid α -helix (Ellenberger *et al.*, 1992) is inserted into the tunnel, oriented to maximize the fraction of the helix inside the tunnel. (B) A model of an IgG domain (Soroka *et al.*, 2003) is placed in the middle of the exit tunnel, again oriented so that the fraction of the domain inside the tunnel lumen will be maximal. (C) The locations of four tungsten-11 (W11) heavy cluster compounds found when they were soaked into preformed crystals of the large ribosomal subunit from *H. marismortui* (Ban *et al.*, 1998). The W11 clusters are displayed lodged in the tunnel in three locations and one in the cleft immediately above the peptidyl transferase site (PTC).

large ribosomal subunits from two different species (Ban *et al.*, 2000; Harms *et al.*, 2001) and for complete ribosomes from two other species (Schuwirth *et al.*, 2005; Yusupov *et al.*, 2001).

Finally, in considering the possibility that the tunnel functions as a chaperone, it is useful to compare the interior chamber of a known chaperone, the thermosome (Ditzel *et al.*, 1998) with the interior of the ribosomal tunnel. The two volumes are similar in length, but their diameters differ enormously (Figure 4.9). Although, a sphere 50 Å in diameter would fit comfortably within the interior cavity of the thermosome, nothing anywhere near this large will fit inside the ribosome.

4.3.3 Permeability of the Large Ribosomal Subunit to Ions

The translocon is a large, membrane protein complex to which the large subunit of the ribosome binds when it is synthesizing proteins destined for passage through or insertion into membranes. The translocon binds to the distal end of the exit tunnel. One of the interesting properties of the translocon-ribosome complex is that protein secretion is not accompanied by ionic flow that would discharge the electrochemical gradients that exist across membranes. It is not easy to understand how this is accomplished. Since peptide chains are irregular in shape, it is plausible to that a channel big enough to allow peptides to pass would accommodate an occasional low molecular weight ion.

Johnson and coworkers have observed that fluorophores inside the peptide exit tunnel of translocon-bound ribosomes cannot be quenched by iodide ions added to the surrounding solution (Figure 4.10A; Liao *et al.*, 1997). Consequently they proposed that the ribosome itself is part of the system that makes the ribosome translocon complex ion-tight. This hypothesis appears to be further supported by a recent report that puromycin treatment of endoplasmic reticulum membranes containing ribosome-translocon complexes with attached nascent polypeptides, which are normally ion-tight, leads to transmembrane calcium ion flow (Figure 4.10B; Van Coppenolle *et al.*, 2004). It is hypothesized that the release of nascent polypeptides from the ribosomes caused by reaction with puromycin opens the PTC end of the exit tunnel allowing ions to flow.

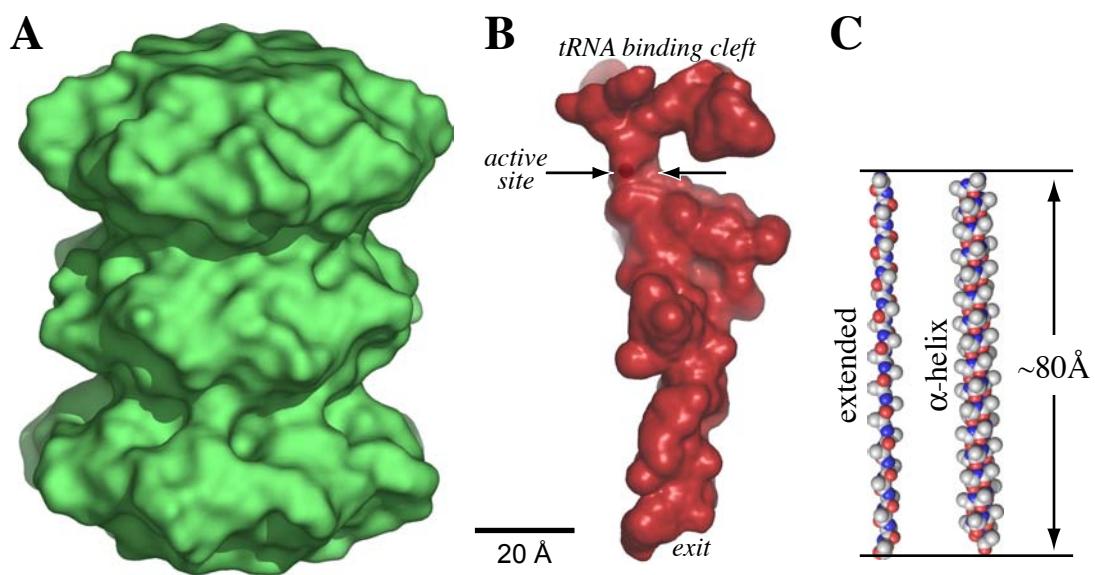


Figure 4.9: The interior volumes of a chaperonin and the polypeptide exit tunnel compared. (A) The excluded surface (green) of the internal chamber of the *T. acidophilum* chaperonin (Ditzel *et al.*, 1998). (B) The excluded surface of *H. marismortui* large ribosomal subunit polypeptide exit tunnel (red) determined using a 2.95 Å radius probe. (C) An extended polyalanine peptide chain (left) and polyalanine α -helix (right) of the exact length of the tunnel from the active site to the exit ($\sim 80\text{ \AA}$). It takes the extended chain 25 amino acids to reach the exterior. The α -helix requires 60 residues to reach the exterior. Images are drawn to the same scale.

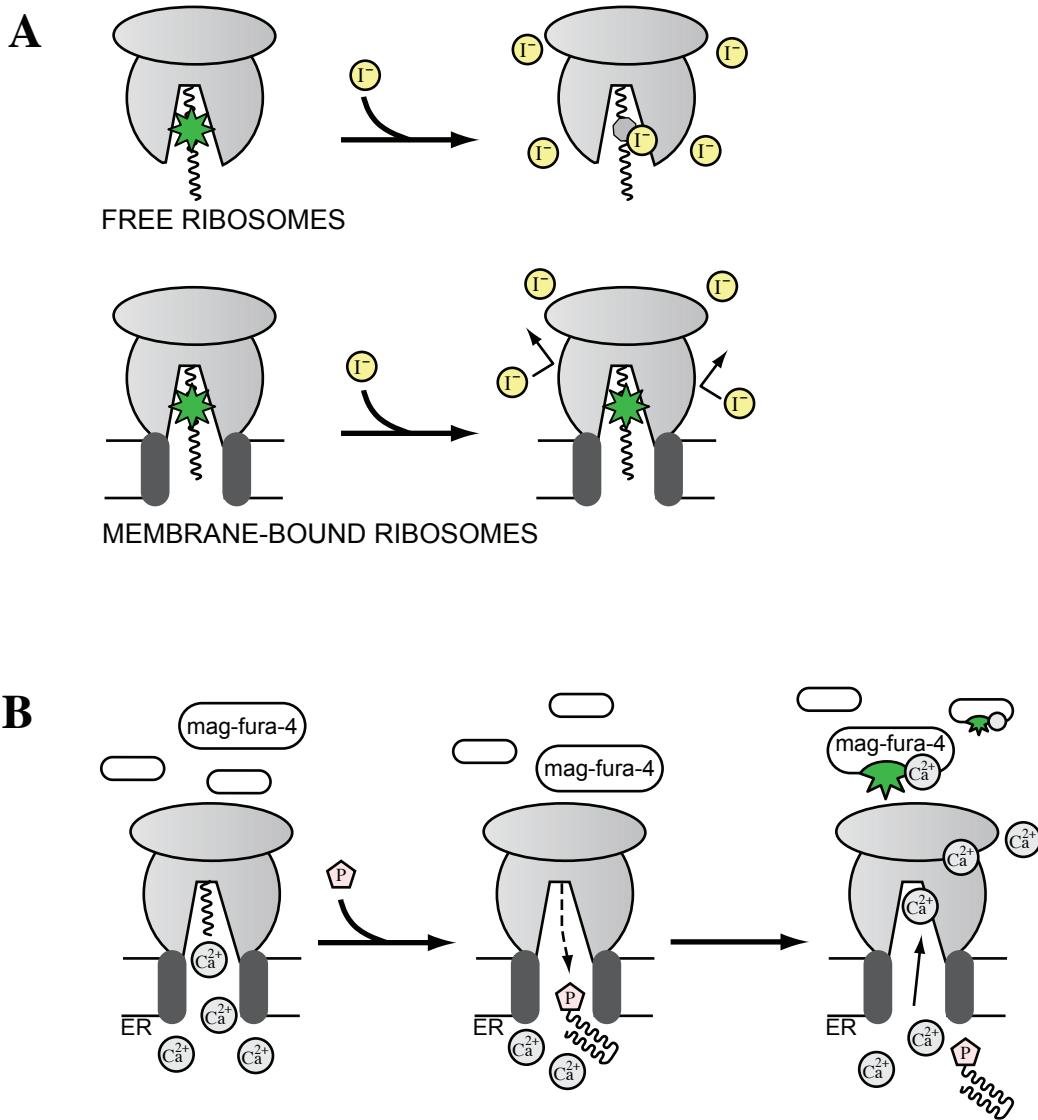


Figure 4.10: Permeability of ions from the exit tunnel. (A) Liao *et al.* (1997) conducted experiments using iodide ions which quench a fluorophor attached to a nascent peptide. Free ribosomes (ribosomes in solution) experienced quenching by the iodide, but membrane-bound ribosomes saw no reduction in fluorescence. (B) Van Coppenolle *et al.* (2004) performed a similar experiment using a fluorophor sensitive to calcium. The ribosomes bound to the calcium-rich endoplasmic reticulum (ER) membrane do not leak calcium ions during the expression of protein resulting in no fluorescence. When puromycin—an antibiotic that causes the nascent peptide chain to terminate and fall off—is added a sudden spike of fluorescence is seen.

It is important to further examine Johnson's result geometrically. The Stokes' radii of hydrated cations and anions fall in the range of 3.3 to 4.1 Å (Table 4.2; Marcus, 1985) which is larger than the radius of the largest sphere (3.0 Å) capable of leaking out of (or into) the exit tunnel laterally (Figure 4.1), assuming that the wall of the tunnel is rigid. One could argue that these ions might enter ribosomal channels having diameters less than their hydrated radii because polar groups in the ribosome can replace waters of hydration. However, even if this were to occur, it is unlikely that the ribosome would be very accommodating to anions such as iodide, because the ribosome itself is a polyanion. The impermeability of the ribosome to iodide ions does not mean that the ribosome is impermeable to protons or water, or that it is part of the seal that keeps the ribosome-translocon complex from leaking ions. It should also be noted that recent crystallographic results suggest that the translocon pore itself is likely to be the component of the system that makes it ion-tight (Menetret *et al.*, 2005).

4.3.4 Mechanism of Polyphenylalanine Translation

It has been proposed and is widely believed that during polyuridylic acid (polyU) translation that the polyphenylalanine (polyphe) product does not travel down the length of the tunnel, but either bunches up in the space between the subunits or exits along another trajectory (Ryabova *et al.*, 1988; Odom *et al.*, 1991). The reason is that polyU directed polyphe synthesis is not inhibited by erythromycin, which appears to prevent the passage of all other nascent peptides down the tunnel (Odom *et al.*, 1991). Nevertheless, there are data from fluorescence measurements indicating that polyphe encounters erythromycin bound in the tunnel (Hardesty & Kramer, 2001). Furthermore, Papadopoulos *et al.* (2006) recently reported that mutations in ribosomal protein L4 makes polyphe synthesis sensitive to erythromycin. L4 contributes to the tunnel wall beyond the site where erythromycin binds (Figure 4.6, page 102)

<i>ion</i>	Stokes Radius (Å)		<i>hydration #</i>
	<i>unhydrated</i>	<i>hydrated</i>	
I ⁻	2.2	3.33	2.8
Cl ⁻		3.34	3.9
K ⁺		3.34	5.1
Na ⁺		3.51	6.5
Ca ²⁺		3.96	10.4
Mg ²⁺		4.12	11.7

Table 4.2: Stokes radii of selected ions. The Stokes radii of several ions are shown (Marcus, 1985).

Looking at the geometry of the exit tunnel, it is difficult to understand how a polypeptide chain could do anything except progress down the exit tunnel. To do otherwise, it would have to initially enter the tunnel, then quickly turn around and exit through some lateral branch. As shown earlier, a probe placed in the top part of the tunnel is trapped inside unless its radius is less than 1.7 Å (Figure 4.6, page 102). An extended polyalanine chain, which is much smaller than a polypeptide chain, has a radius of \sim 3.0 Å (see Methods, page 119) suggesting that this is not possible without major rearrangements of the active site. Though a nascent chain has plenty of room to travel down the axis of the tunnel, there are no observed paths a peptide chain can take that would allow it to enter the subunit interface region of the ribosome. Thus one concludes that polypeptide does proceed down the tunnel, the same as all other nascent proteins, but that its passage down the tunnel is not inhibited by erythromycin.

4.3.5 Thermal Fluctuations

Some of our conclusions about the properties of the polypeptide exit tunnel might be challenged on the grounds that biological structures cannot be fully understood on the basis of their time-averaged structures. For example, the atoms that form the wall of the exit tunnel of the ribosome, like every other atom in the ribosome, exhibit thermal fluctuations in position. An estimate of the magnitude of this effect within the populations of macromolecules found in crystals can be obtained from the Debye-Waller factors (B-factors) assigned to the atoms as their structures are refined. The average value of the B-factors for RNA atoms in the *H. marismortui* large ribosomal subunit crystal structure is 30.8 (\pm 13.9) Å² (Klein *et al.*, 2001). This number implies that in any given direction, the root mean square variation in the position of the average RNA atom in the crystals used to obtain the structure in question was 0.63 Å. Since static packing disorder is certain to contribute significantly to the B-factors of these crystals, 0.63 Å is an upper bound estimate for the magnitude of thermal fluctuations. Random structural fluctuations of this magnitude cannot alter the dimensions of the

tunnel enough to create protein accessible side branches, or to allow nascent polypeptides to form tertiary structure, or to make it significantly more accessible to hydrated ions than its time-averaged structure indicates.

4.4 METHODS

4.4.1 Atomic Coordinates and VDW Radii

All macromolecule files were taken from the Protein Data Bank (PDB; Berman *et al.*, 2000). For all PDB files, hydrogens were added using the program REDUCE with default settings (Word *et al.*, 1999). The van der Waals (VDW) radii used for atoms were taken from MSMS program based on stereochemistry of the RNA atoms (Sanner *et al.*, 1995; Sanner, 1995; Sanner *et al.*, 1996), and merged with coordinate data to create simple XYXR files that specify the coordinates and VDW radius of every atom in a molecule.

Ribosome structures

1JJ2 was the main PDB file used for the *H. marismortui* large ribosomal subunit (Klein *et al.*, 2001). Additionally, eight different PDB files of the *H. marismortui* 50S structure were assessed: 1K8A, 1K9M, 1KD1, 1M1K (Hansen *et al.*, 2002), 1K73, 1KC8, 1N8R, and 1NJI (Hansen *et al.*, 2003). These files were used to determine confirm solvent volumes and tunnel shapes. The mutant L22 protein structure was taken from the file with PDB code 1YJ9 (Tu *et al.*, 2005). A-site and P-site CCA tRNA tails were taken from the structure with PDB code 1QVG.

At the time this research was conducted only one model of the 50S ribosomal subunit was complete (PDB file: 1JJ2, Klein *et al.*, 2001). The other models from *Deinococcus radiodurans* (PDB file: 1Nkw, Harms *et al.*, 2001) and from *Thermus thermophilus* (PDB file: 1GIY, Yusupov *et al.*, 2001) did not contain any protein side chain information making it impossible to extract an exit tunnel. Subsequently two new models have become available from *Escherichia coli* (PDB files:

2AW4 and 2AWB, Schuwirth *et al.*, 2005) and from *Thermus thermophilus* (PDB files: 2J01 and 2J03, Selmer *et al.*, 2006) as well as two updated models from *Thermus thermophilus* (PDB files: 1YL3 and 1VS9 replacing 1GIY, Jenner *et al.*, 2005; Korostelev *et al.*, 2006) of which all six PDB files are complete models. All of these structures are lightly explored in Appendix C.

Protein structures

Spinach rubisco (1RCX (Taylor & Andersson, 1997) was chosen as a typical protein because it has a central channel, long protein chains and minimal symmetry. The leucine zipper α -helix (Figure 4.8A) is from PDB file 1YSA (Ellenberger *et al.*, 1992) and the IgG domain (Figure 4.8B) is the first domain of NCAM Ig1-2-3 protein, PDB file 1QZ1 (Soroka *et al.*, 2003).

4.4.2 The Rolling Probe Approach to Determining Volumes

The rolling probe approach was used exactly the same as in Chapter 3 (page 83). The exceptions are listed below.

Two methods for defining the tunnel: connected versus touching

There are two methods of placing a probe inside the tunnel. Both methods come down to semantics, but basically first method, the “accessible method” requires that the probe start at a point and explore space and the second method, the “direct method” has many probes and whenever one probe can “touch” another probe there are connected. The accessible method is when the probe is placed inside the tunnel and is allowed to freely roam. While the accessible method is easier to describe in words it is actually more complex computationally. In contrast, the direct method is more simple computationally, but not as physically relevant. The direct method takes into account all connected solvent, *i.e.* if the probe does not necessarily have to pass through a narrow channel, but only connect to another region in the channel that can fit the probe. Because of this, the direct method always has

a larger volume than the accessible method by definition. To get a relevant version of the exit tunnel the accessible method is used.

Extraction of interior volumes using the accessible method

As in Chapter 3, the first step in the determination of the surface of the polypeptide tunnel was the calculation of the shell volume of the macromolecule, determined using a 10 Å radius probe, on a fine grid. Second, the accessible surface was determined on the same grid by the rolling probe method (Figure 3.1, page 51) using a smaller probe. Subsequently, the accessible volume grid of the smaller probe was subtracted from the shell volume grid to yield the set of interior grid points that are accessible to the smaller probe. This subtraction yields all points reachable by the probe center within the shell, but not necessarily from the tunnel. The subset of grid points connected to a list of pre-defined tunnel points was then identified from the larger set. This connected subset contains only points accessible to the probe center that are reachable from the exit tunnel lumen. In order to obtain the excluded volume of the tunnel, the accessible volume subset was convoluted with the probe sphere. The set of interior grid points so defined is equivalent to what would be obtained if a probe placed manually in the lumen of the tunnel were allowed to roam freely in the structure without exiting the tunnel. Typical results are shown in Figure 4.2 (page 96). To ensure the continuity of the exit tunnel with the tRNA binding cleft of the large subunit, the highly dynamic uridine base 2620 (with an average B-factor of 37.9 Å² compared to values of ~20 Å² for nucleotides in the vicinity) was removed from above the tunnel when generating figures, but not when numerical calculations were being done. The unlabeled polypeptide exit tunnel surfaces shown in figures were generated using probes between 2.9 Å and 3.2 Å.

The interior volume of the thermosome (Figure 4.9), from *Thermus acidophilum* (PDB ID: 1A6D; Ditzel *et al.*, 1998) was computed as described above except that its shell volume was computed using a 40 Å probe in order to ensure that its interior cavity would be entirely included in

its shell. The probe used to obtain its interior volume had a radius of 4 Å; smaller probes can “leak” out of its interior.

Cross-sectional area

To obtain the cross-sectional area of the exit tunnel (Figure 4.5), planes were placed at steps of 0.5 Å perpendicular to the axis of the tunnel. The cross-sectional area was then calculated by dividing the plane up into a grid squares (pixels) with smaller dimensions than half the size of the original three-dimensional voxels (grid points). Each square of known area is determined to be inside or outside of the tunnel and all the squares that fell inside the tunnel were added and multiplied by the surface area of each square (pixel). The area was then calculated for all planes along the length of the tunnel using this technique.

4.4.3 Calculation of Linear Dimensions

To determine the width extended polyalanine chains and α -helices, alanines were modeled in increments of 50 amino acids. The lengths of the models were calculated using $C\alpha$ distances and volumes using a rolling probe of 5 Å. The cross-sectional area of both alanine chains was then determined from the slopes of a linear regression of the data (Figure 4.11). Diameters were then calculated assuming the cross-section was circular:

$$A = \pi r^2 \implies r = \sqrt{\frac{A}{\pi}} \quad (4.1)$$

For β -sheet, extended alanine chains the slope equals 24.5 Å² with a 2 Å probe and 29.6 Å² with a 5 Å probe. This corresponds to a radius of 2.79 Å and 3.07 Å, respectively (Figure 4.11, green lines). Likewise, an α -helical polyalanine chain have a slope of 59.7 Å² and 68.2 Å² with radii of 4.36 Å and 4.66 Å, respectively (Figure 4.11, red lines). The size of the polyphenylalanine chain was not investigated, because it has an irregular shape and can exists in several different isoforms.

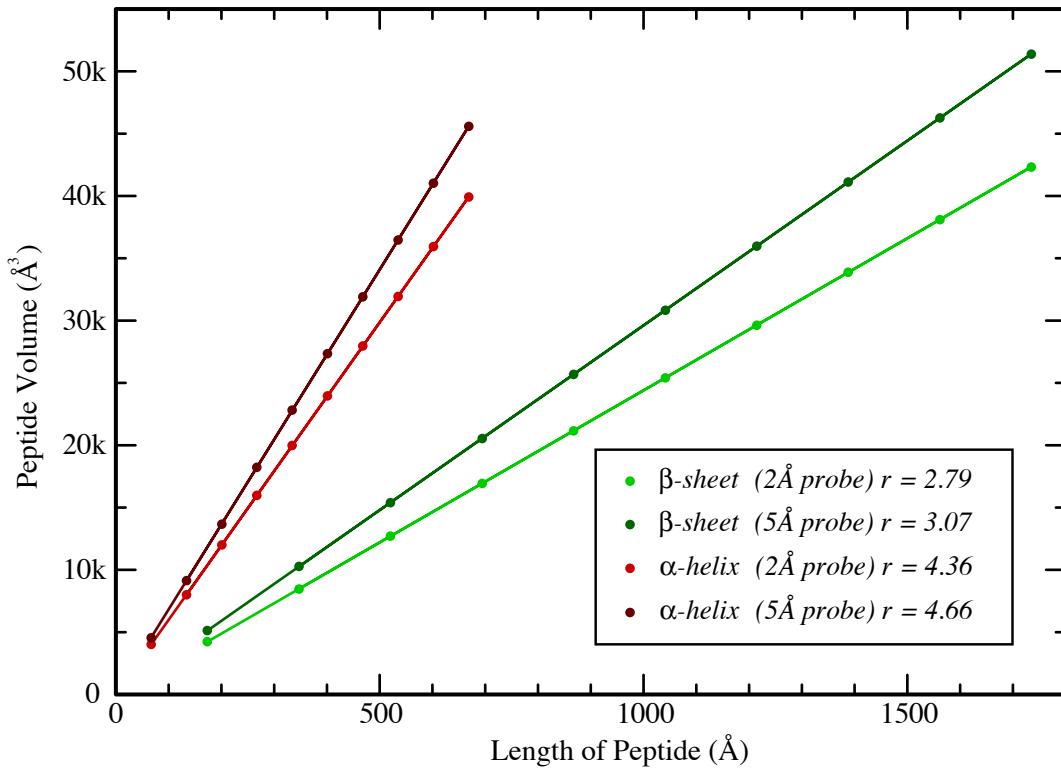


Figure 4.11: Calculation of linear dimensions for polyalanine chains. Lengths and volumes for both β -sheet (green) and α -helical (red) polyalanine chains in increments of 50 amino acids were calculated. The x-axis shows the length (from end to end) of the chains. The y-axis shows the volume at two different probe radii (2 Å, lighter colors and 5 Å, darker colors) of both types of alanine chains.

The size of the IgG domain was calculated by fitting its volume (19,391 Å³) and surface area (3,836 Å²) to that of an oblate spheroid. The major and minor axis lengths of the resulting spheroid were 56.6 Å and 25.6 Å, respectively.

4.4.4 Availability of Software

The volume calculation programs and source code described here are intended to be applicable to similar problems and can be obtained online at <http://geometry.molmovdb.org/3v/> and Appendix F.

4.5 REFERENCES

- I. Agmon, T. Auerbach, D. Baram, H. Bartels, A. Bashan, R. Berisio, P. Fucini, H. A. Hansen, J. Harms, M. Kessler, M. Peretz, F. Schluenzen, A. Yonath, and R. Zarivach (2003). “On peptide bond formation, translocation, nascent protein progression and the regulatory properties of ribosomes. Derived on 20 October 2002 at the 28th FEBS Meeting in Istanbul”. *Eur J Biochem*, **270** (12): pp. 2543–56.
- N. Ban, B. Freeborn, P. Nissen, P. Penczek, R. A. Grassucci, R. Sweet, J. Frank, P. B. Moore, and T. A. Steitz (1998). “A 9 Å resolution X-ray crystallographic map of the large ribosomal subunit”. *Cell*, **93** (7): pp. 1105–15.
- N. Ban, P. Nissen, J. Hansen, M. Capel, P. B. Moore, and T. A. Steitz (1999). “Placement of protein and RNA structures into a 5 Å-resolution map of the 50S ribosomal subunit”. *Nature*, **400** (6747): pp. 841–7.
- N. Ban, P. Nissen, J. Hansen, P. B. Moore, and T. A. Steitz (2000). “The complete atomic structure of the large ribosomal subunit at 2.4 Å resolution”. *Science*, **289** (5481): pp. 905–20.
- D. Baram, E. Pyetan, A. Sittner, T. Auerbach-Nevo, A. Bashan, and A. Yonath (2005). “Structure of trigger factor binding domain in biologically homologous complex with eubacterial ribosome reveals its chaperone action”. *PNAS*, **102** (34): pp. 12017–22.
- H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne (2000). “The Protein Data Bank”. *Nucleic Acids Res*, **28** (1): pp. 235–42.
- C. Bernabeu and J. A. Lake (1982). “Nascent polypeptide chains emerge from the exit domain of the large ribosomal subunit: Immune mapping of the nascent chain”. *PNAS*, **79** (10): pp. 3111–5.
- J. F. Conway, B. L. Trus, F. P. Booy, W. W. Newcomb, J. C. Brown, and A. C. Steven (1996). “Visualization of three-dimensional density maps reconstructed from cryoelectron micrographs of viral capsids”. *J Struct Biol*, **116** (1): pp. 200–8.
- L. Ditzel, J. Lowe, D. Stock, K. O. Stetter, H. Huber, R. Huber, and S. Steinbacher (1998). “Crystal structure of the thermosome, the archaeal chaperonin and homolog of CCT”. *Cell*, **93** (1): pp. 125–38.
- T. E. Ellenberger, C. J. Brandl, K. Struhl, and S. C. Harrison (1992). “The GCN4 basic region leucine zipper binds DNA as a dimer of uninterrupted α -helices: Crystal structure of the protein-DNA complex”. *Cell*, **71** (7): pp. 1223–37. [http://dx.doi.org/10.1016/S0092-8674\(05\)80070-4](http://dx.doi.org/10.1016/S0092-8674(05)80070-4).
- J. Frank, J. Zhu, P. Penczek, Y. Li, S. Srivastava, A. Verschoor, M. Radermacher, R. Grassucci, R. K. Lata, and R. K. Agrawal (1995). “A model of protein synthesis based on cryo-electron microscopy of the *E. coli* ribosome”. *Nature*, **376** (6539): pp. 441–4.
- I. S. Gabashvili, S. T. Gregory, M. Valle, R. Grassucci, M. Worbs, M. C. Wahl, A. E. Dahlberg, and J. Frank (2001). “The polypeptide tunnel system in the ribosome and its gating in erythromycin resistance mutants of L4 and L22”. *Mol Cell*, **8** (1): pp. 181–8.

- R. J. Gilbert, P. Fucini, S. Connell, S. D. Fuller, K. H. Nierhaus, C. V. Robinson, C. M. Dobson, and D. I. Stuart (2004). “Three-dimensional structures of translating ribosomes by cryo-EM”. *Mol Cell*, **14** (1): pp. 57–66.
- F. Gong and C. Yanofsky (2002). “Instruction of translating ribosome by nascent peptide”. *Science*, **297** (5588): pp. 1864–7.
- J. L. Hansen, J. A. Ippolito, N. Ban, P. Nissen, P. B. Moore, and T. A. Steitz (2002). “The structures of four macrolide antibiotics bound to the large ribosomal subunit”. *Mol Cell*, **10** (1): pp. 117–28.
- J. L. Hansen, P. B. Moore, and T. A. Steitz (2003). “Structures of five antibiotics bound at the peptidyl transferase center of the large ribosomal subunit”. *J Mol Biol*, **330** (5): pp. 1061–75.
- B. Hardesty and G. Kramer (2001). “Folding of a nascent peptide on the ribosome”. *Prog Nucleic Acid Res Mol Biol*, **66**: pp. 41–66.
- J. Harms, F. Schluelzen, R. Zarivach, A. Bashan, S. Gat, I. Agmon, H. Bartels, F. Franceschi, and A. Yonath (2001). “High resolution structure of the large ribosomal subunit from a mesophilic eubacterium”. *Cell*, **107** (5): pp. 679–88.
- L. Jenner, P. Romby, B. Rees, C. Schulze-Briese, M. Springer, C. Ehresmann, B. Ehresmann, D. Moras, G. Yusupova, and M. Yusupov (2005). “Translational operator of mRNA on the ribosome: how repressor proteins exclude ribosome binding”. *Science*, **308** (5718): pp. 120–123.
- D. J. Klein, T. M. Schmeing, P. B. Moore, and T. A. Steitz (2001). “The kink-turn: A new RNA secondary structure motif”. *Embo J*, **20** (15): pp. 4214–21. <http://dx.doi.org/10.1093/emboj/20.15.4214>.
- A. Korostelev, S. Trakhanov, M. Laurberg, and H. F. Noller (2006). “Crystal structure of a 70S ribosome-tRNA complex reveals functional interactions and rearrangements”. *Cell*, **126** (6): pp. 1065–1077.
- S. Liao, J. Lin, H. Do, and A. E. Johnson (1997). “Both luminal and cytosolic gating of the aqueous ER translocon pore are regulated from inside the ribosome during membrane protein integration”. *Cell*, **90** (1): pp. 31–41.
- J. Lu and C. Deutsch (2005). “Folding zones inside the ribosomal exit tunnel”. *Nat Struct Mol Biol*, **12** (12): pp. 1123–9.
- L. I. Malkin and A. Rich (1967). “Partial resistance of nascent polypeptide chains to proteolytic digestion due to ribosomal shielding”. *J Mol Biol*, **26** (2): pp. 329–46.
- Y. Marcus (1985). *Ion solvation*. Wiley, Chichester; New York.
- J. F. Menetret, R. S. Hegde, S. U. Heinrich, P. Chandramouli, S. J. Ludtke, T. A. Rapoport, and C. W. Akey (2005). “Architecture of the ribosome–channel complex derived from native membranes”. *J Mol Biol*, **348** (2): pp. 445–57.
- K. Mitra, C. Schaffitzel, T. Shaikh, F. Tama, S. Jenni, C. L. Brooks 3rd, N. Ban, and J. Frank (2005). “Structure of the *E. coli* protein-conducting channel bound to a translating ribosome”. *Nature*, **438** (7066): pp. 318–24.

- H. Nakatogawa and K. Ito (2002). “The ribosomal exit tunnel functions as a discriminating gate”. *Cell*, **108** (5): pp. 629–36.
- H. Nakatogawa, A. Murakami, and K. Ito (2004). “Control of SecA and SecM translation by protein secretion”. *Curr Opin Microbiol*, **7** (2): pp. 145–50.
- O. W. Odom, W. D. Picking, T. Tsalkova, and B. Hardesty (1991). “The synthesis of polyphenylalanine on ribosomes to which erythromycin is bound”. *Eur J Biochem*, **198** (3): pp. 713–722. URL <http://search.epnet.com/login.aspx?direct=true&db=aph&an=13705915>.
- S. Pal, S. Chandra, S. Chowdhury, D. Sarkar, A. N. Ghosh, and C. D. Gupta (1999). “Complementary role of two fragments of domain V of 23S ribosomal RNA in protein folding”. *J Biol Chem*, **274** (46): pp. 32771–7.
- G. Papadopoulos, S. Grudinin, D. L. Kalpaxis, and T. Choli-Papadopoulou (2006). “Changes in the level of poly(Phe) synthesis in *Escherichia coli* ribosomes containing mutants of L4 ribosomal protein from *Thermus thermophilus* can be explained by structural changes in the peptidyltransferase center: A molecular dynamics simulation analysis”. *Eur Biophys J*. <http://dx.doi.org/10.1007/s00249-006-0076-4>.
- L. A. Ryabova, O. M. Selivanova, V. I. Baranov, V. D. Vasiliev, and A. S. Spirin (1988). “Does the channel for nascent peptide exist inside the ribosome? Immune electron microscopy study”. *FEBS Lett*, **226** (2): pp. 255–260. [http://dx.doi.org/10.1016/0014-5793\(88\)81434-0](http://dx.doi.org/10.1016/0014-5793(88)81434-0).
- M. Sanner, A. J. Olson, and J. C. Spehner (1995). “Fast and robust computation of molecular surfaces.” In “Proc. 11th ACM Symp. Comp. Geom”, pp. C6–C7.
- M. F. Sanner (1995). “MSMS: Michel Sanner’s molecular surface”. URL http://www.scripps.edu/~mb/olson/people/sanner/html/msms_home.html.
- M. F. Sanner, A. J. Olson, and J. C. Spehner (1996). “Reduced surface: An efficient way to compute molecular surfaces”. *Biopolymers*, **38** (3): pp. 305–20.
- B. S. Schuwirth, M. A. Borovinskaya, C. W. Hau, W. Zhang, A. Vila-Sanjurjo, J. M. Holton, and J. H. D. Cate (2005). “Structures of the bacterial ribosome at 3.5 Å resolution”. *Science*, **310** (5749): pp. 827–834.
- M. Selmer, C. M. Dunham, F. V. t. Murphy, A. Weixlbaumer, S. Petry, A. C. Kelley, J. R. Weir, and V. Ramakrishnan (2006). “Structure of the 70S ribosome complexed with mRNA and tRNA”. *Science*, **313** (5795): pp. 1935–1942.
- V. Soroka, K. Kolkova, J. S. Kastrup, K. Diederichs, J. Breed, V. V. Kiselyov, F. M. Poulsen, I. K. Larsen, W. Welte, V. Berezin, E. Bock, and C. Kasper (2003). “Structure and interactions of NCAM Ig1-2-3 suggest a novel zipper mechanism for homophilic adhesion”. *Structure*, **11** (10): pp. 1291–301.
- T. C. Taylor and I. Andersson (1997). “The structure of the complex between rubisco and its natural substrate ribulose 1,5-bisphosphate”. *J Mol Biol*, **265** (4): pp. 432–44.
- T. Tenson and M. Ehrenberg (2002). “Regulatory nascent peptides in the ribosomal tunnel”. *Cell*, **108** (5): pp. 591–4.

- D. Tu, G. Blaha, P. B. Moore, and T. A. Steitz (2005). “Structures of MLS_BK antibiotics bound to mutated large ribosomal subunits provide a structural explanation for resistance”. *Cell*, **121** (2): pp. 257–70.
- F. Van Coppenolle, F. Vanden Abeele, C. Slomianny, M. Flourakis, J. Hesketh, E. Dewailly, and N. Prevarskaya (2004). “Ribosome-translocon complex mediates calcium leakage from endoplasmic reticulum stores”. *J Cell Sci*, **117** (Pt 18): pp. 4135–42.
- X. Wei, M. H. Dickman, and M. T. Pope (1997). “New routes for multiple derivatization of polyoxometalates, bis(acetato)dirhodium-11-tungstophosphate, [(PO₄)W₁₁O₃₅{Rh₂(OAc)₂}]⁵⁻”. *Inorganic Chemistry*, **36** (2): pp. 130–131. <http://dx.doi.org/10.1021/ic960991a>.
- C. A. Woolhead, P. J. McCormick, and A. E. Johnson (2004). “Nascent membrane and secretory proteins differ in FRET-detected folding far inside the ribosome and in their exposure to ribosomal proteins”. *Cell*, **116** (5): pp. 725–36.
- J. M. Word, S. C. Lovell, J. S. Richardson, and D. C. Richardson (1999). “Asparagine and glutamine: Using hydrogen atom contacts in the choice of side-chain amide orientation”. *J Mol Biol*, **285** (4): pp. 1735–47.
- M. M. Yusupov, G. Z. Yusupova, A. Baucom, K. Lieberman, T. N. Earnest, J. H. Cate, and H. F. Noller (2001). “Crystal structure of the ribosome at 5.5 Å resolution”. *Science*, **292** (5518): pp. 883–96.
- J. M. Zengel, A. Jerauld, A. Walker, M. C. Wahl, and L. Lindahl (2003). “The extended loops of ribosomal proteins L4 and L22 are not required for ribosome assembly or L4-mediated autogenous control”. *Rna*, **9** (10): pp. 1188–97. <http://dx.doi.org/10.1261/rna.5400703>.
- G. Ziv, G. Haran, and D. Thirumalai (2005). “Ribosome exit tunnel can entropically stabilize α -helices”. *PNAS*, **102** (52): pp. 18956–61. <http://dx.doi.org/10.1073/pnas.0508234102>.

Chapter 5

Progress Towards the Crystallization of Ribosomes From Animals

5.1 INTRODUCTION

5.1.1 The Differences Between Prokaryotic and Eukaryotic Ribosomes

As discussed in Chapter 1, the ribosomes of prokaryotes and eukaryotes are quite similar. Both types of ribosomes have two subunits, the same active site architecture, and catalyze an RNA-dependent protein synthesis with aminoacyl-tRNAs as substrates (Noller, 1984). However, they are not identical and their structural differences are significant even if the functional importance of these differences are not clear in every case. From a practical point of view, their structural distinctions result in differences in antibiotic resistance that can be exploited to cure bacterial infections. To date all published atomic resolution structures of the ribosome are those of the smaller prokaryotic ribosomes or their subunits, and the best eukaryotic ribosome structure available, which was produced by cryo-electron microscopy, provides no atomic details because its resolution is only 7.4 Å (Halic *et al.*, 2006). Resolution is not always directly comparable between different methods, but the need for higher resolution structures of eukaryotic ribosomes is obvious.

Composition and size

The first differences observed between eukaryotic and prokaryotic ribosomes were gross features about shape and size. Tissieres & Watson (1958) were first to recognize that the 80S ribonucleoprotein found in eukaryotes and the 70S ribosomal particles of prokaryotes are equivalent. The connection was made because both types of particles are more than half ribonucleic acid and reversibly break down into two unequal units. It is now known that the eukaryotic ribosome is almost 50% larger than its prokaryotic counterpart. Their mass ranges from 3.9 to 4.5 MDa compared to 2.6 MDa for prokaryotes. Prokaryotic ribosomes have a sedimentation coefficient of 70S, and consist of a small 30S and a large 50S subunit, whereas eukaryotes have 80S ribosomes, that consist of a small 40S and a large 60S subunit. The “additional mass” of the eukaryotic ribosome results from their having an additional rRNA molecule, longer rRNA chains, and about two dozen more proteins.

The structural differences between the two classes of particles are large enough to visualize by electron microscopy in negative stain. The profile of the eukaryotic small ribosomal subunit resembles that of a duck with three distinct visible features: a bill, platform and the lobes (Figure 5.1, bottom). While eukaryotes have all three features, archaea have only the bill and platform and eubacteria have only the platform (Lake *et al.*, 1982). Similar studies on the large ribosomal subunit identified similar differences amongst the three domains (Figure 5.1, top; Lake *et al.*, 1984).

Studies of the composition of the ribosome revealed that the eukaryotic ribosome contains several more molecular components than the prokaryotic ribosome. Table 5.1 compares the molecular composition of the ribosomes from several species. In all species, the small subunit contains one rRNA molecule, which sediments at 16S in prokaryotes, and due to increased length of ~300 nucleotides (nts), the corresponding RNA in eukaryotes sediments at 18S. In contrast, the large subunit is composed of two rRNAs in prokaryotes (23S and 5S) and three in eukaryotes (28S, 5.8S, and 5S). The additional rRNA in the 60S large subunit of eukaryotes is the 5.8S rRNA, which

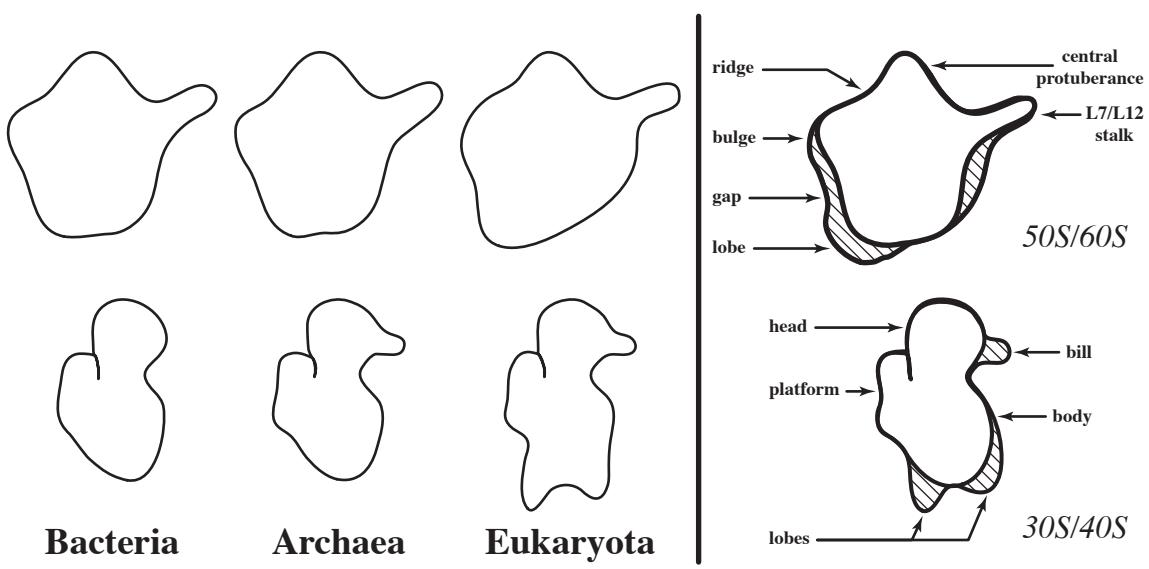


Figure 5.1: Low resolution profiles of the ribosomal subunits. Lake *et al.* (1982) characterized the major features of the small subunit and then later Lake *et al.* (1984) characterized the large subunit for three domains of cellular organisms.

is about 160 nts in length and similar in sequence to the 5'-end of the 23S rRNA (Nazar, 1980). The large rRNA of eukaryotes is generally referred to as 28S rRNA and is 300 nts (algae) to 2300 nts (human) longer than the 23S rRNA of prokaryotes (Table 5.1).

Eukaryotic ribosomes also contain several more proteins than their prokaryotic counterparts; *E. coli* has 56 ribosomal proteins (Arnold & Reilly, 1999), yeast has 78 proteins (Planta & Mager, 1998) and mammals have 79 proteins (Wool *et al.*, 1996). Despite this difference in numbers, many of the proteins are homologous between species. For rat and *E. coli*, 24 ribosomal proteins can be correlated directly and an additional 33 proteins can be related using the proteins from other organisms (Wool *et al.*, 1996). And among eukaryotic species, 65 of the 79 ribosomal proteins from rat have a homolog in yeast with over 60% sequence identity (Wool *et al.*, 1996). Taken together the additional rRNA and proteins of the eukaryotic ribosome make it much larger than its prokaryotic counterpart.

Expansion segments

The ribosomal RNAs from many species have been sequenced and their secondary structures worked out in considerable detail. Comparison of these secondary structures has shown that all large rRNAs contain a core secondary structure, which is approximately that of *E. coli* rRNA, and there are additional blocks of nonconserved sequences inserted into that core structure (Veldman *et al.*, 1981; Ware *et al.*, 1983; Clark *et al.*, 1984). The inserted sequences have sometimes been called “divergent domains” or “variable regions” or “expansion segments,” and I will refer to them as “expansion segments” (ES) following Gerbi (1996).

In 28S rRNA, there are 41 expansion segments inserted into the 23S rRNA-like core structure and in 18S rRNA there are 12 such segments augmenting its 16S-like core. The positions of the expansion segments in the secondary structures of 18S and 28S rRNA are universally conserved, but the sequences are highly variable not only between eukaryotic species, but there are also slight

species	rRNA sequence length				number of ribosomal proteins		
	23S/28S	5.8S	5S	16S/18S	whole	large	small
bacteria, <i>E. coli</i>	2904	N/A	120	1542	56	34	22
yeast, <i>S. cerevisiae</i>	3550	159	118	1800	78	46	32
brine shrimp, <i>Artemia</i>	3630	162	120	1810	—	—	—
fruit fly, <i>D. melanogaster</i>	4123	122+30	120	1995	78	46	32
rabbit, <i>O. cuniculus</i>	—	162	122	1860	—	—	—
rat, <i>R. norvegicus</i>	4943	156	121	1874	79	47	32
human, <i>H. sapiens</i>	5190	159	121	1870	79	47	32

Table 5.1: Molecular composition of ribosomes from a prokaryote and several eukaryotes. *Artemia* refers to both the North American species, *Artemia franciscana* and the European species, *Artemia salina*. The *Drosophila melanogaster* 5.8S is different and broken into two pieces, the m5.8S rRNA (122 nts) and 2S rRNA (30 nts). Ribosomal protein numbers determined by Arnold & Reilly (1999) for *E. coli*, Planta & Mager (1998) for yeast, Wool *et al.* (1996) for rat, Other numbers taken from the Ribosomal Protein Database (Nakao *et al.*, 2004) removing duplicates.

differences between rDNA repeats in the same organism (Gonzalez *et al.*, 1985; Maden *et al.*, 1987; Kuo *et al.*, 1996). These intraorganism variations exist primarily in the expansion segments and consist of insertions of small repeated motifs (Kuo *et al.*, 1996). It was also shown that the proportion of each 28S rRNA variation in existing ribosomes is maintained from tissue to tissue in humans.

The role of these expansion segments in the ribosomal structure has been debated (Houge *et al.*, 1995; Houge & Doskeland, 1996; van Nues *et al.*, 1997; Morgan *et al.*, 2000). One experiment showed that removing the largest expansion segment 27 (ES27) from large subunit rRNA affects the processing and stability of the ribosome, but when the ES27 is replaced with a sequence from another organism, ribosome function is restored (Sweeney *et al.*, 1994). Other than the removal of expansion segments affecting the processing and stability of the ribosomes, no experimental data have been obtained showing the expansion segments have a role in protein translation. The tertiary locations of several expansion segments have been mapped using electron microscopy and it was found that they are mainly on the periphery of the particle far from any functional areas, but some form intersubunit interactions (Spahn *et al.*, 2001a). Some expansion segments have peculiar structures, for example in Trypanosomes, ES6 is a long helix that protrudes from the back of the 40S subunit (Gao *et al.*, 2005). Even though every eukaryotic ribosome contains expansion segment and they have conserved locations with the sequence, the expansion segment most likely serve no function in translation.

Translational machinery

In both prokaryotes and eukaryotes, protein synthesis is accomplished by ribosomes, which bind to mRNA and tRNAs (the adapter molecules), and perform the same chemical reactions. However, eukaryotic protein synthesis uses more protein cofactors than prokaryotes and the three phases of protein synthesis: initiation, elongation, and termination, are more complicated in eukaryotes than they are in prokaryotes. The differences are highlighted below.

Initiation in prokaryotes follows three major steps. First, IF1, IF3, and the small 30S subunit form a complex and bind directly to the mRNA at the purine-rich Shine-Dalgarno sequence (Figure 5.2, top). After binding to the Shine-Dalgarno sequence, IF2-GTP and fMet-tRNA bind to this assembly and form the “30S initiation complex” (Figure 5.2, middle left). Next the 50S subunit associates with the complex causing IF2 to hydrolyze its GTP into GDP that in turn causes the release of the initiation factors. The resulting “70S initiation complex” containing both ribosomal subunits and the initial fMet-tRNA next proceeds into the elongation phase of protein synthesis (Figure 5.2, bottom left).

Eukaryotic translation initiation is far more complex than it is in prokaryotes. Eukaryotic translation does not use a specific sequence to distinguish initiation AUG codons from internal AUG codons nor do eukaryotes synthesize polycistronic mRNAs. Instead, all eukaryotic mRNA are monocistronic and translation begins at the AUG nearest to the 5'-end of the mRNA. A 40S subunit initiation complex with eIF4, eIF3, eIF2-GTP, and Met-tRNA forms at the cap structure that is found at the 5'-end of most eukaryotic mRNA (Figure 5.3, top). The 40S complex then scans down the mRNA to the first AUG start codon propelled by ATP-driven helicases (Figure 5.3, middle left). After the start codon is found, the 60S subunit joins the complex and through GTP hydrolysis the initiation factors are released (Figure 5.3, bottom left).

Both the elongation and termination steps of protein synthesis are more similar in prokaryotes and eukaryotes than the translation initiation step. Each set of protein cofactors for prokaryotes has an analogous set of protein cofactors in eukaryotes. Prokaryotes use elongation factors, EF-Tu and EF-Ts; eukaryotes use elongation factors EF1 α , which is homologous to EF-Tu, as well as EF1 β , EF1 γ , and EF1 δ , which together play the same role as EF-Ts. The EF1 α -GTP complex delivers the aminoacylated-tRNA to the active site of the ribosome and the EF1 $\beta\gamma\delta$ complex hydrolyzes the GTP into GDP. As EF-G does in prokaryotes, eukaryotic EF2 mediates GTP-driven translocation of the tRNAs. Release factor, eRF1 carries out termination in eukaryotes taking the place of both

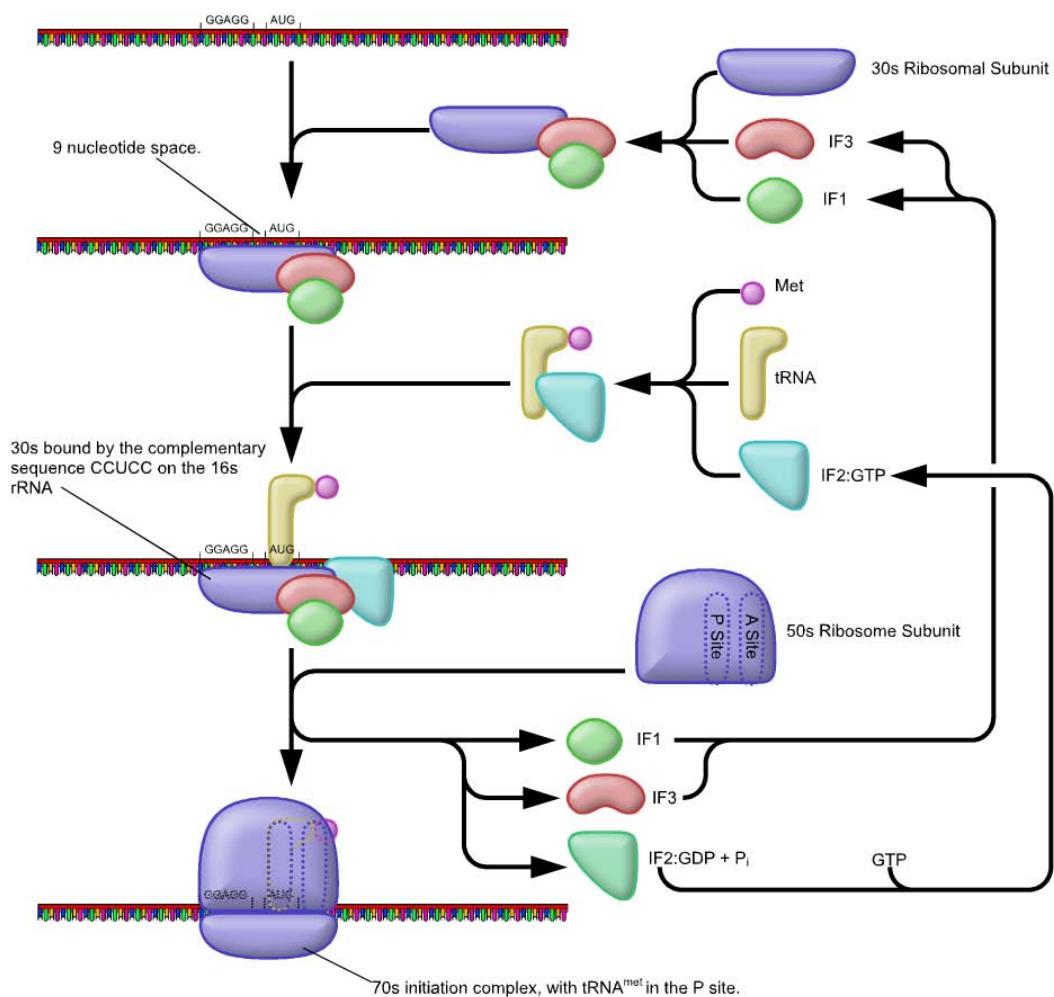


Figure 5.2: The process of initiation of translation in prokaryotes. Figure by Wheeler (2005b).

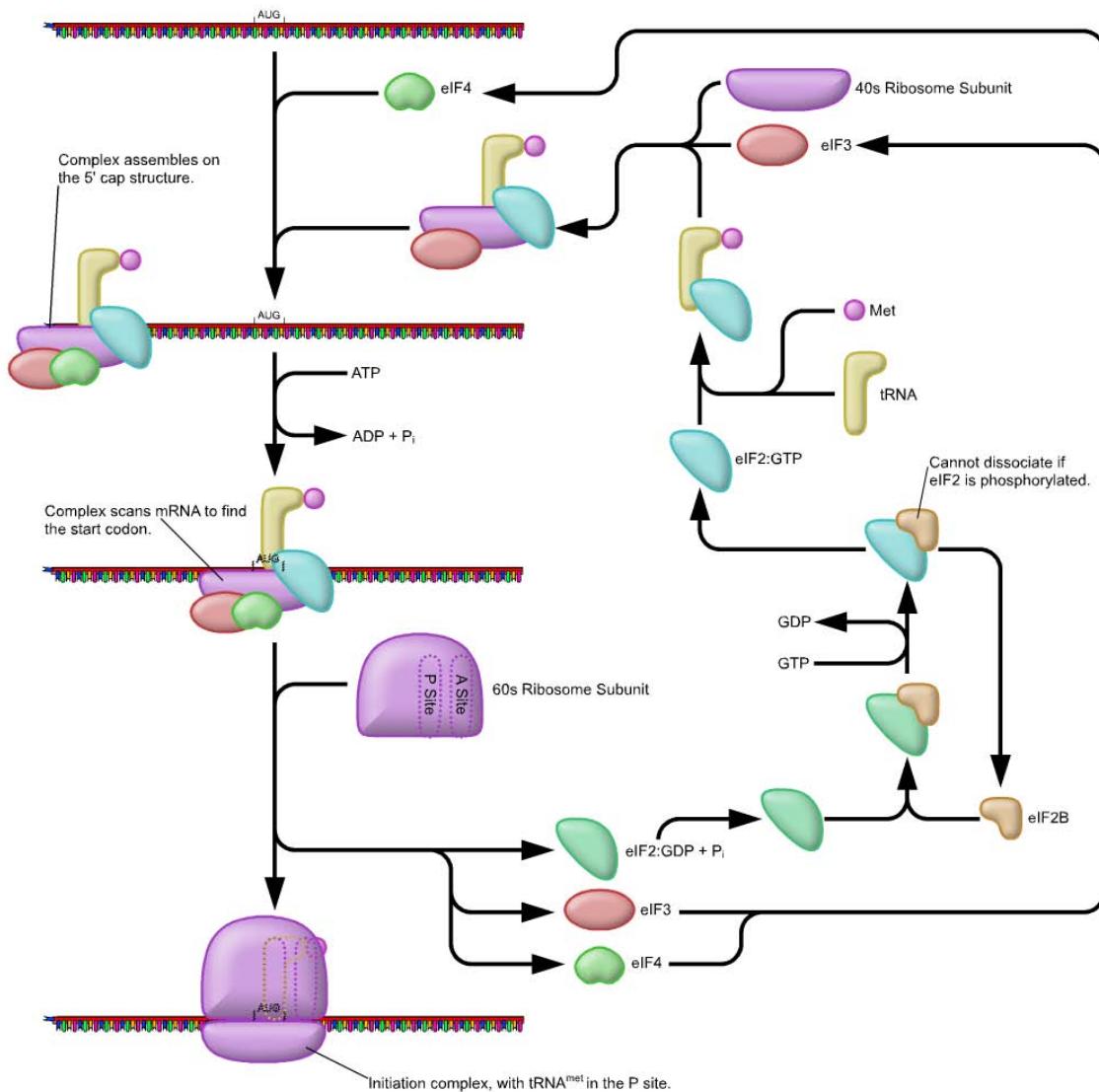


Figure 5.3: The process of initiation of translation in eukaryotes. Figure by Wheeler (2005a).

release factors, RF1 and RF2 of prokaryotes. Reassociation of ribosomal subunits is then prevented by eIF3 in eukaryotes, much like IF3 does in prokaryotes.

5.1.2 Ribosomal Crystallography and Structure

Prokaryotes

Efforts to crystallize ribosomes began in the 1960s with the first, large, three-dimensional crystals being reported in 1980 (Yonath *et al.*, 1980). Over the following decade A. Yonath and H. G. Wittmann produced over a dozen different crystals of complete particles and subunits from four different species: *Bacillus stearothermophilus* (Yonath *et al.*, 1980), *Escherichia coli* (Wittmann *et al.*, 1982), *Haloarcula marismortui* (Shevack *et al.*, 1985), and *Thermus thermophilus* (Glotz *et al.*, 1987). During this same period, Trakhanov *et al.* (1989) also independently produced crystals from *Thermus thermophilus*.

All the crystals were produced using vapor diffusion, but involved different solvents. The first ribosomal crystallizations used organic solvents and alcohols, such as methanol, methylpentanediol (MPD), and ethylene glycol (EG) as precipitants (Yonath *et al.*, 1982). Efforts were then made to move away from organic solvents due to the difficulty of handling of any resulting crystals (Shevack *et al.*, 1985). Eventually, polyethylene glycol (PEG), which is commonly used today was found to be a more practical precipitant (Yonath & Wittmann, 1988).

As discussed in Chapter 1, the first atomic resolution structures of ribosomes and ribosomal subunits appeared only recently. Based on the crystals from the 1980s, several structures of ribosomal subunits from *H. marismortui* and *T. thermophilus* were solved at high resolution between 2000 and 2001 (Ban *et al.*, 2000; Schluenzen *et al.*, 2000; Wimberly *et al.*, 2000). Over the past two years, complete 70S ribosome structures have been solved from *E. coli* (Schuwirth *et al.*, 2005) and *T. thermophilus* (Selmer *et al.*, 2006) at much higher resolutions than previous 70S structures.

Eukaryotes

Ordered two-dimensional arrays of ribosomes have been observed in several different eukaryotic organisms. Ribosomes in hypothermic chicken embryos (Byers, 1966), lizard oocytes during winter hibernation (*Lacerta sicula*, Unwin & Taddei, 1977), mouse oocytes (Burkholder *et al.*, 1971), amoebas (*Entamoeba histolytica*, Kress *et al.*, 1971), water strider oocytes, (*Gerris najas*, Choi & Nagl, 1977) and the brains of senile humans (O'Brien *et al.*, 1980) will all form ordered two-dimensional sheets *in vivo*, often with P4 symmetry. These sheets have been isolated from living tissue (Barbieri *et al.*, 1970), and also have been formed *in vitro* from chicken embryo material (Milligan & Unwin, 1982). Under certain conditions, the ordered sheets of ribosomes found in hypothermic chicken embryos will stack to form a three-dimensional crystal with P422 symmetry (Byers, 1967). Nevertheless, these *in vivo* ribosome arrays have never given rise to macroscopic three-dimensional crystals suitable for X-ray diffraction.

All the structures available for eukaryotic ribosomes have been produced by cryo-electron microscopy. Table 5.2 lists of all three-dimensional eukaryotic ribosome structures produced this way to date using this technique. While the resolutions of the more recent structures have sub-nanometer resolutions, progress has been slow, and atomic resolution ($\sim 3.5 \text{ \AA}$ or better) is still well out of reach.

The resolution criteria of X-ray and cyro-electron microscopy are very different. Ban *et al.* (1998) showed that a 20 \AA resolution X-ray map contains significantly more detail than a 20 \AA resolution EM map. On the other hand, Valle *et al.* (2003) found that their 9 \AA resolution EM map was significantly better than any other 9 \AA EM maps and could see rearrangements to the shape of the tRNA molecules. For both methods resolution alone does not always correlate to the quality of the electron density. In X-ray crystallography, good phases can make up for poor resolution, but bad phases can ruin good resolution. Likewise in cyro-electron microscopy, a bad initial model can also give incorrect results under certain conditions.

source	particle	resolution	EMD id	citation
rabbit retic	40S	38.5†	—	Verschoor <i>et al.</i> (1989)
rabbit retic	80S	37†	—	Verschoor & Frank (1990)
rabbit retic	40S	55†	—	Srivastava <i>et al.</i> (1995)
wheat germ	80S	38†	—	Verschoor <i>et al.</i> (1996)
yeast	80S	35	—	Verschoor <i>et al.</i> (1998)
yeast	80S	26	—	Beckmann <i>et al.</i> (1997)
rat liver	80S	25‡	—	Dube <i>et al.</i> (1998a)
rabbit retic	80S	21‡	—	Dube <i>et al.</i> (1998b)
yeast	80S	17.5	—	Gomez-Lorenzo <i>et al.</i> (2000)
yeast	80S	24	—	Morgan <i>et al.</i> (2000)
rabbit retic	80S	24	—	Morgan <i>et al.</i> (2000)
canine pancreas	80S	27	—	Menetret <i>et al.</i> (2000)
rabbit retic	40S	19.8	—	Spahn <i>et al.</i> (2001b)
yeast	80S	15.4	1076	Spahn <i>et al.</i> (2001a)
canine pancreas	80S	17.5	—	Morgan <i>et al.</i> (2002)
wheat germ	80S	12.0	1063	Halic <i>et al.</i> (2004)
yeast	80S	11.7	1067	Spahn <i>et al.</i> (2004a)
human	80S	18.3	1093	Spahn <i>et al.</i> (2004b)
wheat germ	80S	9.5	1125	Halic <i>et al.</i> (2005)
canine pancreas	80S	10.0	—	Menetret <i>et al.</i> (2005)
trypanosome	80S	12.0	—	Gao <i>et al.</i> (2005)
green algae	80S	25	—	Manuell <i>et al.</i> (2005)
canine pancreas	80S	20	1168	Blau <i>et al.</i> (2005)
human	80S	25.0	1138	Boehringer <i>et al.</i> (2005)
rabbit retic	80S	14.0	1199	Namy <i>et al.</i> (2006)
wheat germ	80S	7.4	1217	Halic <i>et al.</i> (2006)
yeast	80S	9.9	1233	Andersen <i>et al.</i> (2006)

Table 5.2: Progress of three-dimensional eukaryotic ribosome structures obtained by cryo-electron microscopy. Horizontal lines separate five year periods. “rabbit retic” stands for rabbit reticulocytes. The species for sources listed in the table are: yeast, *Saccharomyces cerevisiae*; human, *Homo sapiens* HeLa cells; trypanosomes, *Trypanosoma cruzi*; green algae, *Chlamydomonas reinhardtii*. Entries are sorted by date. †denotes that the 3D differential phase residual criterion was used for resolution (Frank *et al.*, 1981; Radermacher *et al.*, 1987). ‡denotes that the 3σ threshold criterion was used for resolution (Harauz & van Heel, 1986). All other resolutions values were calculated using the Fourier shell correlation method with a cutoff value at 50% (Bottcher *et al.*, 1997; Penczek, 1998).

5.1.3 Concluding Remarks

The goal of the work described in this chapter is the isolation, purification, crystallization and structure determination of the ribosome or ribosomal subunits from higher eukaryotes. Provided the crystals obtained are well-ordered, the structures obtained this way could be far superior to electron microscopy in terms of resolution (Ban *et al.*, 1998). An atomic structure of a eukaryotic ribosome would make it possible to understand in detail how and why these particles differ from their prokaryotic counterparts both structurally and functionally.

5.2 RESULTS AND DISCUSSION

5.2.1 System Selection

The first step in any crystallization project is to select a source for the macromolecule to be examined. Ribosomes are abundant in all cells and therefore do not require overexpression. Thus for higher eukaryotes, almost any species, tissue and/or cell type could be used. Historically, the first preparations of animal ribosomes were obtained from bovine pancreas (Keller & Cohen, 1961; Dickman *et al.*, 1962; Keller *et al.*, 1963, 1964). However, pancreatic tissue, as with many other animal tissues, often contains significant quantities of ribonucleases, which can degrade ribosomes. Bovine pancreas was the standard source until Zendzian & Barnard (1967) pointed out that the bovine pancreas has a higher yields of ribonuclease than any other organism (Table 5.3). Since this discovery other mammalian tissues have been used for ribosomal preparations used (Beeley *et al.*, 1968; Keller *et al.*, 1968).

As Table 5.3 shows, bovine pancreas has a large amounts of ribonuclease (Zendzian & Barnard, 1967). On the other hand, canine pancreas had the smallest amount of ribonuclease of many organisms (Table 5.3) and therefore it has become the preferred source of ribosomes from animals (Walter & Blobel, 1983; Scheele, 1983). Unfortunately, canine pancreas has become difficult to

Common name	RNase Yield, ($\mu\text{g/g tissue}$)
Sting ray	5
Tuna	2
Frog	17
Chicken	20
Cow (<i>bovine</i>)	1000
Goat	600
Rabbit	0.5
Dog (<i>canine</i>)	0.5
Cat (<i>felinae</i>)	0.5
Rat	220
Human	1

Table 5.3: Distribution of pancreatic ribonuclease A in vertebrates. Ribonuclease (RNase) yield is expressed as the equivalent quantity of bovine RNase A as determined from the activity in the supernatant fluids of pH 5 homogenates. Higher amounts of RNase in an organism is bad for ribosome preparations. Table reproduced from Zendzian & Barnard (1967).

obtain due to the activities of the animal rights group, PETA, which have caused suppliers to leave the business. As an alternative to canine, rabbit pancreas was used; rabbit pancreas remains readily available and is comparably low in ribonuclease activity (Table 5.3).

Rabbit pancreas

Rabbit pancreas was obtained from Pel-Freez Biologicals. A number of published protocols are available on the preparation of ribosomes from rabbit pancreas (Perkins & Pandol, 1992; Keller *et al.*, 1968), however, neither of them describe a protocol for preparing subunits. In general, subunits are obtained by simply reducing the magnesium concentration in the buffer of whole ribosomes causing them to fall apart. For my sample, any attempts to produce subunits using sucrose gradients failed even in the absence of magnesium. This may be a result of the low concentration of monovalent salt in the buffers used (~ 50 mM), which should cause the subunit to dissociate. Further optimization of the monovalent salt concentration, possibly above 200 mM, as is done for other ribosomal preparations, which might yield better results. Additionally, the rabbit pancreas contains large amounts of fatty tissue (Figure 5.4), which renders ineffective many of the standard tools for breaking cells, *e.g.* the French press and microfluidizer and so in general Dounce homogenizers are commonly used (Perkins & Pandol, 1992; Keller *et al.*, 1968). Yields of 80S particles were inconsistent and typically very low (<0.1 mg ribosomes per g cells). After several unsuccessful attempts to obtain enough material for crystallization trials, a different source for ribosomes was investigated.

Rabbit reticulocytes

Reticulocytes are immature red blood cells that appear in the blood when animals are recovering from anemia. Typically, reticulocytes comprise about 1% of the red blood cells in the body, but they are much more abundant in the blood of anemic rabbits. Reticulocytes have two major advantages over pancreas as a source of ribosomes. First, the reticulocytes are free of fatty tissue and hence

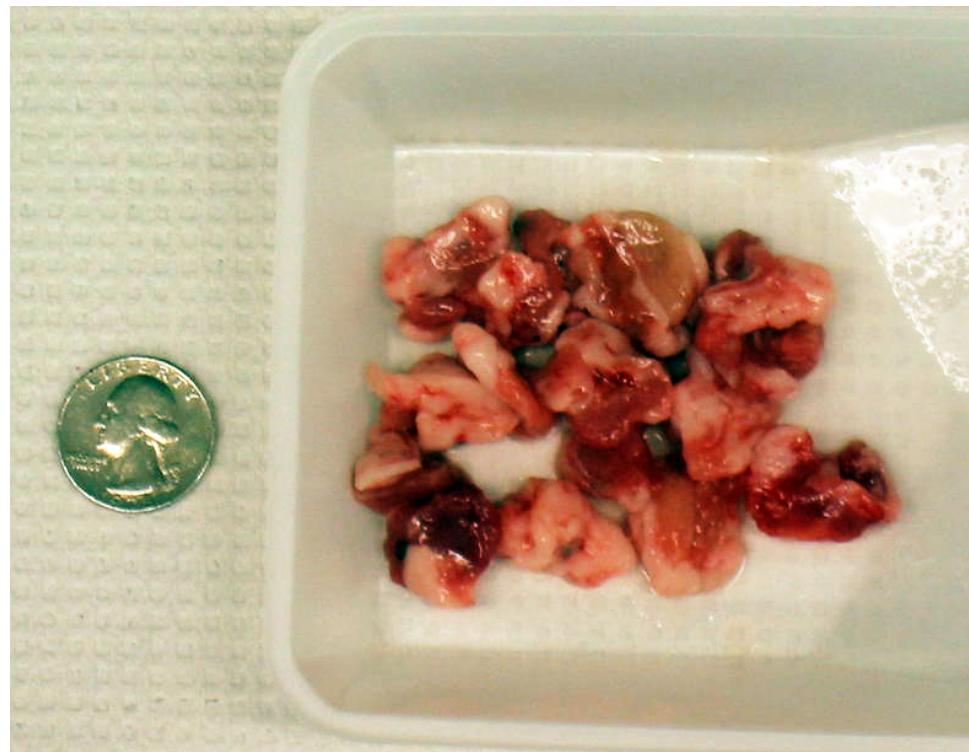


Figure 5.4: Image of several whole rabbit pancreases. This image shows several (about 15) pancreases taken from a young New Zealand white rabbit. The fatty tissue is the white material on many of the pancreases. An American quarter is shown for size comparison.

much easier to handle. Second, reticulocyte ribosomes have been well studied (Hardesty *et al.*, 1971), and are commonly used in tissue culture experiments, which might make genetic experiments possible in the future. Presumably, reticulocytes also make only a few types of proteins, unlike the pancreas, which makes many and therefore should have a more homogenous translation system. The secondary structure for the reticulocyte ribosome 18S rRNA has been worked out (Raikar *et al.*, 1988) and its tertiary structure has been solved at low resolution (currently 14.0 Å) by cryo-electron microscopy (Dube *et al.*, 1998b; Morgan *et al.*, 2000; Spahn *et al.*, 2001b; Namy *et al.*, 2006).

Rabbit reticulocytes are also available from Pel-Freez Biologicals, however unlike the rabbit pancreas that can be frozen at -80°C, ribosomes from reticulocytes must be processed as soon as they are obtained to prevent spoilage. Preparation of ribosomes from reticulocytes is similar to their preparation from rabbit pancreas, however, the ribosomes produced need to be dialyzed before crystallization. This is because the phosphate buffers used to purify the reticulocytes from the raw serum could lead to the formation of magnesium ammonium phosphate crystals. Another drawback to reticulocytes is that their ribosomes tend to form polysomes resulting in a large loss of yield during sucrose gradient purification. The ribosome yield obtained was again insufficient ($\gg 10$ mg) for crystallization trials, so a new source for ribosomes was investigated.

Brine shrimp

The brine shrimp (*Artemia*) system has four main advantages over either of the rabbit systems: (1) *Artemia* cysts can be stored at 4°C indefinitely, primarily because they are protected by a chitinous outer shell that is removed by washing with bleach (Figure 5.5B,C). Neither rabbit pancreases nor reticulocytes can be stored without loss. (2) The resulting ribosome preparations are completely free of polysomes because the ribosomes in cysts are at rest and free of mRNA (Tate & Marshall, 1991). In contrast, both rabbit tissues contain active cells that are expressing proteins during purification, which leads to polysome formation. (3) Using a protocol from Zasloff & Ochoa (1974), the ribosomal material is simple to obtain and use. (4) In rabbit cells, most active ribosomes

are attached to the membrane for protein excretion, so detergents must be added to preparations to dissociate the ribosomes from the membranes, and any detergent micelles must be removed from preparations before crystallization trials are done. All of these factors combine to make the ribosome yields from brine shrimp much higher than those from either of the rabbit systems.

Characteristics of brine shrimp, *Artemia*

Brine shrimp of the genus, *Artemia*, are aquatic crustaceans that are more closely related to zooplankton than to true shrimp. *Artemia* are found throughout the world in environments where the salt concentrations are high (0.3–4.0M), such as salt lakes and salt evaporation flats. More than 50 different strains are known to exist that differ in characteristics such as hatching rate, size, viability, optimal temperature and salinity range requirements. There are mixed views on whether the several varieties of *Artemia* should be thought of as separate species. Nonetheless, *Artemia franciscana* is the name attached to major type of brine shrimp harvested in the North America, while *Artemia salina* is the name given to the dominant strain in Europe (Eriksen & Belk, 1999).

The eggs of *Artemia* (called cysts) are available at most aquarium shops because it is a popular type of fish food (Figure 5.5A). *Artemia* consume primarily micro-algae, but will also eat protozoa, yeast, or organic detritus particles. *Artemia* produces cysts when the body of water they live in dries out, *i.e.* when the salt level in the water increases above a critical point. Unhatched *Artemia* cysts have no measurable metabolism and are known to remain viable for at least 50 years, but may persist for up to 1000 years under dry, oxygen-free conditions. Encysted *Artemia* embryos contain large amounts of 80S ribosomes (Golub & Clegg, 1968; Hultin & Morris, 1968), competent mRNA (Nilsson & Hultin, 1974; Grosfeld & Littauer, 1976; Sierra *et al.*, 1976), elongation factors (Slobin & Möller, 1976), termination factors (Reddington *et al.*, 1978), and tRNA synthetases (Bagshaw *et al.*, 1970). In dormant cysts, ribosomes are not found in polysomes, but polysomes do form within 15 minutes after the hydration of the cysts (Moreno *et al.*, 1991; Hultin & Morris, 1968; Golub & Clegg, 1968). It is thus presumed that the ribosomes in cysts are free of mRNA, tRNA,

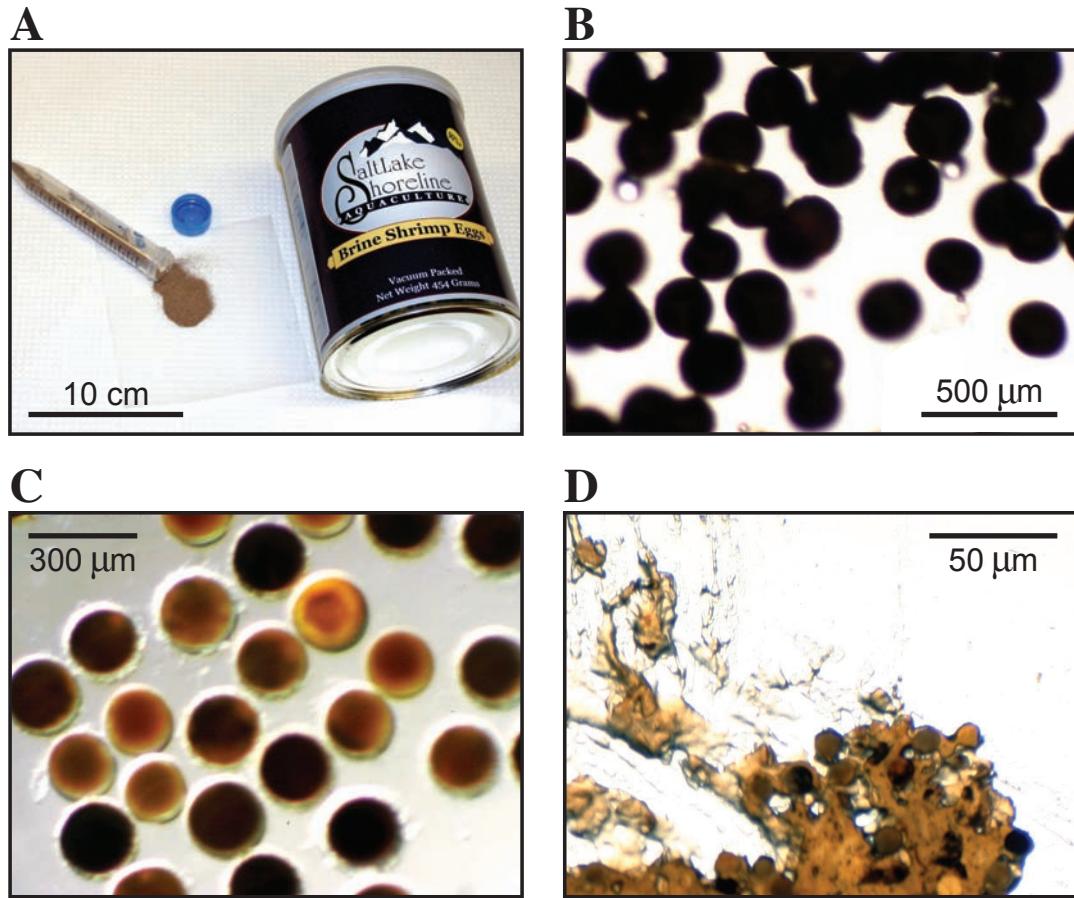


Figure 5.5: Pictures of brine shrimp, *Artemia*, cysts. (A) A picture of the container the *Artemia* cysts come in. (B) Image of cysts taken under a light microscope before decapsulation. (C) Image of cysts after decapsulation. (D) Image of cysts after being ground in a mortar and pestle.

and protein cofactors. Once placed in water, the cysts hatch within a few hours, and grow to a mature length of around one centimeter on average. Due to their availability and ease of handling, I used *Artemia franciscana* cysts as a source of eukaryotic ribosomes for crystallization.

5.2.2 Purification

Ribosomal purification has 4 major steps: (1) cell breakage, (2) removal of cellular detritus by centrifugation, (3) pelleting of the ribosomes, and (4) purification of crude ribosomes on sucrose gradients. In general, steps 2-4 are similar for all ribosomal preparations, but step 1 varies depending on the source and which type of tissue is used.

Breaking of *Artemia* cysts

Artemia cysts are difficult to break. Zasloff & Ochoa (1974) used a mortar and pestle for this purpose (Figure 5.5D), but I found this technique difficult. If too much buffer was used the cysts could not be broken at all using ceramic or glass mortars and pestles. If the amount of buffer was minimized the cysts did break, but it was hard work to obtain enough sample for ribosome crystallization. Sonication was tried, but was found not to cause cysts to break. Other cell breakage devices, such as the french press and the microfluidizer did break the cysts, but the yolk from the cysts tended to gum up the apparatus, which made the use of these devices for this purpose very tedious.

Eventually, it was discovered that Dounce homogenizers, which were used to break up the rabbit tissues, worked well for *Artemia* cysts and provide the highest yield of all techniques tried. However, this method requires the application of so much brute force that many might find it a challenge. In the end, I relied on a Ultra-Turrax T-25 motorized tissue homogenizer from IKA Works, which spins two circular blades in opposite directions and breaks the cells not only with a blender action, but by the torsion forces between the blades. The motorized tissue homogenizer turned out to be the best compromise between effort and yield (Table 5.4).

breakage method	Yield, (mg/g)	
	post-pellet	post-grad
Bead Beater	0.0	—
Blender	0.0	—
Dounce Homogenizer	2.0-4.1	0.4-1.0
French Press	0.8-1.6	0.4-1.0
Microfluidizer	0.6	0.3
Mortar & Pestle (<i>sand</i>)	2.7	—
Mortar & Pestle (<i>alumina</i>)	0.3-0.4	—
Sonication	0.0	—
Tissue Homogenizer	1.5-3.8	0.3-1.2
Zasloff & Ochoa (1974)	3.2	—

Table 5.4: Ribosome yield for different cell breaking techniques. All values are given in milligrams (mg) of ribosomes (calculated by absorption at 260 nm) per gram (g) of dry cysts. “post-pellet” stands for the amount of ribosomes collected from pelleting by ultracentrifuge. “post-grad” stands for the amount of ribosomes collected from the sucrose gradient. Dashed lines indicates that gradients were not run.

Buffer conditions for ribosomal subunits

Aggregation of ribosomal subunits proved to be a serious problem. Two other groups have reported similar findings (Nieuwenhuysen & Clauwaert, 1981; Goss & Harrigan, 1986; Goss *et al.*, 1988). Nieuwenhuysen & Clauwaert (1981) found that while the buffer used for separation of subunits by Zasloff & Ochoa (1974) calls for 700 mM monovalent salt, they found that ribosomes did not aggregate when the monovalent salt concentration was lowered to 400 mM for sucrose gradients and 250 mM otherwise. In addition to aggregation, subunit stability that also depended on buffer conditions, became an area of concern for the electron microscope studies that are described later (see page 160).

Further purification by hydrophobic interaction column

In an effort to further purify the ribosomal sample, material obtained from sucrose gradients was chromatographed on a Resource-PHE hydrophobic-interaction column (HIC) from Amersham Biosciences. An HIC column works, in essence, by separating macromolecules based on their surface area. Macromolecules with large surface areas travel through the column slower than those with smaller surface areas. This technique should therefore be able to distinguish ribosomes with protein cofactors bound from those without based on their differences in surface area.

When the 80S ribosomes were run over the HIC column, four broad overlapping peaks were observed (Figure 5.6A). In this kind of chromatography, samples are loaded onto the column in high chaotropic salt buffers and eluted by gradually lowering the salt concentration. In this case, the first peak eluted from the column at ammonium sulfate concentration of about 1.1 M and the last peak had completed its elution at a concentration of 0.3 M. Subsequent protein gels showed that all four peaks contained 80S ribosomes. Sucrose gradients also confirmed that 80S ribosomes remain intact in 1.5 M ammonium sulfate. Next, it was necessary to determine whether the column was separating ribosomes that differ in protein or RNA composition or if it was merely separating different,

interconvertible states of the ribosomes. To address this, each peak was collected, concentrated, and passed over the column again. The results of the second pass over the column are shown in Figure 5.6B,C. Each peak isolated from the first pass over the column came back as a spectrum of all four peaks with at most 36% eluting at its original position. This suggests that the four peaks are compositionally equivalent and possibly differ only in configuration. Following this experiment, no further purification via hydrophobic interaction columns were attempted.

5.2.3 Assessing Sample Quality

Sucrose gradient profiles

Sucrose gradient profiles can provide important information about ribosome purifications. Figure 5.7 shows the sucrose gradient profiles for my *Artemia* ribosome preparations. Under association conditions (normal magnesium and a low concentration of monovalent ions), 80S particles migrate as a single peak (Figure 5.7A). Under dissociation conditions (lowered magnesium and a high concentration of monovalent ions), the 80S particles should dissociate into 60S and 40S subunits. For *Artemia*, if the initial ribosomal pellet is directly loaded onto gradients in dissociation buffer, a 20S peak is observed as well as the expected 60S and 40S peaks (Figure 5.7B), but if the 80S ribosomes collected from an association gradient are loaded under the same conditions the 20S peak is not observed (Figure 5.7C). The 60S and 40S peaks from these gradients had A_{260}/A_{280} ratios of 2.10-2.15 with characteristic ribosome absorption curve shapes suggesting the presence of large amounts of RNA. On the other hand, the 20S peak collected from the gradient had an A_{260}/A_{280} ratio of 1.2-1.3 and did not show the typical ribosome absorption curve, but more of a flat curve. Thus, the 20S peak is relatively protein-rich.

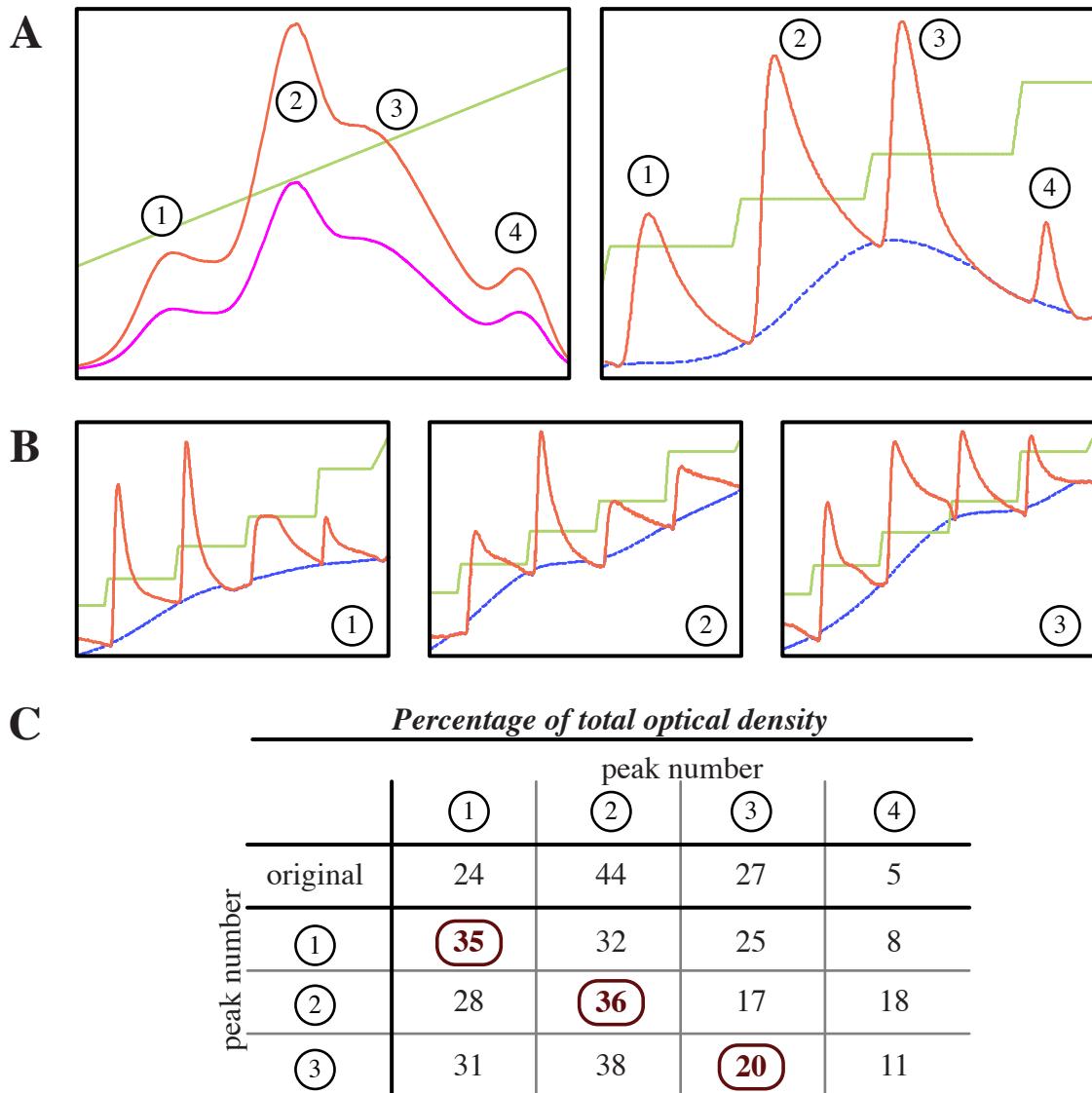


Figure 5.6: FPLC profile for *Artemia* 80S ribosomes. (A) Original FPLC trace (right) showing four peaks and the forced peaks FPLC trace (left). (B) Results from running peaks 1, 2, and 3 over the FPLC again. The circled number indicates which peak was loaded from part A. (C) A table showing the percentage of area under the peaks for each peak after each run. The self-retention percentages are circled and bold. In parts A and B, the colors are orange for absorption at 260 nm, pink for absorption at 280 nm (only in part A, left), blue for the base line use to calculate area under the curve, and yellow for the buffer concentration, the x-axis is a function of time.

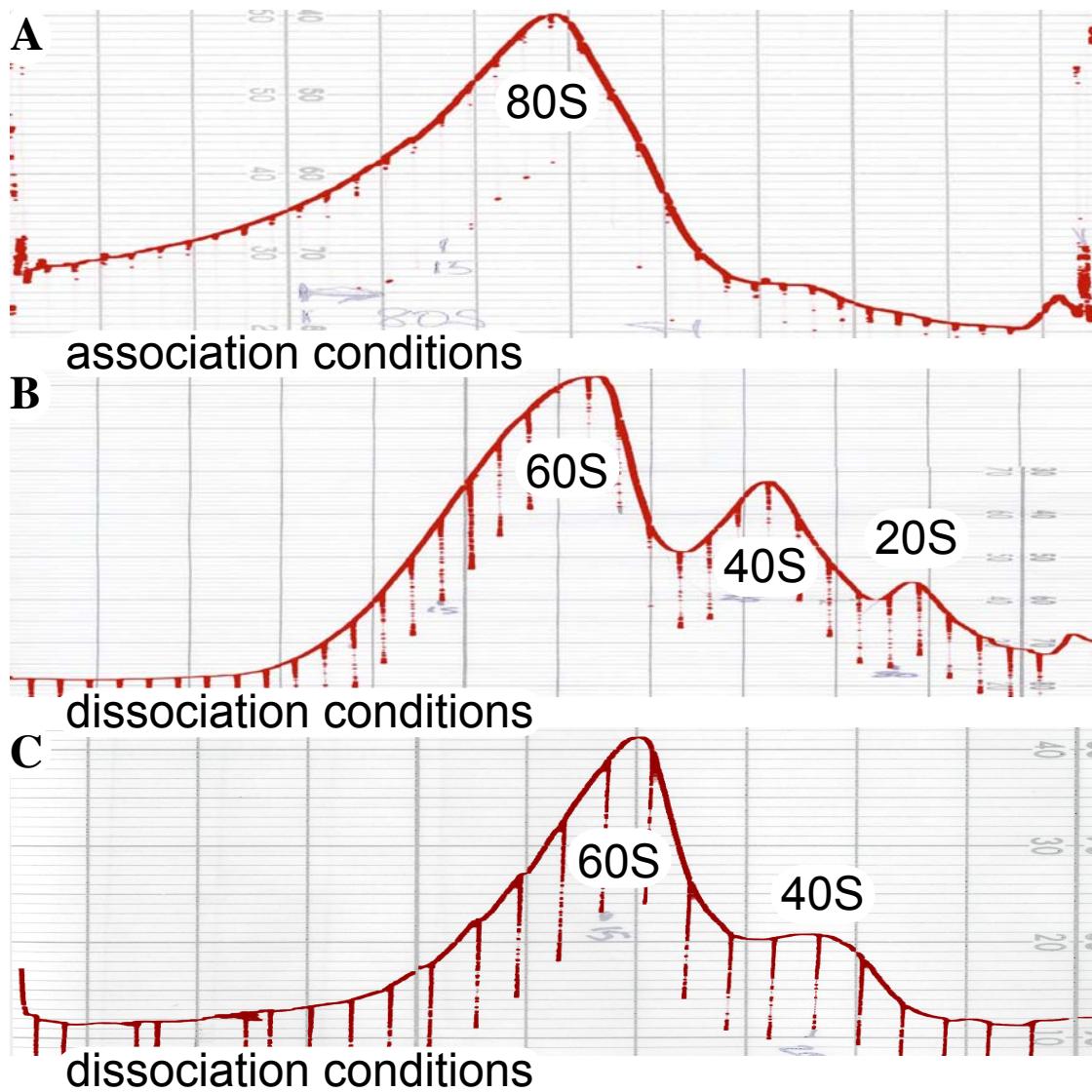


Figure 5.7: Sucrose gradient profiles for *Artemia* ribosomes. The bottom of the gradient (large macromolecules) is on the left and top (small molecules) is on the right. (A) Profile of 80S ribosomes under association conditions. (B) Profile of subunits under dissociation conditions from raw ribosomal pellet. (C) Profile of subunits under dissociation conditions from previously isolated 80S ribosomal fractions.

The 20S peak: artemin and p26

In *Artemia* cysts there are two abundant large protein complexes having molecular weights of 700 kDa that sediment at about 20S. These protein complexes, artemin and p26, which each comprise about 10-15% of the total non-yolk protein in *Artemia* cysts (De Herdt *et al.*, 1979; Clegg *et al.*, 1994), help preserve the organism in dry conditions, and are not present in the adult *Artemia* (Crack *et al.*, 2002).

These proteins complexes are very different, both functionally and structurally. Artemin is a 24-mer protein complex that sediments at 19S and is noted for its RNA binding capacity (De Herdt *et al.*, 1979). Each artemin monomer consists of 229 amino acids, has a molecular mass of 26.0 kDa, and has high sequence similarity to ferritin, an iron transport and storage protein complex found in vertebrates (De Graaf *et al.*, 1990). Electron micrographs of artemin show a uniform, rosette-like complex, similar to that of ferritin (De Graaf *et al.*, 1990). However, despite its many similarities to ferritin, artemin does not seem to be associated with iron (De Graaf *et al.*, 1990; Chen *et al.*, 2003). RNA bound to artemin is highly stable, resisting destruction after 30 minutes at 90°C, and hence it has been suggested that artemin could be a RNA molecular chaperone that protects stored mRNA within the cysts (Warner *et al.*, 2004).

The other 20S protein complex, p26 is similar in structure to α -crystallin, a small heat shock protein (Liang *et al.*, 1997a; Clegg *et al.*, 1994). In general, α -crystallin proteins are 12-43 kDa in size (Qiu *et al.*, 2004) and contain a conserved domain of about 100 amino acids with flanking variable regions. Despite their small size, these proteins have the ability to form large particles by oligomerizing through β -sheet interactions (Chen *et al.*, 1994). The p26 monomer consists of 191 amino acids and has a molecular weight of 20.7 kDa (Liang *et al.*, 1997a). p26 commonly forms an oligomer of about 700 kDa (Liang *et al.*, 1997b), which is about 27 monomers. p26 is known to be a molecular chaperone (Sun *et al.*, 2004), preserves cysts against desiccation (Ma *et al.*, 2005),

and protects mammalian cells against oxidative damage when introduced into cultured cells (Collins & Clegg, 2004).

Ribosomal protein analysis

A second way to assess ribosome preparations is to examine their protein content on gels. A protein gel profile allows the comparison of different preparations for consistency and helps identify any non-ribosomal proteins having abnormally large molecular weights that could prevent crystallization.

For *D. melanogaster*, the closest relative to brine shrimp with a sequenced genome, ribosomal proteins range in size from 6.2 kDa (L40) to 46.8 kDa (L3). An SDS-PAGE gel of *Artemia* ribosomal proteins (Figure 5.8) shows that they range in molecular weight between 6 and 45 kDa, and that ribosomal protein is a dominant component of total yolk protein. The two largest bands in the 80S lane are about 48 kDa and 56 kDa. While the 48 kDa band could correspond to the 46.8 kDa L3 protein or 45.0 kDa L4 protein (or both), the 56 kDa protein is not so easily assigned. Many eukaryotic initiation factors (eIFs) fall within this range, for example eIF3 in *D. melanogaster* has a calculated mass of 52 kDa. The acidic large ribosomal proteins P0, P1, and P2 of *Artemia* are known to oligomerize to form a complex similar to the L10 ribosomal protein complex from *E. coli* (Uchiumi *et al.*, 1987). Thus it appears likely that my *Artemia* 80S ribosome preparations contain proteins other than ribosomal proteins.

rRNA analysis

It is important for all ribosome preparations to determine the degree to which nucleolytic degradation has occurred. The best way determine the amount of degradation in a sample is to evaluate the rRNAs using denaturing gel electrophoresis.

There are 4 rRNA in the 80S ribosomal particle of *Artemia* (Table 5.1). The 5S and 5.8S rRNAs are small compared to the 18S and 28S rRNAs and are not detectable on gels of mixed rRNAs

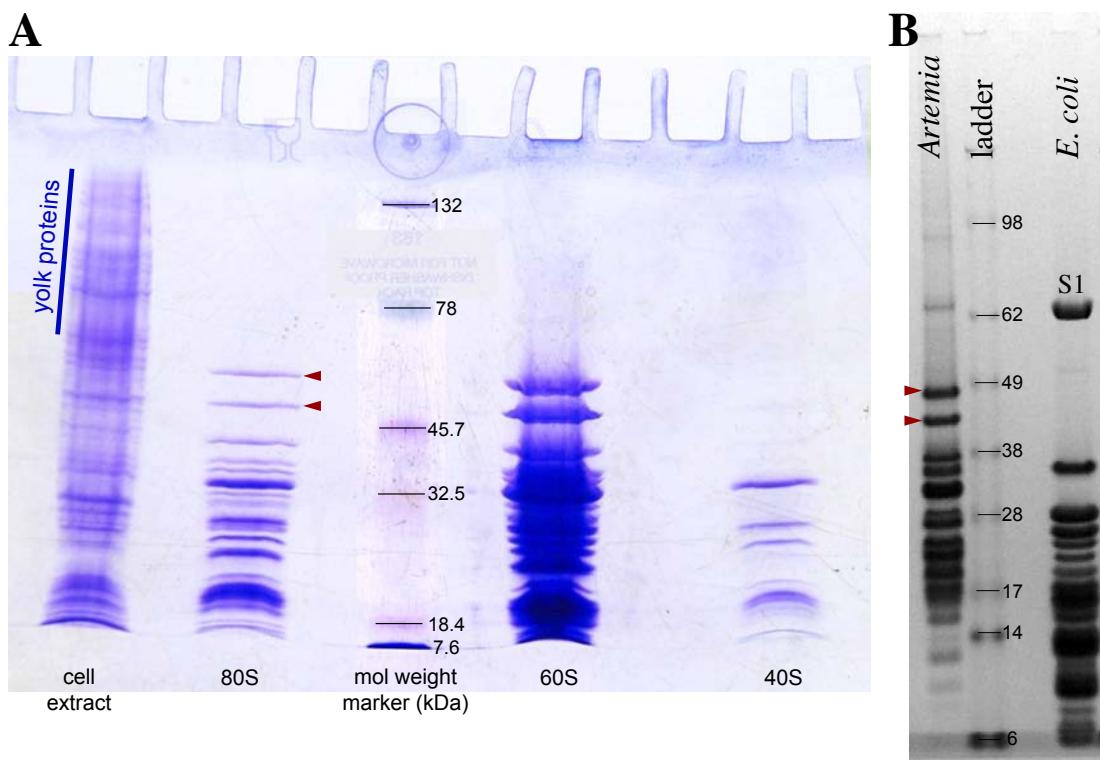


Figure 5.8: Acrylamide gel of the *Artemia* ribosomal proteins. (A) The ribosomal proteins from cell extract, 80S ribosomes, 60S subunits, and 40S subunits from *Artemia*. The two largest proteins in the 80S lane are approximately 56 kDa and 48 kDa as marked with arrows. Yolk proteins characterized by Clegg *et al.* (1994) are labeled on the far left. The 48 kDa band falls within the accepted range of molecular weight for the proteins; the 56 kDa band does not. Bio-Rad Criterion Tris-HCl 10.5–14% Gradient Gel with Coomassie blue stain was used. (B) Ribosomal proteins from *Artemia* 80S ribosomes and *E. coli* 70S ribosomes. Bio-Rad Criterion XT gel produced by Gregor Blaha using Invitrogen SeeBlue Plus2 standard molecular weight markers.

unless they are overloaded for 18S and 28S rRNAs. The sequences for 5S, 5.8S, and 18S rRNAs for *Artemia* are known in their entirety and are 120, 162, and 1810 nucleotide (nts) in length (Table 5.1). But for *Artemia* 28S rRNA, only a 3628 nt partial sequence is available (Mallatt *et al.*, 2004). Based on alignments with several known 28S rRNA sequences, approximately 50 nts are missing at either end of this sequence. Gel electrophoresis measurements have shown that the *Artemia* 28S rRNA is approximately 3750 nts long (Roberts & Vaughn, 1982; Cruces *et al.*, 1982).

When the brine shrimp rRNA I prepared was run over a gel, the amount of 28S rRNA was low compared to the 18S rRNA band (Figure 5.9A), and rRNA from 60S subunits included a strong 18S rRNA band (Figure 5.9A) despite the fact that there should be no 18S rRNA in the large subunit. When the same gel was run longer (Figure 5.9B), the 18S rRNA band separated into multiple bands. These findings are explained by a report in the literature which notes that all insects and crustacea, including brine shrimp, have a “hidden break” in their 28S rRNAs that divides the molecules into two pieces having approximately the same electrophoretic motility as the 18S rRNA (Ishikawa, 1973).

Hidden break

Ishikawa initiated a structural analysis of ribosomal RNAs across the animal kingdom and discovered that 28S rRNA molecules of the superphylum protostomes are structurally distinct from all other organisms (Ishikawa, 1973, 1975a,b). Protostomes are arthropods, mollusks and worms including *D. melanogaster* and *Artemia*, and are distinct from the deuterostomes, which include all vertebrates. Ishikawa found that mature 28S rRNA molecules of the protostomes generally contained a nick called the “hidden break,” while mature 28S rRNA molecules of the deuterostomes do not (Ishikawa, 1973).

The “hidden break” of protostome 28S rRNAs occur about halfway down this molecule and breaks it into a 5' piece called the 28S α fragment and a 3' piece called the 28S β fragment. The hidden break occurs in expansion segment 19 (ES19 using the Gerbi, 1996 notation and is also

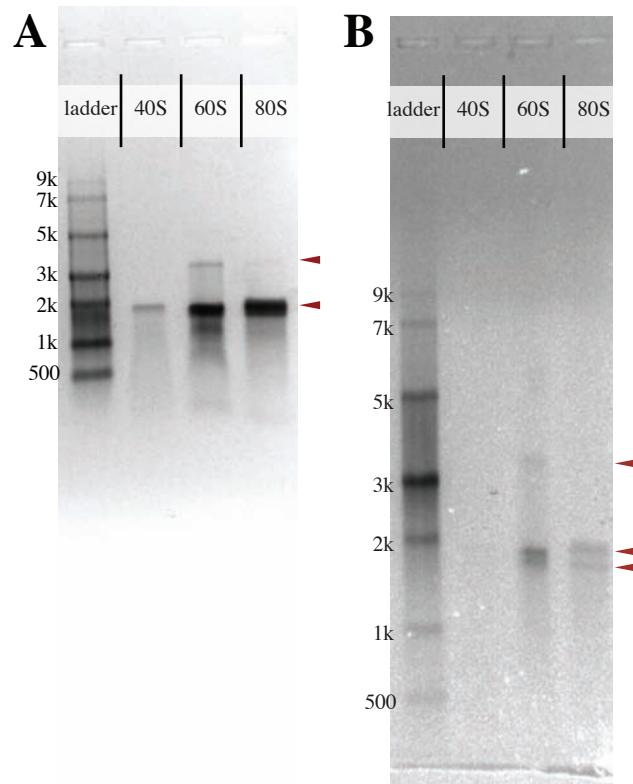


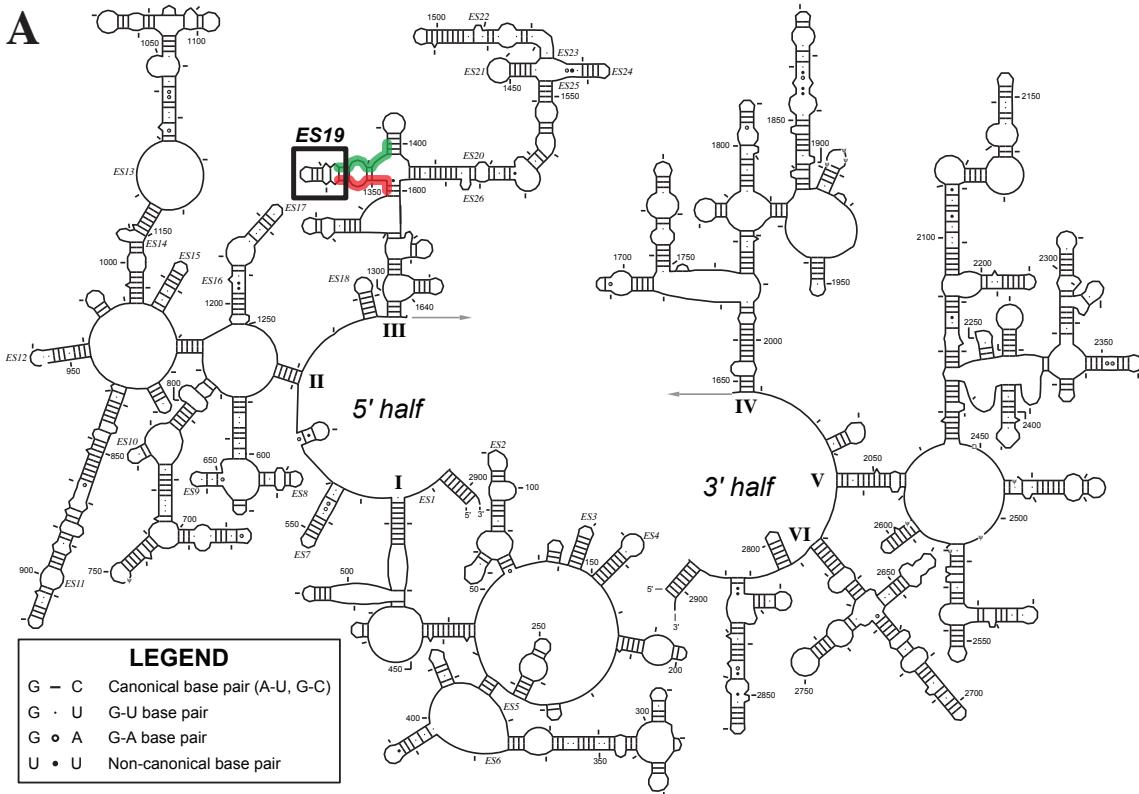
Figure 5.9: Agarose gel of the *Artemia* ribosomal RNA. (A) Run for a short time with lanes for the 40S and 60S subunits, and entire 80S particle. (B) The gel run for a longer time showing the separation of the RNA band. 1% agarose gel with ethidium bromide stain. The image colors are inverted for clarity. Provides similar results to Cruces *et al.* (1982).

called divergent segment, D7a and variable region, V9 by other notations) of domain III of the 23S rRNA (Figure 5.10A). Its tertiary structure location is on the right side of the ribosome when facing its subunit interface near proteins L15 and L7ae (Figure 5.10B). Its specific location within the sequence has been mapped in six different organisms and the sequence of the 28S α fragment for *Artemia* was determined accidentally by Nelles *et al.* (1984b).

In order to locate the entire “hidden break” in *Artemia*, and to determine the predicted size of its 28S rRNA fragments, the *Artemia* sequence was aligned to other known complete 28S rRNA fragments. Figure 5.11 shows a sequence alignment with the gapped regions. The alignment shows the canonical “CGAAAGGG” motif identified by Fujiwara & Ishikawa (1986) on the β strand. It is evident that the gapped area is not conserved between species and it is located in a remote area (Figure 5.10B). Other researchers have noted that ES19 is highly variable throughout all organisms suggesting this region has no functional importance (Mallatt *et al.*, 2004).

According to Nelles *et al.* (1984b), the α piece would be of length 1716 nucleotides (nts) and based on the alignment in Figure 5.11 the β piece would have a length between 1855 and 1911 nts. Figure 5.12 shows the predicted structures of the expansion segment for *Artemia* and six other protostomes. In general, both the α and β cuts are located in the same place in a helical or loop region near the conserved sequences (boxed in Figure 5.11). Using the secondary structure, the gap region for *Artemia* was placed at the most likely spot on the opposite side of the helix as the α cut (Figure 5.12, bottom center) which results in a gap length of 33 nts and 28S β fragment length of \sim 1879 nts.

In conclusion, the *Artemia* sequence contains the same features as the other protostomes, which explains the unexpected results in Figure 5.9. Figure 5.9B shows three faint bands in the 80S: two are approximately close to the 2k nucleotide (nt) marker which correspond to the 28S β fragment (\sim 1879 nts) and 18S rRNA (1810 nts, Weekers *et al.*, 2002; Nelles *et al.*, 1984a) and the third one – smaller than the other two – corresponds to the 28S α fragment (1716 nts).



B

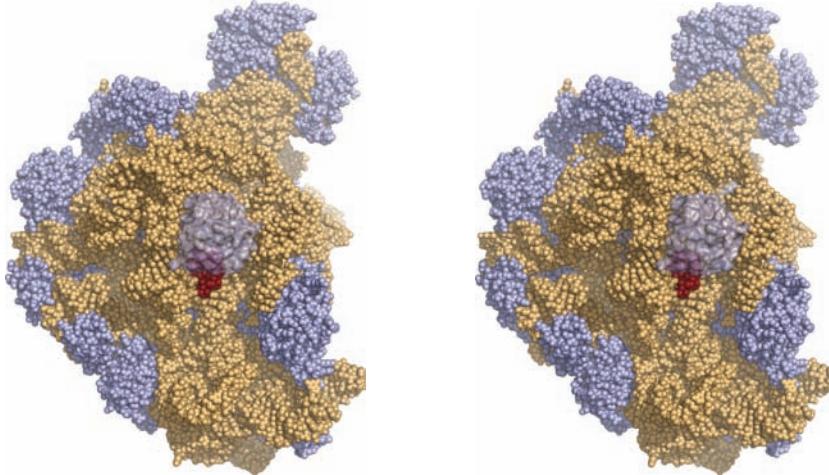


Figure 5.10: Secondary and tertiary location of expansion segment number 19. (A) The secondary structure of the 23S rRNA from the *E. coli* large ribosomal subunit. Every tenth nucleotide is marked with a tick mark and every 50th nucleotide is numbered. All 26 expansion segment locations on the 5' half of the 23S rRNA are marked with *ES* and the number. Expansion segment 19, *ES19* is shown in bold with a black box located at nucleotides 1350–1380 in *E. coli*. The boxed conserved sequences in Figure 5.11 are highlighted in red for the 5' and green for the 3'. (B) Side view of the atomic structure of the 50S large ribosomal subunit from *H. marismortui* (Klein *et al.*, 2001). Proteins are colored in blue and the rRNA is in orange. The location of expansion segment 19, *ES19* is colored in dark red at nucleotides 1460–1480 (*H. marismortui* numbering). Ribosomal proteins L15 and L7ae are transparent to allow for a better view of the rRNA helix.

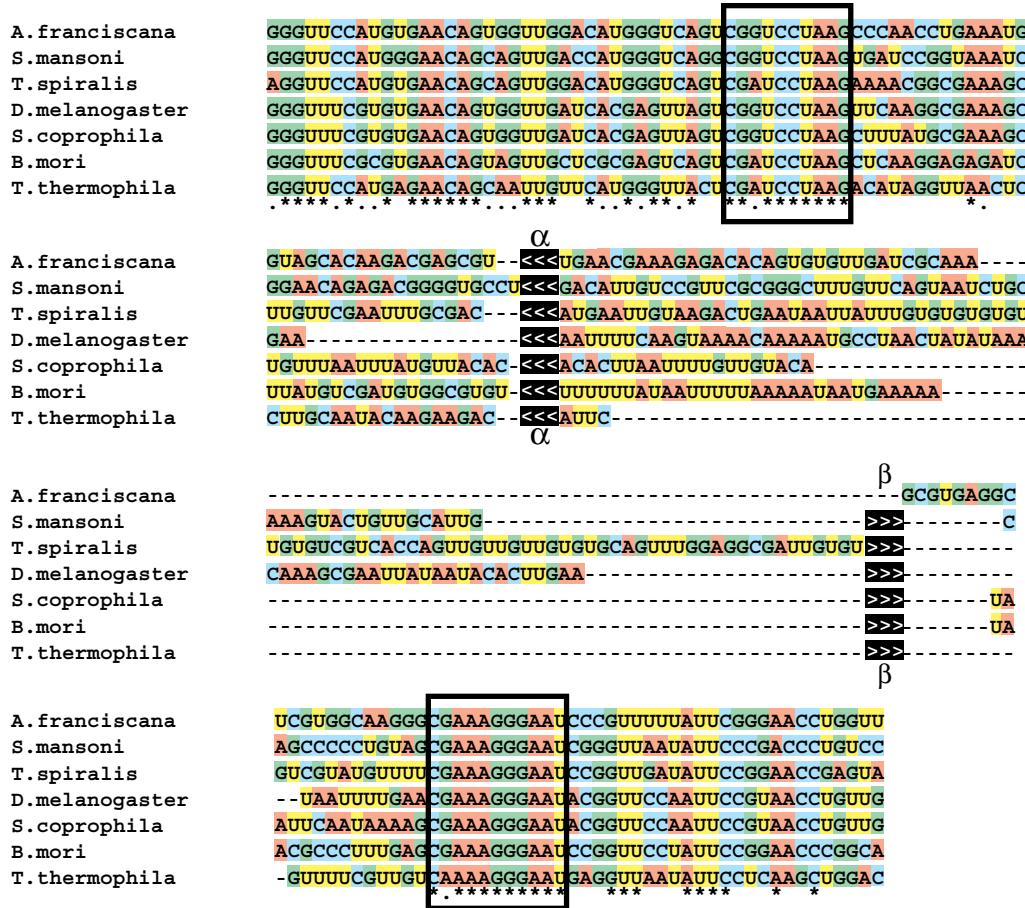


Figure 5.11: 28S rRNA sequences of protostomes with known “hidden break” locations aligned to the *Artemia* 28S rRNA. The conserved closing loop regions, 3' CGRUCCUAAG and 5' CRAAAGGGAAU sequences are boxed. The 5'-28S α fragment cut location is marked in black with “<<<” and the 3'-28S β fragment cut location is marked in black with “>>>”. The “*” means that the nucleotide is conserved in all seven species and “.” means that a pyrimidine or purine is conserved in all seven species. Alignments done with CLUSTALW (Chenna *et al.*, 2003) using the default settings.

Artemia franciscana, brine shrimp (Nelles *et al.*, 1984b), sequence from Mallatt *et al.* (2004), *Schistosoma mansoni*, parasitic worm (van Keulen *et al.*, 1991), *Trichinella spiralis*, roundworm (Zarlenga & Dame, 1992), *Drosophila melanogaster*, fruit fly (De Lanversin & Jacq, 1983), sequence from Tautz *et al.* (1987), *Sciara coprophila*, fungus gnat (Ware *et al.*, 1985), *Bombyx mori*, silkworm (Fujiwara & Ishikawa, 1986) and *Tetrahymena thermophila*, ciliate protozoa (Engberg & Nielsen, 1990).

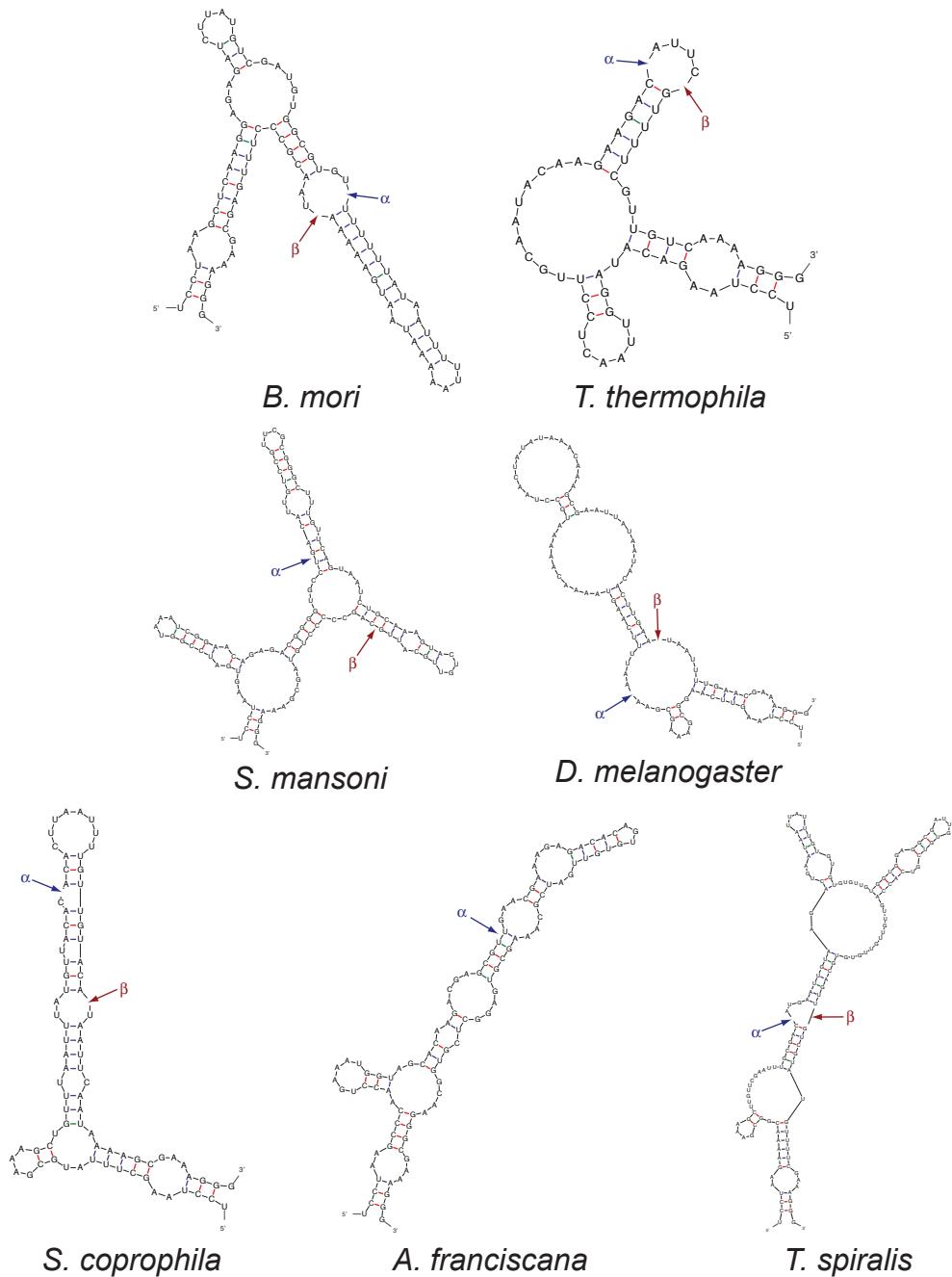


Figure 5.12: Location of the “hidden break” in predicted ES19 secondary structures. Secondary structures were calculated using MFOLD (Zuker, 2003) of regions between the conserved 3' UCCUAAG and 5' CRAAAGGG sequences. The locations of the α and β splice sites are shown with arrows.

Instability of the 40S particles

Throughout the preparations of *Artemia* ribosomes, there was a great difficulty purifying 40S subunits. Some insight was obtained from the following experiment. A sample of 80S particles that had been purified on a sucrose gradient (Figure 5.13A) was divided into two aliquots. The first aliquot was incubated at 37°C for 30 minutes and the second aliquot was left on ice. When the two aliquots were run on sucrose gradients under dissociation conditions, differences in the 40S peak appeared (Figure 5.13B,C). The 40S peak obtained from the sample incubated at 37°C consisted of two peaks a large one that runs the same position as the 40S peak from the sample held at 4°C and a second peak that is smaller in size. In order to understand what is happening to these samples, electron micrographs were obtained.

Negative stain study of particles

Electron micrographs of the *Artemia* ribosomes were taken using tungsten negative stain. Samples had a tendency to clump, and for clarity Figure 5.14 shows only regions on the microscope grid where the sample was relatively unclumped. For the 80S and 60S particles, average particle diameters were calculated, which correspond well with the hydrodynamic diameters determined for them by Nieuwenhuysen & Clauwaert (1981). However, the 40S subunit did not appear to have a uniform shape suggesting that they had undergone some kind of denaturation (Figure 5.14, right). The second 40S peak from sample incubated at 37°C was not imaged because it was not made in large enough quantities. Other than the 40S, the images compare well with the micrographs of *Artemia* ribosomes taken by Boublík & Hellmann (1978) that are shown in Figure 5.14C. Clearly 40S subunits do not survive under the ionic conditions used for their purification.

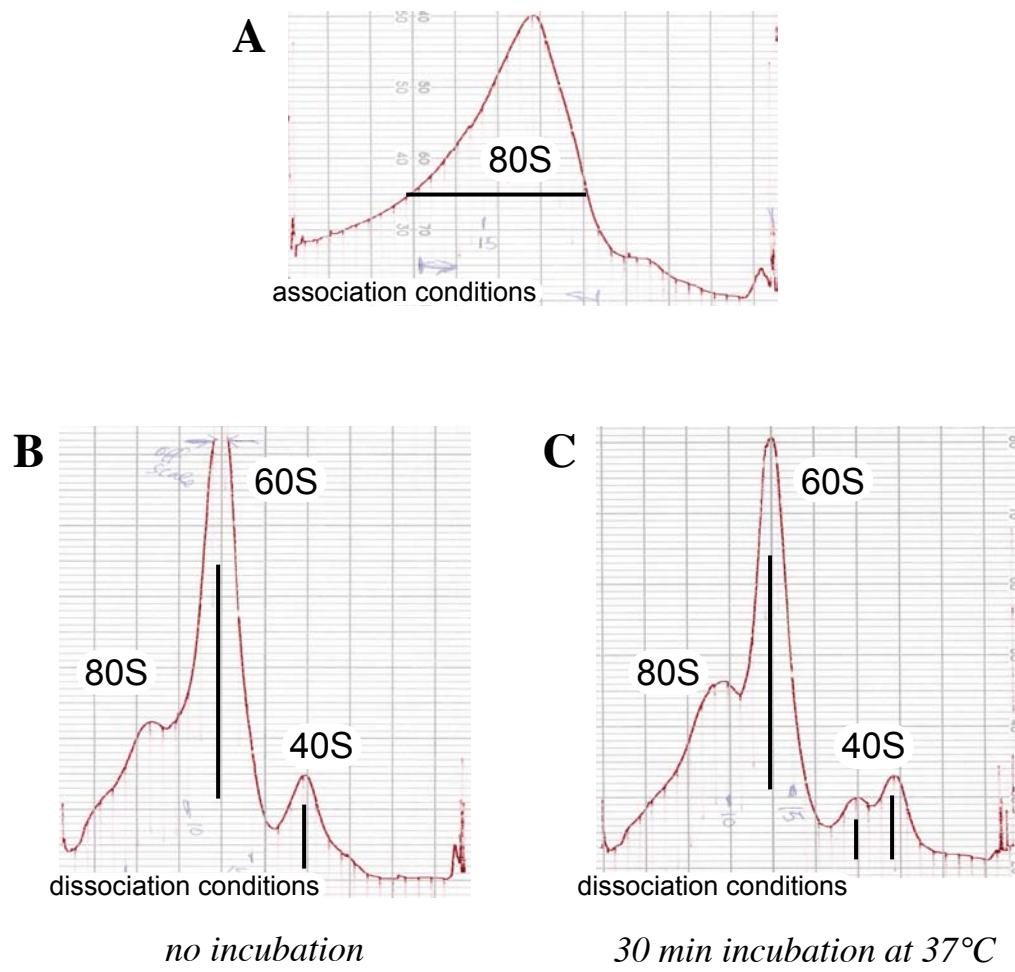


Figure 5.13: Sucrose gradient profiles for incubated and chilled ribosomal preparations. (A) Initial preparation under association conditions. (B) Sucrose gradient profile for normal ribosomal preparation. (C) Sucrose gradient profile for a ribosomal preparation incubated for 30 minutes at 37°C from the same batch.

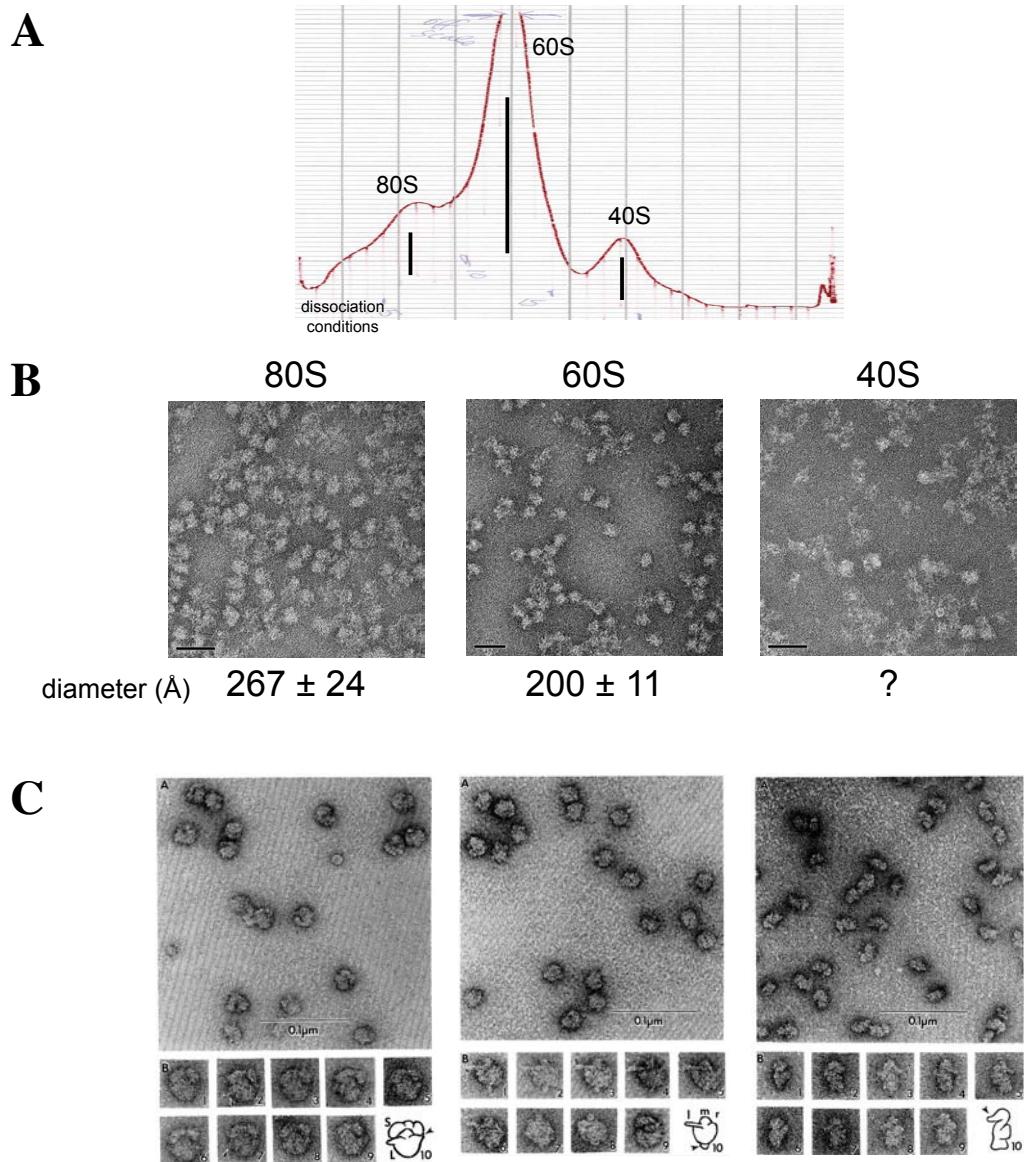


Figure 5.14: Electron micrographs of the *Artemia* ribosomes using negative stain. (A) Sucrose gradient of ribosomes and subunit collected and used in part B and the same as Figure 5.13. (B) Micrographs taken by Vinzenz Unger with the calculated average particle diameter written below. (C) Previously published micrographs of *Artemia* ribosomes Boublík & Hellmann (1978).

5.2.4 Crystallization

Solubility of *Artemia* ribosomes

Before setting up crystal trays it is important to establish the solubility of the sample in the presence of the precipitant. For the *Artemia* ribosomes, it seemed best to use polyethylene glycol (PEG) (molecular weights between 400 and 20,000 Daltons) as precipitants due the successes that have been achieved with PEG for crystallizing prokaryotic ribosomes. Table 5.5 shows the solubility limits for *Artemia* 80S ribosomes for the various PEGs investigated. Simple crystal screens setup using these concentrations of PEG resulted in only amorphous precipitate after a few days.

Low throughput crystallizations: CRYSTOOL

The solubility information in Table 5.5 was entered in the web-based program, CRYSTOOL (Segelke, 2001), a tool that generates efficient crystallization screens by covering the vast parametric space available using probability algorithms. Crystal trays were set up focusing on PEGs in the molecular weight range of 2k–8k, primarily 4k. pH was varied between 5.0 and 8.5 in increments of 0.5 with the pH range 6.5–7.5 weighted to occur more frequently. A variety of salts with 16 different cations and 4 different anions, alcohols, and other additives such as spermine and spermidine, were included in the screen. Over two thousand different conditions for 80S ribosomes and hundreds of conditions for 60S subunits were setup in duplicate at both 12°C and 19°C using these buffers. The rate at which crystal trays were setup was limited solely by the rate at which the buffers required could be prepared. A survey of trays showed that 14% immediately formed precipitate, 64% formed precipitate over a span of 2–5 days, and 22% never formed precipitate. Thus while the majority of drops set up led to definite conclusions, albeit in short time frames, none produced any crystals.

precip.	mol. weight	lower bound	upper bound	screen range
PEG	400	7.7%	11%	
PEG	2k	4.5%	7.5%	
PEG	4k	2.4%	3.9%	1.0 - 2.5%
PEG	6k		8%	
PEG	8k–20k		4%	

Table 5.5: Solubility of ribosomes in the presence of the precipitant, polyethylene glycol. The lower bound of the solubility is defined as the maximum concentration of precipitant, polyethylene glycol (PEG) listed in the first column in the sample where the sample appears completely clear. The upper bound of the solubility is defined as the minimum concentration of precipitant in the sample where precipitation is visible.

High throughput crystallizations: HWI

The Hauptman-Woodward Institute provides a high-throughput crystallization screening service. The Hauptman-Woodward Institute (HWI) located in Buffalo, NY receives samples in the mail and prepares crystal-growth screening experiments in 1536-well microassay plates using the microbatch technique. Further, this process is free to those who submit materials as long as the results of the crystallization trials can be added to the HWI's database.

Because of the relative ease of sending the sample away compared to setting up 1500 conditions in the lab, both 80S ribosomes and 60S large subunits samples were submitted to HWI. One caveat of this is that of the 1536 crystallization experiments done at HWI, over 221 involved phosphate buffers, which readily form magnesium phosphate crystals. Since ribosomes require magnesium ions to remain folded, ribosome crystallization trials run with phosphate buffers are essentially worthless. In addition to experiments with phosphate buffers, 66 of the buffers used contained cyanate, 19 had tartrate in them, and 2 had succinate buffers. All of these anions can produce magnesium salt crystals under certain conditions. In addition, 143 conditions had a pH of 4 or below, which is too acidic and 226 had a pH of 9 or higher, which is too basic. Basic buffers are a large problem because the phosphate backbone of RNA tends to hydrolyze at high pH. Salt crystals are usually easy to discriminate from macromolecular crystals because they tend to form very quickly (~ 1 day) and are large and flawless. Macromolecular crystals typically take much longer to form and are often small and ugly unless heavily optimized. Thus only 958 of the 1536 conditions tested at HWI were worth investigating. Another caveat of the HWI conditions is that they are optimized for small proteins and therefore usually involve higher concentrations of precipitant than is optimal for ribosomes causing them to precipitate almost immediately.

Of these 958 conditions, 8 had some crystalline material for 80S ribosomes and 6 had crystalline material for the 60S subunits (Table 5.6, Figures 5.15 and 5.16). Two conditions (850, 1413) have crystals for both 80S and 60S samples, suggesting the ribosome buffer was responsible for the

crystals seen, and not the ribosomes. Of the 8 conditions for 80S ribosomes that had crystalline material, only 4 conditions (570, 571, 1413, and 1425) produced crystals of any appreciable size (Figure 5.15). Similarly, the 6 conditions for 60S subunits had only 3 conditions (1247, 1413, and 1515) with appreciable crystals (Figure 5.16). Nonetheless, all conditions in Table 5.6 were repeated using both microbatch and vapor diffusion techniques with no success. Unfortunately, only pictures are available of the crystals that formed.

High throughput crystallizations: pre-made buffer kits

Three popular pre-made buffers kits (Crystal Screen I, Crystal Screen II, and Natrix) from Hampton Research were tested for both *Artemia* 80S ribosomes and 60S subunits. The kits are geared toward protein crystallography and some of the conditions provided by these kits include phosphate buffers that as pointed out earlier are incompatible with RNA.

Using the Hydra II microdispenser robot from Matrix in the Yorgo Modis lab, two pre-made 96-well buffer kits were setup. Crystal trays were setup with the PEG buffers from “The PEG Suite” from Nextal Biotechnologies (now QIAGEN) and the “Heavy & Light Pack” from Molecular Dimensions. Similar to several HWI conditions the precipitants in the pre-made PEG buffers were too high to conduct proper crystallization trials, so the pre-made buffer concentrations were diluted to one-third their initial concentrations. The “Heavy & Light Pack” did not produce any hits, but six conditions from “The PEG Suite” did produce crystals initially (Table 5.7, Figure 5.17). Four of these crystal conditions contained phosphate buffers and hence produced large, beautiful crystals. The remaining two conditions that produced crystals had no phosphate, but tartrate instead. One of these crystals was harvested and shot on the X-ray home source. The intense spots seen and the large distances between spots, indicated that the crystal was salt, presumably magnesium tartrate. After checking the trays 16 months later, five more non-obvious conditions contained crystals, with no discernible trend (Table 5.7, top). The crystals are birefringent, but small. However, because they

Complete 80S ribosome

Crystal description									
#	Additive	conc	Buffer	conc	pH	Precipitant	conc	time	size (count)
64	Li·Cl	4.42 M	Tris	0.1 M	8.0			3 weeks	tiny (10+)
85	Mg·SO ₄	0.53 M	HEPES	0.1 M	7.0			3 weeks	micro
570	Mg·Cl ₂	0.10 M	HEPES	0.1 M	7.0	PEG 4000	20%	2 weeks	medium (3)
571	Mg·SO ₄	0.10 M	MOPS	0.1 M	7.0	PEG 4000	20%	4 weeks	small (1)
850	Amm. sulfate	0.10 M	HEPES	0.1 M	7.0	PEG 400	40%	1 day	needles
871	K ₂ CO ₃	0.10 M	Na-acetate	0.1 M	5.0	PEG 400	40%	2 weeks	micro
1413	Amm. sulfate	0.20 M	Tris	0.1 M	8.5	PEG 3350	25%	3 weeks	small (1)
1425	Amm. acetate	0.20 M	Tris	0.1 M	8.5	PEG 3350	25%	3 weeks	small (1)

Large 60S subunit

Crystal description									
#	Additive	conc	Buffer	conc	pH	Precipitant	conc	time	size (count)
849	Amm. sulfate	0.10 M	MOPS	0.1 M	7.0	PEG 400	40%	1 day	needles
850	Amm. sulfate	0.10 M	HEPES	0.1 M	7.0	PEG 400	40%	1 week	needles
1247	Na·Cl	4.00 M	Tris	0.1 M	8.0			3 weeks	medium (2)
1320	Cs·Cl	0.05 M	MES	0.1 M	6.5	<i>Jeff.</i> M-600	30%	3 weeks	micro
1413	Amm. sulfate	0.20 M	Tris	0.1 M	8.5	PEG 3350	25%	4 weeks	medium (1)
1515	Mg·SO ₄	1.00 M	Tris	0.1 M	8.5			1 weeks	medium (1)

Table 5.6: Crystallization results from the Hauptman-Woodward Institute (HWI). The Hauptman-Woodward Institute (HWI) in Buffalo, NY conditions that resulted in crystal or crystal-like forms. “conc” stands for concentration, “Amm.” stands for ammonium, and “Jeff.” stands for Jeffamine®, a polyoxyalkyleneamine with similar properties to PEG.

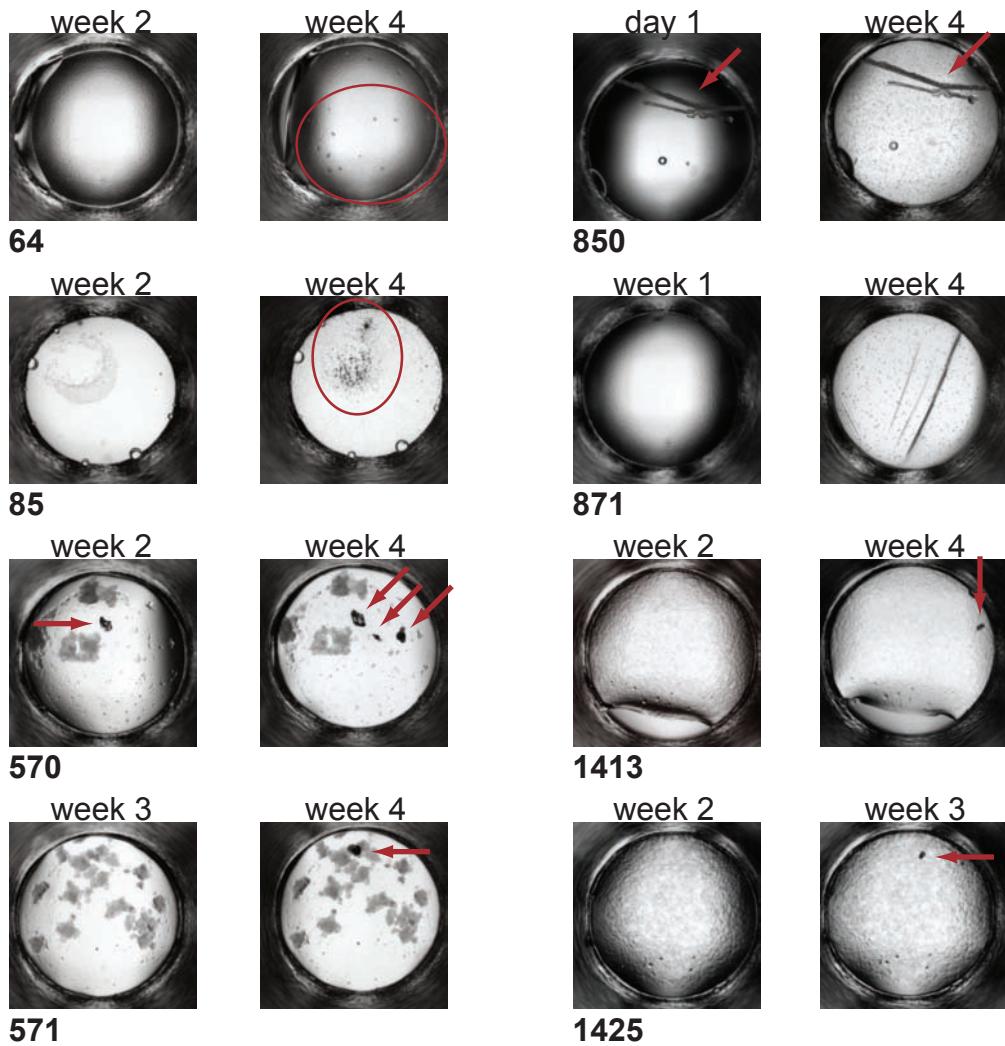


Figure 5.15: Images of crystallization results from HWI for 80S ribosomes. For each pair of images the conditions is listed below on the left and for each image the time at which the picture was taken is listed above. Arrows and circles indicate the locations of any crystals.

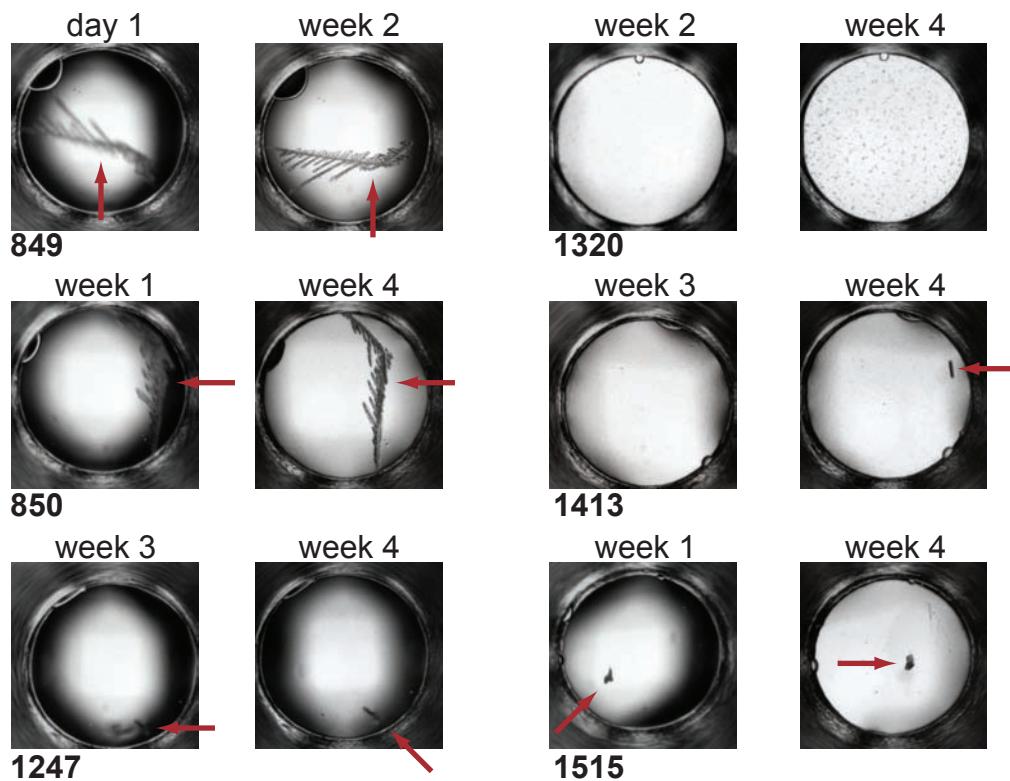


Figure 5.16: Images of crystallization results from HWI for 60S ribosomal subunits. For each pair of images the conditions is listed below on the left and for each image the time at which the picture was taken is listed above. Arrows and circles indicate the locations of any crystals.

##	additive	observed	drop	crystal description
54	Sodium chloride	Oct 2006	both	small (1), small (1)
55	Calcium chloride	Oct 2006	2:1	small (1)
56	Potassium chloride	Oct 2006	1:1	small (1)
57	Ammonium chloride	Oct 2006	both	small (1), small (1)
77	Potassium acetate	Oct 2006	both	medium (1), medium (1)
85	K/Na tartrate	July 2005	both	medium (1), medium (2)
86	di-Ammon. tartrate	July 2005	both	medium† (2), small (6)
88	di-Sodium phosphate	July 2005	both	needles, large (1)
90	di-Potassium phosphate	July 2005	both	large (1), large (1)
91	Ammonium phosphate	July 2005	both	large (1), large (1)
92	di-Ammonium phosphate	July 2005	both	medium (6+), medium (5)

Table 5.7: Crystallization results from 80S ribosome sample and “the PEGs” buffers. “The PEGs” conditions that resulted in crystal or crystal-like forms. 80S ribosome conditions were setup in June 2005 with two drops types: 2 parts sample and 1 part buffer (2:1) and 1 part sample to 1 part buffer (1:1). Crystals appeared in the drop designated in “drop” column. All conditions contain 20% PEG 3350 and 0.2 M of the additive listed in the second column. †denotes that the crystal was harvested and shot on the X-ray home source.

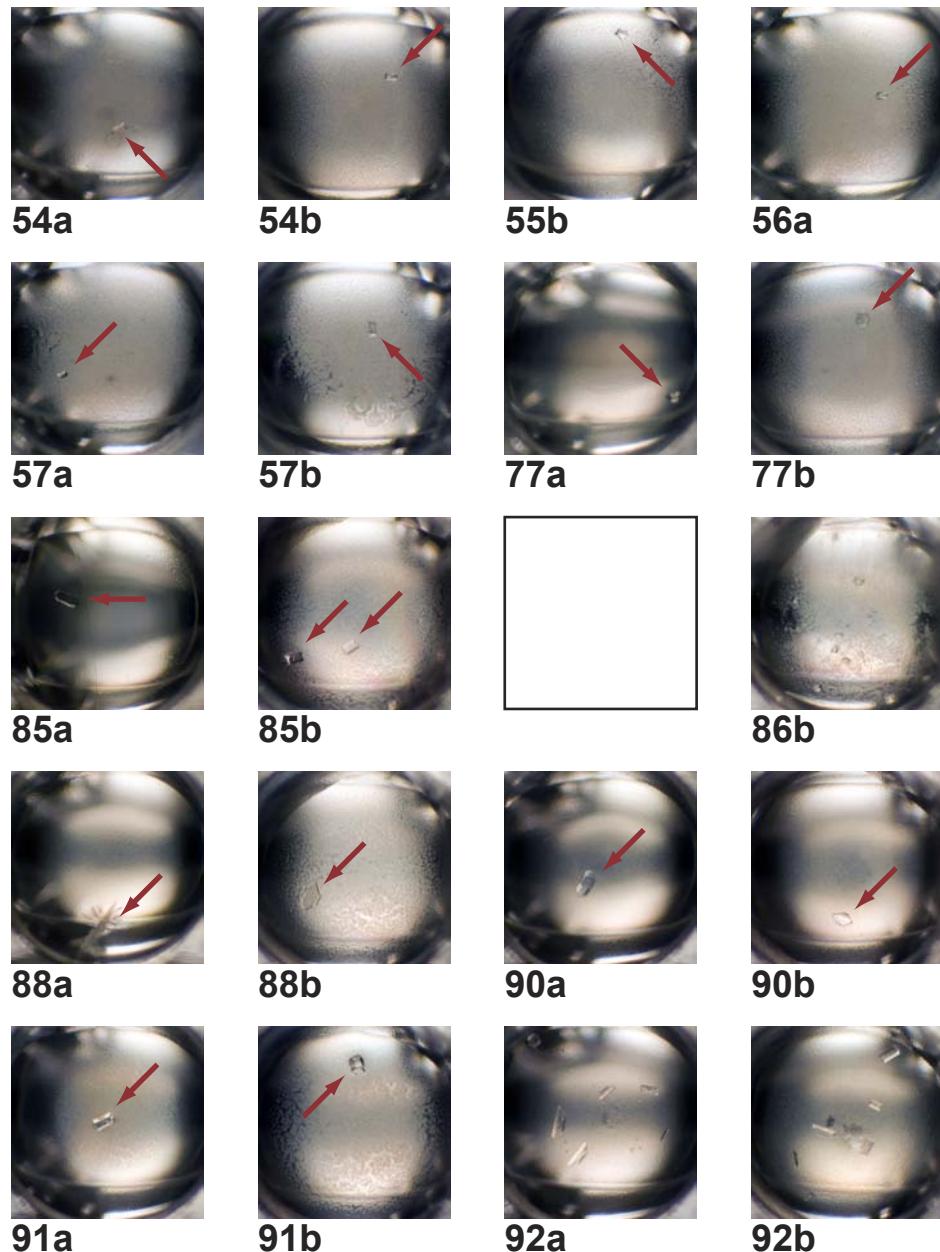


Figure 5.17: Images of crystallization results of “the PEGs” buffers for 80S ribosomes. “The PEGs” conditions that resulted in crystal or crystal-like forms. The conditions for each number is given in Table 5.7. 80S ribosome conditions were setup in June 2005 with two drops types: (a) 0.5 μL sample 0.5 μL buffer and (b) 0.6 μL sample to 0.3 μL buffer. This missing image, 86a, had its crystals harvested and no picture is available. Crystal are indicated with arrows when there are less than 3 crystals present.

appeared so late, numerous external factors could cause their formation and they were not explored further.

Several members from the Thomas Steitz lab have created pre-made buffer “libraries” for use with the crystallization robot for high throughput crystallization trials. This seems to be the best method for crystallization experiments, because thousands of conditions can be setup in only a few hours. When this system is coupled with a machine to take pictures of the crystal trays, an extremely efficient system will be in place. For the 1536 HWI conditions, I was able to evaluate about 2-3 conditions per second and this could be applied locally.

Considerations for future crystallization experiments with *Artemia* ribosomes

Beyond the choice of materials used in crystallization conditions, the most crucial steps in ribosome crystallization is to make sure the preparation is homogenous. To obtain crystals of *E. coli* 70S ribosomes that diffract well, ribosomal protein S1 was removed from preparations and subunits were purified separately before being re-associated (Schuwirth *et al.*, 2005). For *H. marismortui*, samples were back-extracted (precipitated and re-solubilized) before being crystallized (Ban *et al.*, 2000). Yonath & Wittmann (1988) stressed the importance of having functional ribosomes in the crystal trials. It has also been found that empty ribosomes are not as stable conformationally as do not average as well as those with mRNA, tRNA, and protein cofactors bound (Joachim Frank, personal communication).

Before *Artemia* can become a viable source of ribosomes for crystallization several issues need to be addressed. Sample aggregation and 40S subunit instability observed by electron microscopy must be prevented. The high molecular weight proteins observed in protein gel electrophoresis should be characterized to find out if it would be wise to remove them prior to crystallization. Varying buffers and temperature during purification seems to be the best way to proceed. Aggregation and stability can be measured using dynamic light scattering (DLS) or electron microscopy (EM) as an assay. Quick and dirty EM studies could also be used to test

the homogeneity of ribosome samples by producing an EM structure as in Figure 5.14. Since the general structure of the ribosome is known, efforts to create an averaged EM structure should only take on the order of few weeks for a novice and a few days for an experienced user saving valuable time setting up crystal trays.

5.3 METHODS

5.3.1 Materials

Rabbit pancreas

Rabbit Pancreas was obtained from Pel-Freez Biologicals (<http://pelfreez-bio.com>) product number 41231-2: “Young Rabbit Pancreas.” Pancreas shipped on dry ice and upon receiving pancreas were weighed and stored in cryo-buffer (below) at -80°C. Pancreas stored at -80°C remain pink, while those stored at -20°C turn brown and are subject to freezer burn.

Rabbit reticulocytes

Rabbit reticulocytes were also obtained from Pel-Freez Biologicals (<http://pelfreez-bio.com>) product number 31102-1: “Rabbit Reticulocyte Rich Whole Blood.” The rabbit blood was shipped on ice on the third Monday of every month. No suitable cryo-conditions were attempted, but should be possible if stored in proper buffer. Upon receiving blood, retic cells were immediately pelleted from the blood and used in ribosomal purification.

Brine shrimp

Brine shrimp (*Artemia franciscana*) cysts were initially obtained from Salt Lake Shoreline Aquaculture, Logan, UT (<http://saltlakeshoreline.com>) in large amounts. After their website went down, cysts were obtained from San Francisco Bay Brand, Newark, CA (<http://sfbb.com>) after the Salt Lake website went down. Upon receiving both types can be stored at 4°C indefinitely.

5.3.2 Buffers

Rabbit pancreas

- Lysis Buffer (pH 7.5) — 20 mM HEPES-KOH, 250 mM sucrose, 50 mM K·Ac, 3 mM Mg·Ac₂, 1 mM EDTA, 1 mM DTT, 1 mM Na·N₃, 50 mM TEA, 3 mM Mg·Cl₂,
- Cryo-Storage Buffer (pH 7.5) — 5 mM HEPES-KOH, 1 M sucrose, 50 mM K·Ac, 3 mM Mg·Ac₂, 1 mM EDTA, 1 mM DTT, 1 mM Na·N₃, 3 mM Mg·Cl₂,
- Purification Buffer (pH 7.5) — 20 mM HEPES-KOH, 50 mM K·Ac (pH 7.5), 3 mM Mg·Ac₂, 1 mM EDTA (pH 7.5), 1 mM DTT, 3 mM Mg·Cl₂,

Rabbit reticulocytes

- Saline Buffer — 10 mM NaH₂PO₄, 30 mM K₂HPO₄, 125 mM NaCl, pH 7.4
- Purification Buffer (pH 7.5) — 20 mM HEPES-KOH, 50 mM K·Ac (pH 7.5), 3 mM Mg·Ac₂, 1 mM EDTA (pH 7.5), 1 mM DTT, 3 mM Mg·Cl₂

Brine shrimp

- Association Buffer (Assoc): 35 mM Tris-H.Cl (or HEPES-K.OH) pH 7.4, 70 mM K·Cl, 9 mM Mg·Cl; *optional:* 1 mM EDTA, 1mM DTT, 0.1 mg/mL Heparin
- Gradient Dissociation Buffer (Dis400): 20 mM HEPES-K·OH pH 7.5, 400 mM K·Cl, 11 mM Mg·Cl
- Dissociation Buffer (Dis250): 20 mM HEPES-K·OH pH 7.5, 250 mM K·Cl, 11 mM Mg·Cl

as storage buffer to reduce aggregation. Optionally buffers were supplemented with

1mM DTT (or β ME), 1 mM EDTA, 0.1 mg/mL Heparin

as protease and ribonuclease inhibitors.

5.3.3 Cell Breakage

For each system a different method is used to break the cells, but after the cells are broken all methods are essentially the same except for the buffers used.

Rabbit pancreas

1. Thaw pancreas in chilled water for about 20 min. Pour off cryo-storage buffer and if possible, cut any fat tissue off pancreas.
2. Mix 15 *grams* of pancreas with 60 mL of Lysis Buffer. Blend in Waring blender until fine with three 20-second bursts, shaking in between bursts.
3. Centrifuge the blended suspension at ~3000g (5.2k rpm) for 2 min (IEC Centra CL3R with 801 fixed angle rotor).
4. Scoop off the fatty layer floating on top of the sample. Then aspirate off any remaining fatty floating material at top. Place the supernatant and the resuspended pellet in 55 mL Dounce homogenizer.
5. Homogenize: 5 strokes (10 sec down, 10 sec up) in the cold room at a reasonably fast speed. Monitor the temperature of the sample; it may need to be chilled in between strokes. Leave homogenizer in “down” position on last stroke and collect only liquid sample and discard the large chunks of material below the piston. I would recommend using the T-25 motorized homogenizer from IKA for any future preps.
6. Pour liquid through glass wool to filter out any remaining insoluble material
7. Add 5% CHAPS detergent (pH 7.5) to the sample about one-fourth the total volume and place on ice for about 15 minutes. Digitonin is commonly used by others (Walter & Blobel, 1983), but was too expensive for my purposes and I found CHAPS works well. Do not add detergent before this step otherwise bubbles will form.
8. Centrifuge the blended suspension at ~3000g (5.2k rpm) for 60 min (IEC Centra CL3R with 801 fixed angle rotor) and aspirate off any remaining fatty floating material at top.

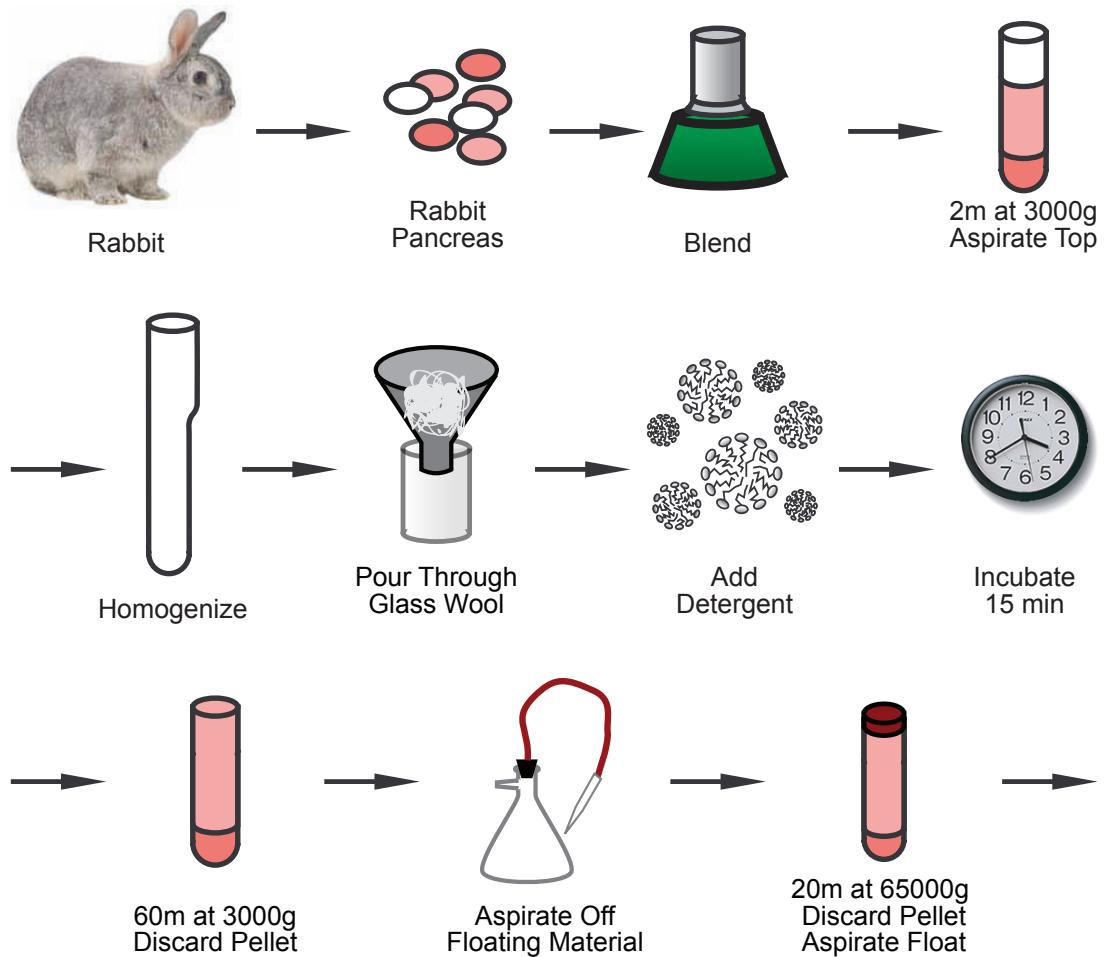


Figure 5.18: Lysis protocol of rabbit pancreas.

9. Centrifuge the blended suspension at 65,000g (30k rpm) for 20 min (Beckman Ti70 rotor) and again aspirate off any remaining fatty floating material at top.
10. Measure the absorption, OD₂₆₀, and proceed to pellet the ribosomes.

Rabbit reticulocytes

1. Anemic rabbit blood was filtered through a cheesecloth and then centrifuged at ~2,000g for 10 min (Sorvall SA-600 rotor). The bloody supernatant was poured off and reticulocyte pellet was resuspended in Lysis buffer. The centrifugation and resuspension was repeated three times. During the last centrifugation, the saline buffer is removed and the reticulocytes are resuspended in 1.5 volumes of distilled water and mixed thoroughly.
2. The suspension was then transferred to the 55 mL Dounce homogenizer and all remaining steps were performed in the same way as the rabbit pancreas.

Brine shrimp

1. In a large beaker, mix 50 grams of *Artemia* cysts with 500 mL of 4°C distilled water and 210 mL of bleach (purchased from Sigma, but Clorox works just as well).
2. Stir mixture every 5 minutes and allow cysts to soak in the bleach for about 20 minutes until white foam starts to build at the surface. Very slowly pour off the bleach mixture while retaining the cysts at the bottom. The color of the cysts should have gone from a dark brown to light brown as in Figure 5.5B,C.
3. After pouring off as much liquid as possible, fill the beaker to the top with fresh, cold, distilled water and allow cysts to settle. Once cysts are settled, pour off water again. Repeat this step at least 5 times to remove all traces of bleach.
4. Before draining the last step, pour cysts and water over a Whatman 1 filter. Then wash with more distilled water.
5. Scrape cysts from filter. These are the decapsulated cysts.

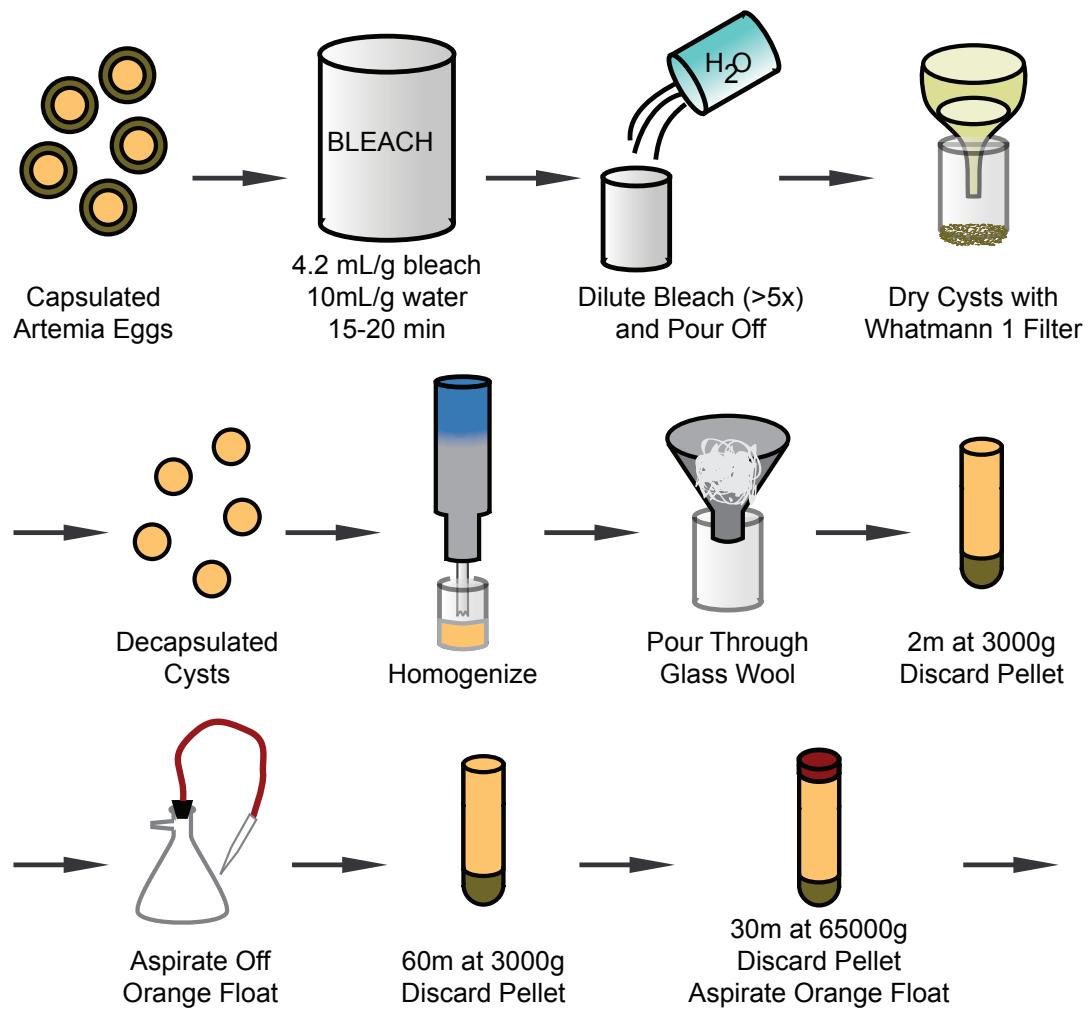


Figure 5.19: Lysis protocol of *Artemia* cysts.

6. Homogenize the cysts using minimal buffer in ice using the T-25 basic Ultra-Turrax motorized homogenizer (product number, 29530.00) from IKA Works, Inc. (<http://ika.net>) with S25N-18G, flat tooth dispersing tool (product number, 05934.00). Homogenize for 10 minutes at max speed (24,000 rpm).
7. Pour solution through glass wool to filter out insoluble material. If large amount of cysts are present in the solution, the solution was centrifuged and the pellet was re-homogenized in fresh buffer for better yield.
8. Centrifuge the homogenized solution at ~3000g (5.2k rpm) for 2 min (IEC Centra CL3R with 801 fixed angle rotor).
9. Aspirate off the orange lipid layer floating on top of the sample, discard pellet and pour supernatant through glass wool again to filter out any remaining insoluble material
10. Centrifuge the blended suspension at ~3000g (5.2k rpm) for 60 min (IEC Centra CL3R with 801 fixed angle rotor) and aspirate off any remaining orange floating material at top.
11. Centrifuge the blended suspension at 65,000g (30k rpm) for 30 min (Beckman Ti70 rotor) and again aspirate off any remaining orange floating material at top.
12. Measure the absorption, OD₂₆₀, and proceed to pellet the ribosomes.

5.3.4 Ribosomes from Cell Lysis

1. Place sample into ultracentrifuge tubes.
2. Place 15 mL of sample into ultracentrifuge tubes. Next, take a syringe, place tubing on tip and slowly add 5 mL 30% sucrose cushion on the bottom of the tube.
3. Pellet the ribosomes by spinning at 184,000g (50k rpm) for 10 hrs (Beckman Ti70 rotor) through the 30% sucrose cushion.
4. Ribosomal pellets were washed with Association buffer and resuspended overnight in the cold room using Association or Dissociation buffer.

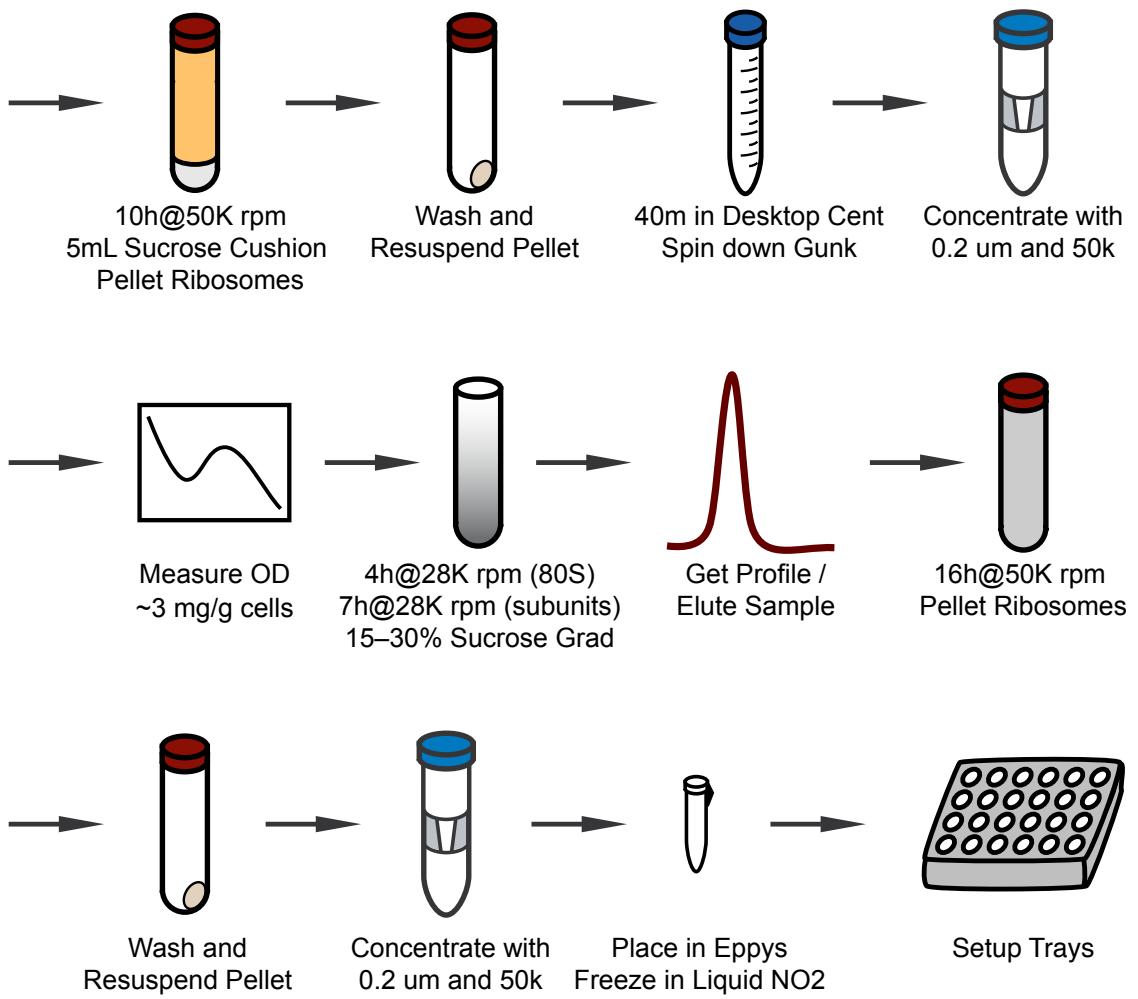


Figure 5.20: Protocol from cell lysis to purified ribosomes.

5. Sample was loaded over 10–30% sucrose gradients and spun at 104,000g (28k rpm) in Beckman SW28 rotor for 4 hrs for 80S ribosomes and 6.5 hrs for subunits.
6. Ribosomal fractions were collected and concentrated to ~10 mg/mL. Depending on the prep, samples either retained the sucrose from the gradients or were dialyzed against fresh buffer. 40S subunits tended to be more stable in the presence of sucrose.

5.3.5 Assessing Sample Quality

RNA agarose gel electrophoresis

Phenol extracted rRNA was analyzed by 1% (w/v) agarose gel electrophoresis using American Bioanalytical Agarose GPG/LE, TBE buffer and 10 μ g ethidium bromide was added to 30 mL solution.

SDS polyacrylamide gel electrophoresis

For ribosomal proteins, Bio-Rad Criterion Tris-HCl 10.5–14% gradient gels were obtained. Ribosomal samples were incubated in the Bio-Rad SDS loading buffer for 10 minutes in boiling water. Samples were loaded and ran for about 2 hours. Gel was removed and stained with Coomassie blue stain overnight and the destained in methanol until the bands were easily distinguished.

5.3.6 Hidden Breaks

Secondary structures for the expansion segments shown in Figure 5.12 were calculated using the MFOLD prediction program (Zuker, 2003).

5.3.7 Crystallization

Crystallization kits

The “3D Heavy & Light Twin Pack HT96” kit (Cat Number, MD1-35) was purchased from Molecular Dimensions Limited (<http://moleculardimensions.com>) because it contained lower precipitant concentrations than other kits. “The PEGs Suite” (Cat Number, 130904) was purchased

from Nextal Biotechnologies (<http://nextalbiotech.com>) recently purchase by QIAGEN. Both kits came in 96-deep well macroblocks for use with the Hydra II 96/384-Channel Microdispenser from Matrix Technologies Corporation (<http://www.matrixtechcorp.com>) in the Yorgo Modis lab. Hampton Research crystallization screens: Crystal Screen (HR2-110), Crystal Screen II (HR2-112) and Natrix (HR2-116) with 50 conditions each were already available in our lab.

5.4 REFERENCES

- C. B. Andersen, T. Becker, M. Blau, M. Anand, M. Halic, B. Balar, T. Mielke, T. Boesen, J. S. Pedersen, C. M. Spahn, T. G. Kinzy, G. R. Andersen, and R. Beckmann (2006). “Structure of eEF3 and the mechanism of transfer RNA release from the E-site”. *Nature*, **443** (7112): pp. 663–668. <http://dx.doi.org/10.1038/nature05126>.
- R. J. Arnold and J. P. Reilly (1999). “Observation of *Escherichia coli* ribosomal proteins and their posttranslational modifications by mass spectrometry”. *Anal Biochem*, **269** (1): pp. 105–112.
- J. C. Bagshaw, F. J. Finamore, and G. D. Novelli (1970). “Changes in transfer RNA in developing brine shrimp”. *Dev Biol*, **23** (1): pp. 23–35.
- N. Ban, B. Freeborn, P. Nissen, P. Penczek, R. A. Grassucci, R. Sweet, J. Frank, P. B. Moore, and T. A. Steitz (1998). “A 9 Å resolution X-ray crystallographic map of the large ribosomal subunit”. *Cell*, **93** (7): pp. 1105–15.
- N. Ban, P. Nissen, J. Hansen, P. B. Moore, and T. A. Steitz (2000). “The complete atomic structure of the large ribosomal subunit at 2.4 Å resolution”. *Science*, **289** (5481): pp. 905–20.
- M. Barbieri, P. Pettazzoni, F. Bersani, and N. M. Maraldi (1970). “Isolation of ribosome microcrystals”. *J Mol Biol*, **54** (1): pp. 121–124.
- R. Beckmann, D. Bubeck, R. Grassucci, P. Penczek, A. Verschoor, G. Blobel, and J. Frank (1997). “Alignment of conduits for the nascent polypeptide chain in the ribosome-Sec61 complex”. *Science*, **278** (5346): pp. 2123–2126.
- J. A. Beeley, E. Cohen, and P. J. Keller (1968). “Canine pancreatic ribosomes. I. Preparation and some properties”. *J Biol Chem*, **243** (6): pp. 1262–1270.
- M. Blau, S. Mullapudi, T. Becker, J. Dudek, R. Zimmermann, P. A. Penczek, and R. Beckmann (2005). “ERj1p uses a universal ribosomal adaptor site to coordinate the 80S ribosome at the membrane”. *Nat Struct Mol Biol*, **12** (11): pp. 1015–1016.
- D. Boehringer, R. Thermann, A. Ostareck-Lederer, J. D. Lewis, and H. Stark (2005). “Structure of the hepatitis C Virus IRES bound to the human 80S ribosome: Remodeling of the HCV IRES”. *Structure (Camb)*, **13** (11): pp. 1695–1706.
- B. Bottcher, S. A. Wynne, and R. A. Crowther (1997). “Determination of the fold of the core protein of hepatitis B virus by electron cryomicroscopy”. *Nature*, **386** (6620): pp. 88–91.
- M. Boulik and W. Hellmann (1978). “Comparison of *Artemia salina* and *Escherichia coli* Ribosome Structure by Electron Microscopy”. *PNAS*.
- G. D. Burkholder, D. E. Comings, and T. A. Okada (1971). “A storage form of ribosomes in mouse oocytes”. *Exp Cell Res*, **69** (2): pp. 361–371.
- B. Byers (1966). “Ribosome crystallization induced in chick embryo tissues by hypothermia”. *J Cell Biol*, **30** (3): pp. 1–6.

- B. Byers (1967). “Structure and formation of ribosome crystals in hypothermic chick embryo cells”. *J Mol Biol*, **26** (2): pp. 155–167.
- Q. Chen, K. Osteryoung, and E. Vierling (1994). “A 21-kDa chloroplast heat shock protein assembles into high molecular weight complexes *in vivo* and in Organelle”. *J Biol Chem*, **269** (18): pp. 13216–13223.
- T. Chen, R. Amons, J. S. Clegg, A. H. Warner, and T. H. MacRae (2003). “Molecular characterization of artemin and ferritin from *Artemia franciscana*”. *Eur J Biochem*, **270** (1): pp. 137–145. <http://dx.doi.org/10.1046/j.1432-1033.2003.03373.x>.
- R. Chenna, H. Sugawara, T. Koike, R. Lopez, T. J. Gibson, D. G. Higgins, and J. D. Thompson (2003). “Multiple sequence alignment with the Clustal series of programs”. *Nucleic Acids Res*, **31** (13): pp. 3497–3500.
- W. C. Choi and W. Nagl (1977). “Ribosome crystals in the oocyte of *Gerris najas* (Heteroptera)”. *Z Naturforsch [C]*, **32** (1-2): pp. 136b–7.
- C. G. Clark, B. W. Tague, V. C. Ware, and S. A. Gerbi (1984). “*Xenopus laevis* 28S ribosomal RNA: A secondary structure model and its evolutionary and functional implications”. *Nucleic Acids Res*, **12** (15): pp. 6197–6220.
- J. S. Clegg, S. A. Jackson, and A. H. Warner (1994). “Extensive intracellular translocations of a major protein accompany anoxia in embryos of *Artemia franciscana*”. *Exp Cell Res*, **212** (1): pp. 77–83.
- C. H. Collins and J. S. Clegg (2004). “A small heat-shock protein, p26, from the crustacean *Artemia* protects mammalian cells (Cos-1) against oxidative damage”. *Cell Biol Int*, **28** (6): pp. 449–455.
- J. A. Crack, M. Mansour, Y. Sun, and T. H. MacRae (2002). “Functional analysis of a small heat shock/α-crystallin protein from *Artemia franciscana*. Oligomerization and thermotolerance”. *Eur J Biochem*, **269** (3): pp. 933–942.
- J. Cruces, J. Sebastian, and J. Renart (1982). “Ribosomal ribonucleic acids from the crustacean *Artemia*”. *FEBS Letters*, **137** (1): pp. 108–110. [http://dx.doi.org/10.1016/0014-5793\(82\)80325-6](http://dx.doi.org/10.1016/0014-5793(82)80325-6).
- J. De Graaf, R. Amons, and W. Möller (1990). “The primary structure of artemin from *Artemia* cysts”. *Eur J Biochem*, **193** (3): pp. 737–750. URL <http://search.epnet.com/login.aspx?direct=true&db=aph&an=13724940>.
- E. De Herdt, H. Slegers, and M. Kondo (1979). “Identification and characterization of a 19S complex containing a 27,000-Mr protein in *Artemia salina*”. *Eur J Biochem*, **96** (3): pp. 423–430. URL <http://search.epnet.com/login.aspx?direct=true&db=aph&an=13674618>.
- G. De Lanversin and B. Jacq (1983). “Sequence of the central break region of the precursor of *Drosophila* 26S ribosomal RNA”. *C R Seances Acad Sci III*, **296** (22): pp. 1041–1044.
- S. R. Dickman, J. T. Madison, and R. L. Holtzer (1962). “Preparation and properties of beef pancreas microsomal fraction”. *Biochemistry*, **1**: pp. 568–574.
- P. Dube, G. Bacher, H. Stark, F. Mueller, F. Zemlin, M. van Heel, and R. Brimacombe (1998a). “Correlation of the expansion segments in mammalian rRNA with the fine structure of the 80S ribosome; a cryo-electron microscopic reconstruction of the rabbit reticulocyte ribosome at 21 Å resolution”. *J Mol Biol*, **279** (2): pp. 403–421. <http://dx.doi.org/10.1006/jmbi.1998.1804>.

- P. Dube, M. Wieske, H. Stark, M. Schatz, J. Stahl, F. Zemlin, G. Lutsch, and M. van Heel (1998b). “The 80S rat liver ribosome at 25 Å resolution by electron cryomicroscopy and angular reconstitution”. *Structure*, **6** (3): pp. 389–399.
- J. Engberg and H. Nielsen (1990). “Complete sequence of the extrachromosomal rDNA molecule from the ciliate *Tetrahymena thermophila* strain B1868VII”. *Nucleic Acids Res*, **18** (23): pp. 6915–6919. GenBank Accession Number: X54512.
- C. Eriksen and D. Belk (1999). *Fairy Shrimps of California’s Puddles, Pools and Playas*. Mad River Press. ISBN 0916422836.
- J. Frank, A. Verschoor, and M. Boulik (1981). “Computer averaging of electron micrographs of 40S ribosomal subunits”. *Science*, **214** (4527): pp. 1353–1355.
- H. Fujiwara and H. Ishikawa (1986). “Molecular mechanism of introduction of the hidden break into the 28S rRNA of insects: Implication based on structural studies”. *Nucleic Acids Res*, **14** (16): pp. 6393–6401.
- H. Gao, M. J. Ayub, M. J. Levin, and J. Frank (2005). “The structure of the 80S ribosome from *Trypanosoma cruzi* reveals unique rRNA components”. *PNAS*, **102** (29): pp. 10206–10211. <http://dx.doi.org/10.1073/pnas.0500926102>.
- S. A. Gerbi (1996). “Expansion segments: Regions of variable size that interrupt the universal core secondary structure of ribosomal RNA”. In R. A. Zimmermann and A. E. Dahlberg, eds., “Ribosomal RNA: Structure, evolution, processing, and function in protein biosynthesis”, pp. 71–87. CRL Press, Boca Raton, FL. ISBN 0849388643.
- C. Glotz, J. Mussig, H. S. Gewitz, I. Makowski, T. Arad, A. Yonath, and H. G. Wittmann (1987). “Three-dimensional crystals of ribosomes and their subunits from eu- and archaebacteria”. *Biochem Int*, **15** (5): pp. 953–960.
- A. Golub and J. S. Clegg (1968). “Protein synthesis in *Artemia salina* embryos. I. Studies on polyribosomes”. *Dev Biol*, **17** (6): pp. 644–656.
- M. G. Gomez-Lorenzo, C. M. Spahn, R. K. Agrawal, R. A. Grassucci, P. Penczek, K. Chakraburty, J. P. Ballesta, J. L. Lavandera, J. F. Garcia-Bustos, and J. Frank (2000). “Three-dimensional cryo-electron microscopy localization of EF2 in the *Saccharomyces cerevisiae* 80S ribosome at 17.5 Å resolution”. *EMBO J*, **19** (11): pp. 2710–2718.
- I. L. Gonzalez, J. L. Gorski, T. J. Campen, D. J. Dorney, J. M. Erickson, J. E. Sylvester, and R. D. Schmickel (1985). “Variation among human 28S ribosomal RNA genes”. *PNAS*, **82** (22): pp. 7666–7670.
- D. J. Goss and T. Harrigan (1986). “Magnesium ion dependent equilibria, kinetics, and thermodynamic parameters of Artemia ribosome dissociation and subunit association”. *Biochemistry*, **25** (12): pp. 3690–3695.
- D. J. Goss, D. Rounds, T. Harrigan, C. L. Woodley, and A. J. Wahba (1988). “Effects of eucaryotic initiation factor 3 on eucaryotic ribosomal subunit equilibrium and kinetics”. *Biochemistry*, **27** (5): pp. 1489–1494.
- H. Grosfeld and U. Z. Littauer (1976). “The translation *in vitro* of mRNA from developing cysts of *Artemia salina*”. *Eur J Biochem*, **70** (2): pp. 589–599.

- M. Halic, T. Becker, J. Frank, C. M. T. Spahn, and R. Beckmann (2005). “Localization and dynamic behavior of ribosomal protein L30e”. *Nat Struct Mol Biol*, **12** (5): pp. 467–468.
- M. Halic, T. Becker, M. R. Pool, C. M. T. Spahn, R. A. Grassucci, J. Frank, and R. Beckmann (2004). “Structure of the signal recognition particle interacting with the elongation-arrested ribosome”. *Nature*, **427** (6977): pp. 808–814.
- M. Halic, M. Gartmann, O. Schlenker, T. Mielke, M. R. Pool, I. Sinning, and R. Beckmann (2006). “Signal recognition particle receptor exposes the ribosomal translocon binding site”. *Science*, **312** (5774): pp. 745–747.
- G. Harauz and M. van Heel (1986). “Exact filters for general geometry three-dimensional reconstruction.” *Optik*, **73**: pp. 146–156.
- B. Hardesty, W. McKeehan, and W. Culp (1971). “Aminoacyl transfer RNA binding enzyme (T-I) from rabbit reticulocytes”. *Methods in Enzymology*, **20**: pp. 316–330. [http://dx.doi.org/10.1016/S0076-6879\(71\)20035-5](http://dx.doi.org/10.1016/S0076-6879(71)20035-5).
- G. Houge and S. O. Doskeland (1996). “Divergence towards a dead end? Cleavage of the divergent domains of ribosomal RNA in apoptosis”. *Experientia*, **52** (10-11): pp. 963–967.
- G. Houge, B. Robaye, T. S. Eikhom, J. Golstein, G. Mellgren, B. T. Gjertsen, M. Lanotte, and S. O. Doskeland (1995). “Fine mapping of 28S rRNA sites specifically cleaved in cells undergoing apoptosis”. *Mol Cell Biol*, **15** (4): pp. 2051–62.
- T. Hultin and J. E. Morris (1968). “The ribosomes of encysted embryos of *Artemia salina* during cryptobiosis and resumption of development”. *Dev Biol*, **17** (2): pp. 143–164.
- H. Ishikawa (1973). “Comparative studies on the thermal stability of animal ribosomal RNA’s”. *Comp Biochem Physiol B*, **46** (2): pp. 217–227.
- H. Ishikawa (1975a). “Comparative studies on the thermal stability of animal ribosomal RNA’s. II. Sea-anemones (*Coelenterata*)”. *Comp Biochem Physiol B*, **50** (1): pp. 1–4.
- H. Ishikawa (1975b). “Comparative studies on the thermal stability of animal ribosomal RNA’s. III. Sponge (Porifera) and the other species (*Tetrahymena* and *Lachnus*)”. *Comp Biochem Physiol B*, **51** (1): pp. 81–85.
- P. J. Keller and E. Cohen (1961). “Enzymic composition of some cell fractions of bovine pancreas”. *J Biol Chem*, **236**: pp. 1407–1413.
- P. J. Keller, E. Cohen, and J. A. Beeley (1968). “Canine pancreatic ribosomes. II. The protein moiety”. *J Biol Chem*, **243** (6): pp. 1271–1276.
- P. J. Keller, E. Cohen, and R. D. Wade (1963). “Bovine pancreatic ribosomes. I. Preparation and some properties”. *Biochemistry*, **2**: pp. 315–321.
- P. J. Keller, E. Cohen, and R. D. Wade (1964). “Bovine pancreatic ribosomes. II. Purification and some properties”. *J Biol Chem*, **239**: pp. 3292–3298.
- D. J. Klein, T. M. Schmeing, P. B. Moore, and T. A. Steitz (2001). “The kink-turn: A new RNA secondary structure motif”. *Embo J*, **20** (15): pp. 4214–21. <http://dx.doi.org/10.1093/emboj/20.15.4214>.

- Y. Kress, M. Wittner, and R. M. Rosenbaum (1971). “Sites of cytoplasmic ribonucleoprotein-filament assembly in relation to helical body formation in axenic trophozoites of *Entamoeba histolytica*”. *J Cell Biol*, **49** (3): pp. 773–784.
- B. A. Kuo, I. L. Gonzalez, D. A. Gillespie, and J. E. Sylvester (1996). “Human ribosomal RNA variants from a single individual and their expression in different tissues”. *Nucleic Acids Res*, **24** (23): pp. 4817–4824.
- J. A. Lake, E. Henderson, M. W. Clark, and A. T. Matheson (1982). “Mapping evolution with ribosome structure: Intralineage constancy and interlineage variation”. *PNAS*, **79** (19): pp. 5948–5952.
- J. A. Lake, E. Henderson, M. Oakes, and M. W. Clark (1984). “Eocytes: A new ribosome structure indicates a kingdom with a close relationship to eukaryotes”. *PNAS*, **81** (12): pp. 3786–3790.
- P. Liang, R. Amons, J. S. Clegg, and T. H. MacRae (1997a). “Molecular characterization of a small heat shock/α-crystallin protein in encysted *Artemia* embryos”. *J Biol Chem*, **272** (30): pp. 19051–19058.
- P. Liang, R. Amons, T. H. Macrae, and J. S. Clegg (1997b). “Purification, structure and *in vitro* molecular-chaperone activity of *Artemia* p26, a small heat-shock/α-crystallin protein”. *Eur J Biochem*, **243** (1-2): pp. 225–232.
- X. Ma, K. Jamil, T. H. Macrae, J. S. Clegg, J. M. Russell, T. S. Villeneuve, M. Euloth, Y. Sun, J. H. Crowe, F. Tablin, and A. E. Oliver (2005). “A small stress protein acts synergistically with trehalose to confer desiccation tolerance on mammalian cells”. *Cryobiology*, **51** (1): pp. 15–28.
- B. E. Maden, C. L. Dent, T. E. Farrell, J. Garde, F. S. McCallum, and J. A. Wakeman (1987). “Clones of human ribosomal DNA containing the complete 18S rRNA and 28S rRNA genes. Characterization, a detailed map of the human ribosomal transcription unit and diversity among clones”. *Biochem J*, **246** (2): pp. 519–527.
- J. M. Mallatt, J. R. Garey, and J. W. Shultz (2004). “Ecdysozoan phylogeny and Bayesian inference: First use of nearly complete 28S and 18S rRNA gene sequences to classify the arthropods and their kin”. *Mol Phylogenet Evol*, **31** (1): pp. 178–191. GenBank Accession Number: AY210805.
- A. L. Manuell, K. Yamaguchi, P. A. Haynes, R. A. Milligan, and S. P. Mayfield (2005). “Composition and structure of the 80S ribosome from the green alga *Chlamydomonas reinhardtii*: 80S ribosomes are conserved in plants and animals”. *J Mol Biol*, **351** (2): pp. 266–279. <http://dx.doi.org/10.1016/j.jmb.2005.06.022>.
- J. F. Menetret, R. S. Hegde, S. U. Heinrich, P. Chandramouli, S. J. Ludtke, T. A. Rapoport, and C. W. Akey (2005). “Architecture of the ribosome-channel complex derived from native membranes”. *J Mol Biol*, **348** (2): pp. 445–457.
- J. F. Menetret, A. Neuhof, D. G. Morgan, K. Plath, M. Radermacher, T. A. Rapoport, and C. W. Akey (2000). “The structure of ribosome-channel complexes engaged in protein translocation”. *Mol Cell*, **6** (5): pp. 1219–1232.
- R. A. Milligan and P. N. Unwin (1982). “*In vitro* crystallization of ribosomes from chick embryos”. *J Cell Biol*, **95** (2 Pt 1): pp. 648–653.

- A. Moreno, R. Mendez, and C. de Haro (1991). “Characterization of cell-free protein-synthesis systems from undeveloped and developing *Artemia* embryos”. *Biochem J*, **276** (3): pp. 809–816.
- D. G. Morgan, J.-F. Menetret, A. Neuhof, T. A. Rapoport, and C. W. Akey (2002). “Structure of the mammalian ribosome-channel complex at 17 Å resolution”. *J Mol Biol*, **324** (4): pp. 871–886.
- D. G. Morgan, J. F. Menetret, M. Radermacher, A. Neuhof, I. V. Akey, T. A. Rapoport, and C. W. Akey (2000). “A comparison of the yeast and rabbit 80S ribosome reveals the topology of the nascent chain exit tunnel, inter-subunit bridges and mammalian rRNA expansion segments”. *J Mol Biol*, **301** (2): pp. 301–321. <http://dx.doi.org/10.1006/jmbi.2000.3947>.
- A. Nakao, M. Yoshihama, and N. Kenmochi (2004). “RPG: The ribosomal protein gene database”. *Nucleic Acids Res*, **32** (Database issue): pp. 168–170.
- O. Namy, S. J. Moran, D. I. Stuart, R. J. C. Gilbert, and I. Brierley (2006). “A mechanical explanation of RNA pseudoknot function in programmed ribosomal frameshifting”. *Nature*, **441** (7090): pp. 244–247.
- R. N. Nazar (1980). “A 5.8S rRNA-like sequence in prokaryotic 23S rRNA”. *FEBS Lett*, **119** (2): pp. 212–214.
- L. Nelles, B. L. Fang, G. Volckaert, A. Vandenberghe, and R. De Wachter (1984a). “Nucleotide sequence of a crustacean 18S ribosomal RNA gene and secondary structure of eukaryotic small subunit ribosomal RNAs”. *Nucleic Acids Res*, **12** (23): pp. 8749–8768. GenBank Accession Number: X01723.
- L. Nelles, C. Van Broeckhoven, R. De Wachter, and A. Vandenberghe (1984b). “Location of the hidden break in large subunit ribosomal RNA of *Artemia salina*”. *Naturwissenschaften*, **71** (12): pp. 634–635.
- P. Nieuwenhuysen and J. Clauwaert (1981). “Physicochemical characterization of ribosomal particles from the eukaryote *Artemia*”. *J Biol Chem*, **256** (18): pp. 9626–9632.
- M. O. Nilsson and T. Hultin (1974). “Characteristics and intracellular distribution of messengerlike RNA in encysted embryos of *Artemia salina*”. *Dev Biol*, **38** (1): pp. 138–149.
- H. F. Noller (1984). “Structure of ribosomal RNA”. *Annu Rev Biochem*, **53**: pp. 119–162.
- L. O’Brien, K. Shelley, J. Towfighi, and A. McPherson (1980). “Crystalline ribosomes are present in brains from senile humans”. *PNAS*, **77** (4): pp. 2260–2264.
- P. Penczek (1998). “Measures of resolution using Fourier shell correlation.” *J Mol Bio.*, **280**: pp. 115–116.
- P. S. Perkins and S. J. Pandol (1992). “Cholecystokinin-induced changes in polysome structure regulate protein synthesis in pancreas”. *Biochim Biophys Acta*, **1136** (3): pp. 265–271.
- R. J. Planta and W. H. Mager (1998). “The list of cytoplasmic ribosomal proteins of *Saccharomyces cerevisiae*”. *Yeast*, **14** (5): pp. 471–477.
- Z. Qiu, R. I. Viner, T. H. MacRae, J. K. Willsie, and J. S. Clegg (2004). “A small heat shock protein from *Artemia franciscana* is phosphorylated at serine 50”. *Biochim Biophys Acta*, **1700** (1): pp. 75–83.

- M. Radermacher, T. Wagenknecht, A. Verschoor, and J. Frank (1987). “Three-dimensional reconstruction from a single-exposure, random conical tilt series applied to the 50S ribosomal subunit of *Escherichia coli*”. *J Microsc*, **146** (Pt 2): pp. 113–136.
- A. Raikar, H. M. Rubino, and R. E. Lockard (1988). “Chemical probing of adenine residues within the secondary structure of rabbit 18S ribosomal RNA”. *Biochemistry*, **27** (2): pp. 582–592.
- M. A. Reddington, A. P. Fong, and W. P. Tate (1978). “The termination of the synthesis of proteins *in vitro* with extracts from the undeveloped cyst of *Artemia salina*”. *Dev Biol*, **63** (2): pp. 402–411.
- M. P. Roberts and J. C. Vaughn (1982). “Ribosomal RNA sequence conservation and gene number in the larval brine shrimp”. *Biochim Biophys Acta*, **697** (2): pp. 148–155.
- G. Scheele (1983). “Methods for the study of protein translocation across the RER membrane using the reticulocyte lysate translation system and canine pancreatic microsomal membranes”. *Methods Enzymol*, **96**: pp. 94–111.
- F. Schluenzen, A. Tocilj, R. Zarivach, J. Harms, M. Gluehmann, D. Janell, A. Bashan, H. Bartels, I. Agmon, F. Franceschi, and A. Yonath (2000). “Structure of functionally activated small ribosomal subunit at 3.3 Å resolution”. *Cell*, **102** (5): pp. 615–623.
- B. S. Schuwirth, M. A. Borovinskaya, C. W. Hau, W. Zhang, A. Vila-Sanjurjo, J. M. Holton, and J. H. Cate (2005). “Structures of the bacterial ribosome at 3.5 Å resolution”. *Science*, **310** (5749): pp. 827–834.
- B. W. Segelke (2001). “Efficiency Analysis of Screening Protocols Used in Protein Crystallization”. *Journal of Crystal Growth*, **232**: pp. 553–562. URL <http://oldcrys.axygenbio.com>.
- M. Selmer, C. M. Dunham, F. V. t. Murphy, A. Weixlbaumer, S. Petry, A. C. Kelley, J. R. Weir, and V. Ramakrishnan (2006). “Structure of the 70S ribosome complexed with mRNA and tRNA”. *Science*, **313** (5795): pp. 1935–1942.
- A. Shevack, H. S. Gewitz, B. Hennemann, A. Yonath, and H. G. Wittmann (1985). “Characterization and Crystallization of Ribosomal Particles from *Halobacterium marismortui*”. *FEBS Letters*, **184** (1): pp. 68–71.
- J. M. Sierra, W. Filipowicz, and S. Ochoa (1976). “Messenger RNA in undeveloped and developing *Artemia salina* embryos”. *Biochem Biophys Res Commun*, **69** (1): pp. 181–189.
- L. I. Slobin and W. Möller (1976). “Characterization of Developmentally Regulated Forms of Elongation Factor 1 in *Artemia salina*”. *Eur J Biochem*, **69** (2): pp. 351–366. <http://dx.doi.org/10.1111/j.1432-1033.1976.tb10919.x>.
- C. M. Spahn, R. Beckmann, N. Eswar, P. A. Penczek, A. Sali, G. Blobel, and J. Frank (2001a). “Structure of the 80S ribosome from *Saccharomyces cerevisiae*–tRNA-ribosome and subunit–subunit interactions”. *Cell*, **107** (3): pp. 373–386.
- C. M. Spahn, J. S. Kieft, R. A. Grassucci, P. A. Penczek, K. Zhou, J. A. Doudna, and J. Frank (2001b). “Hepatitis C virus IRES RNA-induced changes in the conformation of the 40S ribosomal subunit”. *Science*, **291** (5510): pp. 1959–1962.

- C. M. T. Spahn, M. G. Gomez-Lorenzo, R. A. Grassucci, R. Jorgensen, G. R. Andersen, R. Beckmann, P. A. Penczek, J. P. G. Ballesta, and J. Frank (2004a). “Domain movements of elongation factor eEF2 and the eukaryotic 80S ribosome facilitate tRNA translocation”. *EMBO J*, **23** (5): pp. 1008–1019.
- C. M. T. Spahn, E. Jan, A. Mulder, R. A. Grassucci, P. Sarnow, and J. Frank (2004b). “Cryo-EM visualization of a viral internal ribosome entry site bound to human ribosomes: The IRES functions as an RNA-based translation factor”. *Cell*, **118** (4): pp. 465–475.
- S. Srivastava, A. Verschoor, M. Radermacher, R. Grassucci, and J. Frank (1995). “Three-dimensional reconstruction of mammalian 40S ribosomal subunit embedded in ice”. *J Mol Biol*, **245** (5): pp. 461–466.
- Y. Sun, M. Mansour, J. A. Crack, G. L. Gass, and T. H. MacRae (2004). “Oligomerization, chaperone activity, and nuclear localization of p26, a small heat shock protein from *Artemia franciscana*”. *J Biol Chem*, **279** (38): pp. 39999–40006.
- R. Sweeney, L. Chen, and M. C. Yao (1994). “An rRNA variable region has an evolutionarily conserved essential role despite sequence divergence”. *Mol Cell Biol*, **14** (6): pp. 4203–4215.
- W. P. Tate and C. J. Marshall (1991). “Post-dormancy transcription and translation in the brine shrimp”. In R. A. Brown, P. Sorgeloos, and C. N. A. Trotman, eds., “*Artemia biology*”, p. 24. CRC Press, Boca Raton, FL.
- D. Tautz, C. Tautz, D. Webb, and G. A. Dover (1987). “Evolutionary divergence of promoters and spacers in the rDNA family of four *Drosophila* species. Implications for molecular coevolution in multigene families”. *J Mol Biol*, **195** (3): pp. 525–542. GenBank Accession Number: M21017.
- A. Tissieres and J. D. Watson (1958). “Ribonucleoprotein particles from *Escherichia coli*”. *Nature*, **182** (4638): pp. 778–780.
- S. Trakhanov, M. Yusupov, V. Shirokov, M. Garber, A. Mitschler, M. Ruff, J. C. Thierry, and D. Moras (1989). “Preliminary X-ray investigation of 70S ribosome crystals from *Thermus thermophilus*”. *J Mol Biol*, **209** (2): pp. 327–328.
- T. Uchiumi, A. J. Wahba, and R. R. Traut (1987). “Topography and stoichiometry of acidic proteins in large ribosomal subunits from *Artemia salina* as determined by crosslinking”. *PNAS*, **84** (16): pp. 5580–5584.
- P. N. Unwin and C. Taddei (1977). “Packing of ribosomes in crystals from the lizard *Lacerta sicula*”. *J Mol Biol*, **114** (4): pp. 491–506.
- M. Valle, A. Zavialov, W. Li, S. M. Stagg, J. Sengupta, R. C. Nielsen, P. Nissen, S. C. Harvey, M. Ehrenberg, and J. Frank (2003). “Incorporation of aminoacyl-tRNA into the ribosome as seen by cryo-electron microscopy”. *Nat Struct Bio*, **10** (11): pp. 899–906.
- H. van Keulen, P. M. Mertz, P. T. LoVerde, H. Shi, and D. M. Rekosh (1991). “Characterization of a 54-nucleotide gap region in the 28S rRNA gene of *Schistosoma mansoni*”. *Mol Biochem Parasitol*, **45** (2): pp. 205–214. GenBank Accession Number: X13836.
- R. W. van Nues, J. Venema, R. J. Planta, and H. A. Raue (1997). “Variable region V1 of *Saccharomyces cerevisiae* 18S rRNA participates in biogenesis and function of the small ribosomal subunit”. *Chromosoma*, **105** (7-8): pp. 523–531.

- G. M. Veldman, J. Klootwijk, V. C. de Regt, R. J. Planta, C. Branst, A. Krol, and J. P. Ebel (1981). “The primary and secondary structure of yeast 26S rRNA”. *Nucleic Acids Res*, **9** (24): pp. 6935–6952.
- A. Verschoor and J. Frank (1990). “Three-dimensional structure of the mammalian cytoplasmic ribosome”. *J Mol Biol*, **214** (3): pp. 737–749.
- A. Verschoor, S. Srivastava, R. Grassucci, and J. Frank (1996). “Native 3D structure of eukaryotic 80S ribosome: Morphological homology with *E. coli* 70S ribosome”. *J Cell Biol*, **133** (3): pp. 495–505.
- A. Verschoor, J. R. Warner, S. Srivastava, R. A. Grassucci, and J. Frank (1998). “Three-dimensional structure of the yeast ribosome”. *Nucleic Acids Res*, **26** (2): pp. 655–661.
- A. Verschoor, N. Y. Zhang, T. Wagenknecht, T. Obrig, M. Radermacher, and J. Frank (1989). “Three-dimensional reconstruction of mammalian 40S ribosomal subunit”. *J Mol Biol*, **209** (1): pp. 115–126.
- P. Walter and G. Blobel (1983). “Preparation of microsomal membranes for cotranslational protein translocation”. *Methods Enzymol*, **96**: pp. 84–93.
- V. C. Ware, R. Renkawitz, and S. A. Gerbi (1985). “rRNA processing: Removal of only nineteen bases at the gap between 28S α and 28S β rRNAs in *Sciara coprophila*”. *Nucleic Acids Res*, **13** (10): pp. 3581–3597. GenBank Accession Number: X02482.
- V. C. Ware, B. W. Tague, C. G. Clark, R. L. Gourse, R. C. Brand, and S. A. Gerbi (1983). “Sequence analysis of 28S ribosomal DNA from the amphibian *Xenopus laevis*”. *Nucleic Acids Res*, **11** (22): pp. 7795–7817.
- A. H. Warner, R. T. Brunet, T. H. MacRae, and J. S. Clegg (2004). “Artemin is an RNA-binding protein with high thermal stability and potential RNA chaperone activity”. *Arch Biochem Biophys*, **424** (2): pp. 189–200. <http://dx.doi.org/10.1016/j.abb.2004.02.022>.
- P. H. H. Weekers, G. Murugan, J. R. Vanfleteren, D. Belk, and H. J. Dumont (2002). “Phylogenetic analysis of anostracans (Branchiopoda: Anostraca) inferred from nuclear 18S ribosomal DNA (18S rDNA) sequences”. *Mol Phylogenet Evol*, **25** (3): pp. 535–544. GenBank Accession Number: AJ238061.
- R. Wheeler (2005a). “The process of initiation of translation in eukaryotes.” URL http://en.wikipedia.org/wiki/Image:Eukaryotic_Translation_Initiation.png. Zephyris at Wikipedia.org and Undergraduate at the University of Cambridge.
- R. Wheeler (2005b). “The process of initiation of translation in prokaryotes.” URL http://en.wikipedia.org/wiki/Image:Prokaryotic_Translation_Initiation.png. Zephyris at Wikipedia.org and Undergraduate at the University of Cambridge.
- B. T. Wimberly, D. E. Brodersen, W. M. J. Clemons, R. J. Morgan-Warren, A. P. Carter, C. Vonrhein, T. Hartsch, and V. Ramakrishnan (2000). “Structure of the 30S ribosomal subunit”. *Nature*, **407** (6802): pp. 327–339.
- H. G. Wittmann, J. Mussig, J. Pieck, H. S. Gewitz, H. J. Rheinberger, and A. Yonath (1982). “Crystallization of *Escherichia coli* ribosomes”. *FEBS Lett*, **146** (1): pp. 217–220.

- I. G. Wool, Y. L. Chan, and A. Gluck (1996). “Mammalian ribosomes: The structure and the evolution of the proteins”. In J. W. B. Hershey, M. B. Matthews, and N. Sonenberg, eds., “Translational Control”, pp. 685–732. Cold Spring Harbor Press, Cold Spring Harbor, NY. ISBN 0879694580.
- A. Yonath, J. Mussig, and H. G. Wittmann (1982). “Parameters for crystal growth of ribosomal subunits”. *J Cell Biochem*, **19** (2): pp. 145–155.
- A. Yonath and H. G. Wittmann (1988). “Approaching the molecular structure of ribosomes”. *Biophys Chem*, **29** (1-2): pp. 17–29.
- A. E. Yonath, J. Mussig, B. Tesche, S. Lorenz, V. A. Erdmann, and H. G. Wittmann (1980). “Crystallization of the Large Ribosomal Subunits from *Bacillus stearothermophilus*”. *Biochemistry International*, **1** (5): pp. 428–435.
- D. S. Zarlenga and J. B. Dame (1992). “The identification and characterization of a break within the large subunit ribosomal RNA of *Trichinella spiralis*: Comparison of gap sequences within the genus”. *Mol Biochem Parasitol*, **51** (2): pp. 281–289. GenBank Accession Number: M74173.
- M. Zasloff and S. Ochoa (1974). “Purification of eukaryotic initiation factor 1 (eIF1) from *Artemia salina* embryos”. *Methods Enzymol*, **30** (0): pp. 197–206.
- E. N. Zendzian and E. A. Barnard (1967). “Distributions of Pancreatic Ribonuclease, Chymotrypsin, and Trypsin in Vertebrates”. *Arch Biochem Biophys.*, **122** (3): pp. 699–713.
- M. Zuker (2003). “MFOLD web server for nucleic acid folding and hybridization prediction”. *Nucleic Acids Res*, **31** (13): pp. 3406–3415.

Appendix A

Rolling Probe Technique for Calculating Excluded Volumes

The rolling probe method described in Chapter 3 was introduced in 1977 by Richards (Figure 3.1, page 51). This algorithm was initially done in two-dimensions using a grid. The accessible volume was calculated in two-dimensions by drawing circles. The radius of the circle is taken as the radius of each atom plus the spherical probe radius as intersected by a plane in three-dimensions. Because many atoms lie above or below the plane of interest, the effective radii of the atoms is always less than the sum of the atomic radius and the probe radius. Figure A.1 shows the accessible surface in two-dimensions of a molecule (Lee & Richards, 1971). The excluded surface is then calculated by subtracting the probe radius from every point along the surface.

A.1 Analytical technique

An analytical algorithm for extending the rolling probe into three-dimensions was independently solved by Connolly (1983) and Richmond (1984). Several advancements were made to this algorithm by M. Sanner to make this program fast enough for general use (Sanner *et al.*, 1995; Sanner, 1995; Sanner *et al.*, 1996).

The three-dimensional algorithm works as summarized in Figure A.2. First, the accessible surface is generated for the target molecule. Second, intersections between the overlapping spheres is found. The intersection between two spheres is a circle and between three spheres is a point. Three types of surfaces result from this construction: (1) external spheres, (2) internal tori, and (3) internal spheres. External spheres are created as the surface of the atom. Each VDW sphere

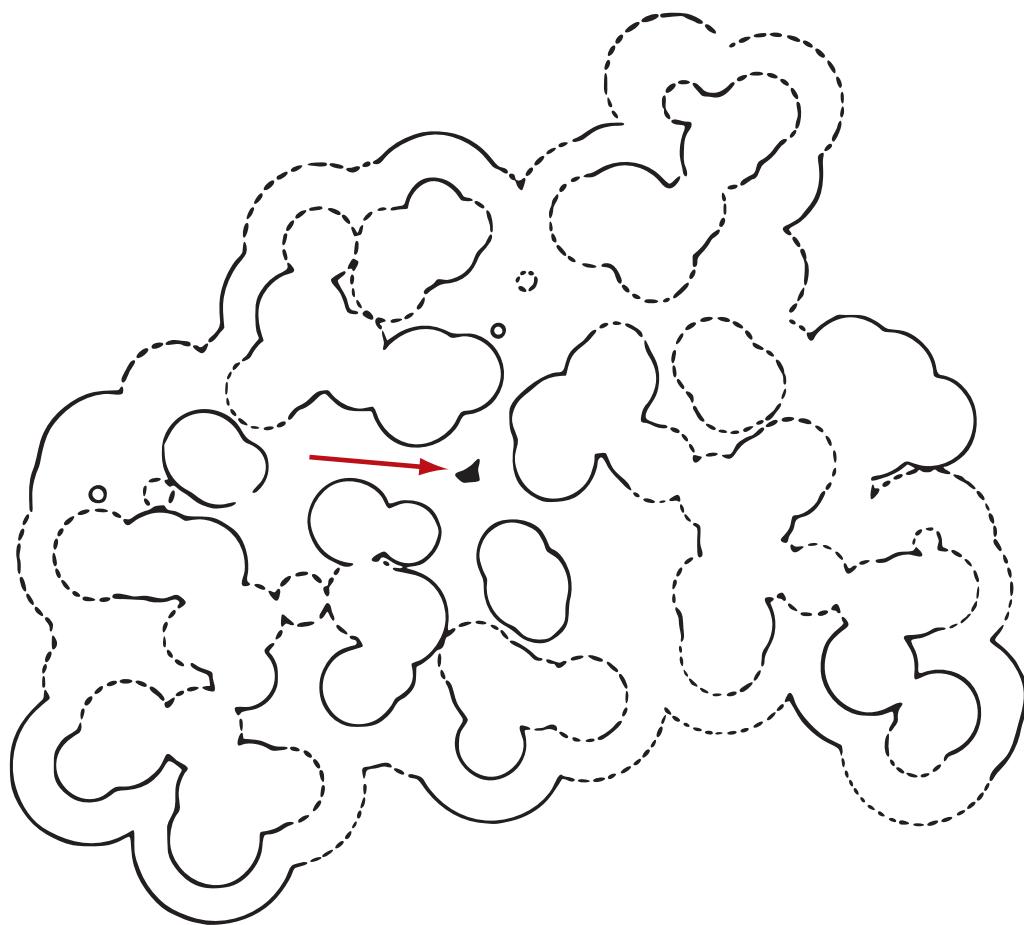


Figure A.1: The van der Waals and solvent accessible contours of a section of the ribonuclease S molecule. The red arrow indicates the cavity inside the ribonuclease S molecule large enough to accommodate a solvent molecule of radius 1.4 Å. Adapted from Lee & Richards (1971).

is an external sphere. When two VDW spheres intersect they form a circle on which the spherical probe rotates. When the probe traces this circle it creates the internal part (“doughnut hole”) of a torus. Last, an internal sphere is created when three accessible-VDW spheres intersect at a pair of points. Each point is subtracted back into an internal sphere. By collecting all three types of surfaces created, the molecular surface is found by intersecting these surfaces and removing any interior portions.

This method produces the exact molecular surface of a molecule, but it is difficult to keep track of so many surfaces. For 3 atoms, this method produces 3 external spheres, 3 internal tori, and 2 internal spheres. For 10 atoms, it produces a maximum of 10 external spheres, 45 internal tori, and 240 internal spheres. Obviously, the complexity of this technique increases rapidly as the number of atoms increases and for the ribosome it is unreasonably complex.

A.2 Discrete technique

Because of the difficulties associated with the analytical method and the failure of previously created software on the ribosome, I re-explored the discrete method of Richards (1977). Instead of using two-dimensions, I expanded the algorithm into three-dimensions. As shown in Figures A.3, the discrete method follows four major steps. (1) Place the accessible volume of the target molecule in a large grid. (2) Locate all the edge voxels, *i.e.* empty grid points on the surface of the molecule. (3) For each empty grid point remove all points within the distance of the probe. (4) The excluded volume of the target molecule is the remaining volume, so just count all the points to get the volume. This method is clearly sensitive to the volume of each individual voxel. Finer grids (see Figure A.4) will produce better results.

Because I use the empty points on the surface for subtracting back the excluded volume from the accessible volume, the resulting volume will always be greater than the exact analytical volume. For example, the exact volume of the excluded volume found in Figure A.2 is 8.20^2 , while the larger grid size of Figure A.3 is 11.4 cm^2 and the smaller grid of Figure A.4 gives a value of 9.16 cm^2 .

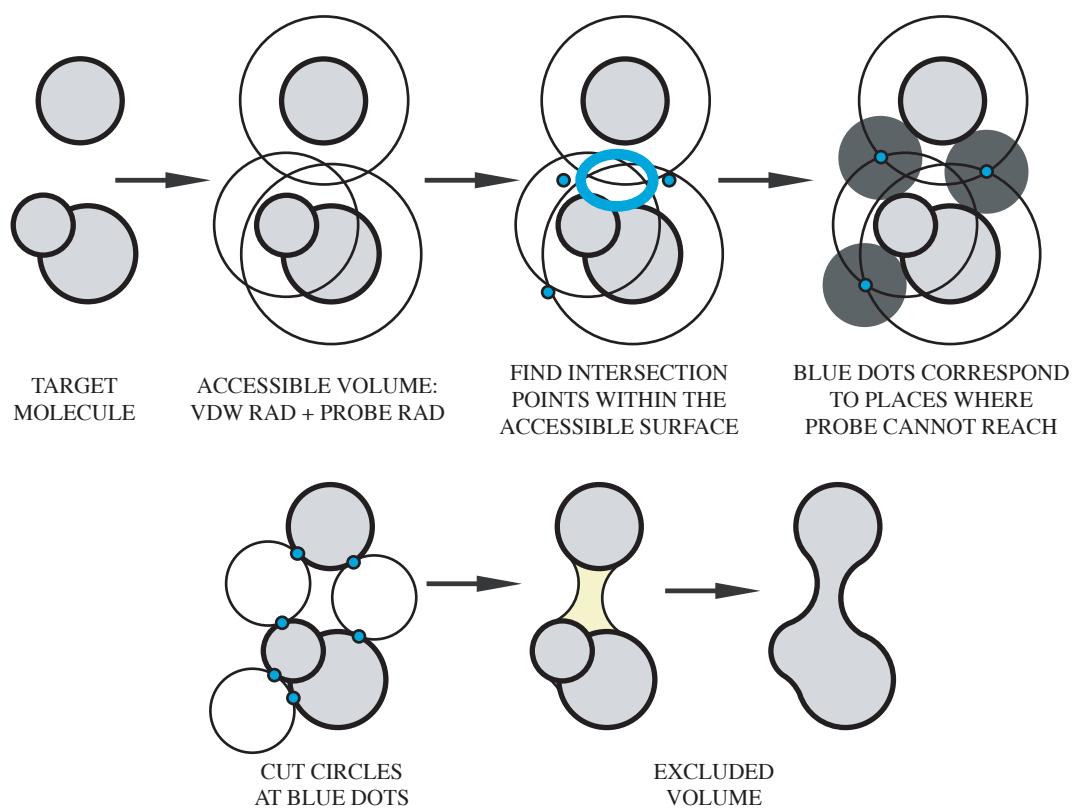


Figure A.2: Analytical rolling probe method.

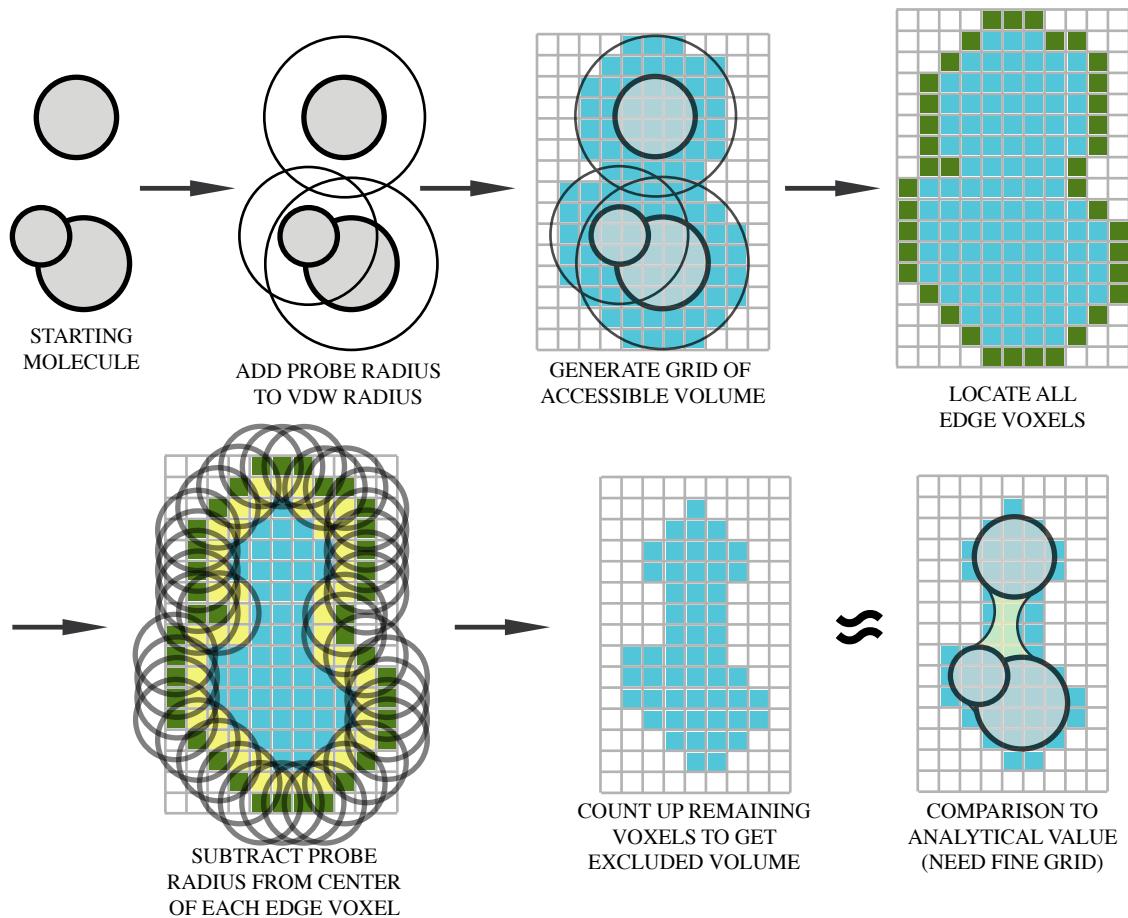


Figure A.3: Discrete rolling probe method.

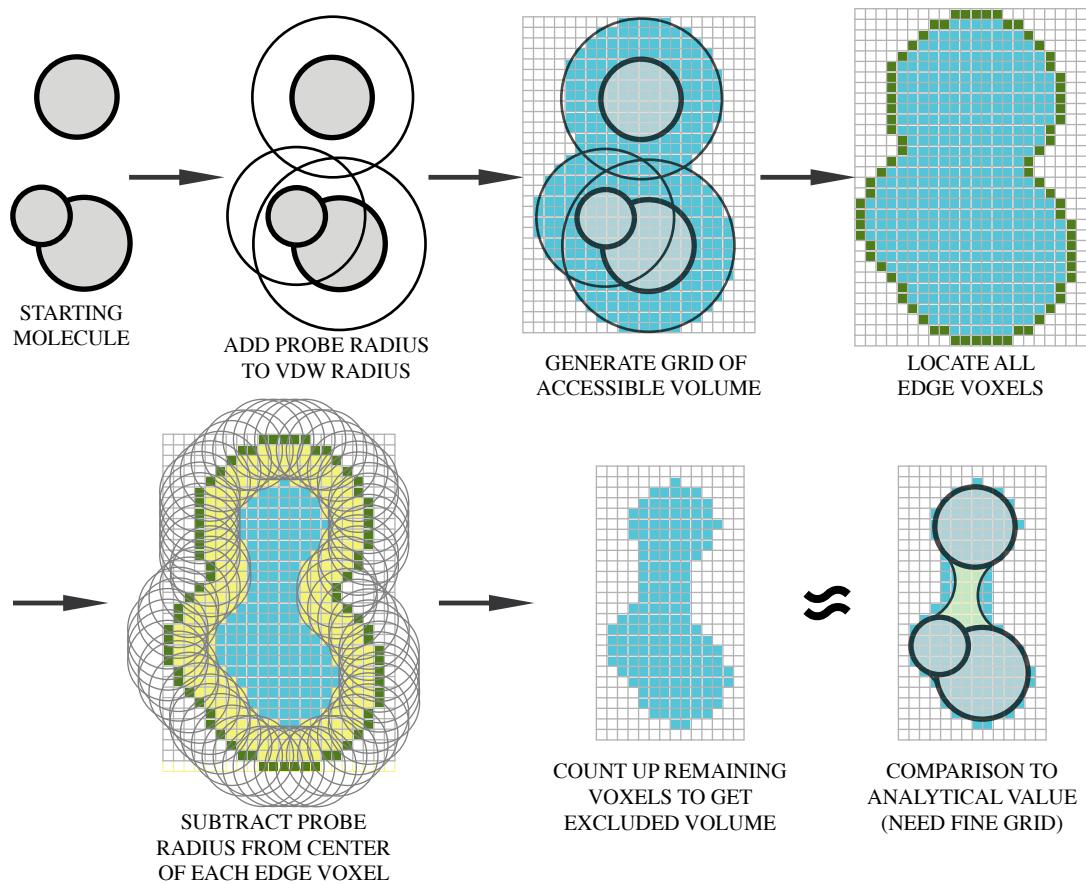


Figure A.4: Discrete rolling probe method with small grid spacings.

A.3 Advantages of the discrete method

One main advantage of the discrete method over the analytical method is grid operations. Imagine trying to subtract the array of surfaces from another set of surfaces provided in the analytical method. This simple request increases the complexity of the analytical method beyond reasonable for any fast analysis of a macromolecule. On the other hand, the discrete method has relatively simple operations. Subtracting one grid from another requires only simple binary math. This subtraction is useful for extracting the interior solvent of a molecule (see Figure 3.7, page 71). Further, points can be selected as distance from a vector, *e.g.* a vector defining the exit tunnel, limiting the processor time required for extraction of the tunnel.

A.4 Surface area calculation

In the analytical method, the surface area is calculated by simply summing the surface area of all the individual surfaces and only the volume calculation is difficult. In contrast, the surface area calculation of the discrete method was something of a daunting task and the volume is straightforward. It is difficult to imagine finding the surface area of a smooth surface using a series of boxes to represent it. Fortunately, this has been a problem of major consideration in the field of magnetic resonance imaging (MRI) brain data. By classifying all surface voxels into nine categories (see Figure A.5), Mullikin & Verbeek (1993); Windreich *et al.* (2003) were able to approximate a surface area. The nine categories only considers the neighbors immediately above-below, front-back and left-right of a given voxel. An alternative algorithm with 13 categories by analyzing 8 voxels for classification has been used for surface estimation (Lindblad, 2003, 2005).

It is simple to obtain a surface area using this technique. The nine categories are each assigned a weighting factor: $S_1 = 0.89$, $S_2 = 1.34$, $S_3 = 1.59$, $S_4 = 2.00$, $S_5 = 2.67$, $S_6 = 3.33$, $S_7 = 1.79$, $S_8 = 2.68$, and $S_9 = 4.08$. The weighting factor is then multiplied by the surface area of one side of a voxel. To get the total surface area of a voxel set, the total number of voxel in each category is multiplied by their weighting factor and the individual voxel surface area and then summed.

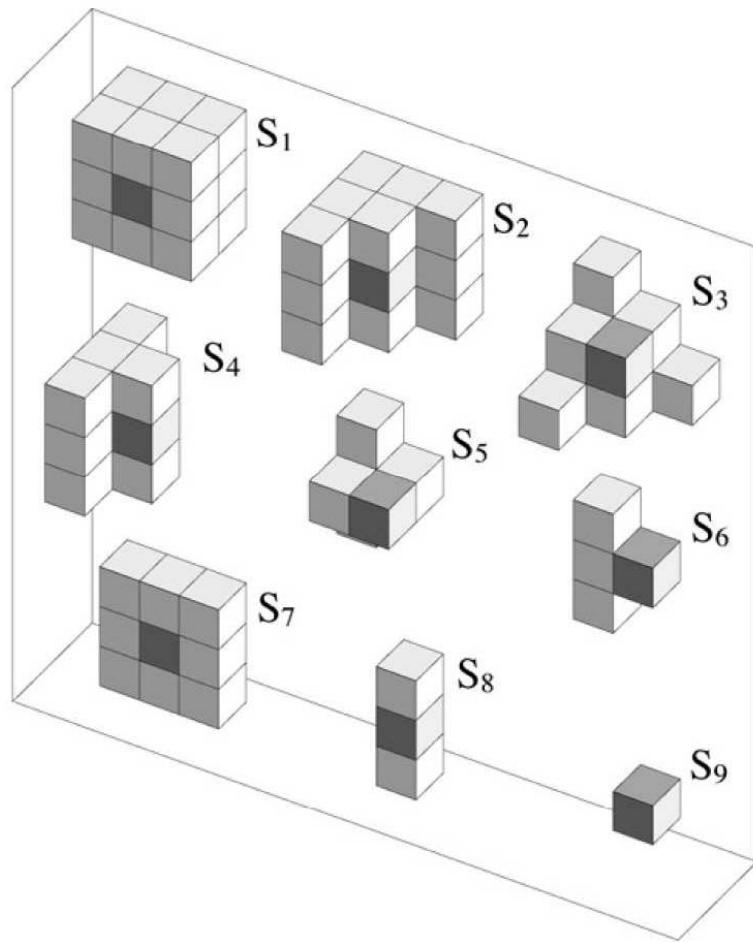


Figure A.5: Surface area calculation for discrete voxels. Mullikin & Verbeek (1993) and Windreich *et al.* (2003) determined a method to calculate the surface area of a set of voxels by assigning all the surface voxels to one of nine different categories each contributing a different amount to the surface area. Figure taken from Windreich *et al.* (2003).

For example, using a Rubik's cube to obtain a surface area. The Rubik's cube has $3 \times 3 \times 3 = 27$ voxels. A Rubik's cube has 6 face voxels, 12 edge voxels, and 8 corner voxels totaling to 26 voxels. The single interior voxel does not contribute to the surface area. The six voxels at the center of each face classified as S_1 , the eight voxels at each corner are S_3 , and the twelve edge pieces are S_2 . So, the estimated surface area is:

$$6 \times 0.89 + 8 \times 1.59 + 12 \times 1.34 = 34.14$$

while this is not the exact value of 54 it is generally close and if we used a finer grid to approximate a cube the estimate gets better.

A.5 References

- M. L. Connolly (1983). “Analytical molecular surface calculation”. *J Appl Cryst*, **16** (5): pp. 548–558. <http://dx.doi.org/10.1107/S0021889883010985>.
- B. Lee and F. M. Richards (1971). “The interpretation of protein structures: Estimation of static accessibility”. *J Mol Biol*, **55** (3): pp. 379–400. [http://dx.doi.org/10.1016/0022-2836\(71\)90324-X](http://dx.doi.org/10.1016/0022-2836(71)90324-X).
- J. Lindblad (2003). “Surface area estimation of digitized planes using weighted local configurations”. In Ingela Nyström and Sanniti di Baja, Gabriella and Stina Svensson, ed., “Lecture Notes in Computer Science”, volume 2886: Discrete geometry for computer imagery, pp. 348–357. Springer, Berlin. ISBN 978-3-540-20499-2.
- J. Lindblad (2005). “Surface area estimation of digitized 3D objects using weighted local configurations”. *Image and Vision Computing*, **23** (2): pp. 111–122.
- J. C. Mullikin and P. W. Verbeek (1993). “Surface area estimation of digitized planes”. *Bioimaging*, **1** (1): pp. 6–16.
- F. M. Richards (1977). “Areas, volumes, packing and protein structure”. *Annu Rev Biophys Bioeng*, **6**: pp. 151–76. <http://dx.doi.org/10.1146/annurev.bb.06.060177.001055>.
- T. J. Richmond (1984). “Solvent accessible surface area and excluded volume in proteins. Analytical equations for overlapping spheres and implications for the hydrophobic effect”. *J Mol Biol*, **178** (1): pp. 63–89.
- M. Sanner, A. J. Olson, and J. C. Spehner (1995). “Fast and robust computation of molecular surfaces.” In “Proc. 11th ACM Symp. Comp. Geom”, pp. C6–C7.
- M. F. Sanner (1995). “MSMS: Michel Sanner’s molecular surface”. URL http://www.scripps.edu/~mb/olson/people/sanner/html/msms_home.html.
- M. F. Sanner, A. J. Olson, and J. C. Spehner (1996). “Reduced surface: An efficient way to compute molecular surfaces”. *Biopolymers*, **38** (3): pp. 305–20.
- G. Windreich, N. Kiryati, and G. Lohmann (2003). “Voxel-based surface area estimation: From theory to practice”. *Pattern Recognition*, **36** (11): pp. 2531–2541.

Appendix B

Alternative Methods to Extract Interior Volumes

Several programs have been written to find interior voids, cavities and channels for macromolecules, but none proved satisfactory for our purposes (Laskowski, 1995; Bakowies & van Gunsteren, 2002; Liang *et al.*, 1998; Laskowski *et al.*, 1996; Smart *et al.*, 1993). Other successful programs like VOIDOO (Kleywegt *et al.*, 2001; Kleywegt & Jones, 1994), which uses a modified version of the rolling probe method, are designed for finding only cavities and therefore cannot extract channels. All methods use one of five algorithms for extracting channels from a macromolecule or structure: (1) space overlapping (2) level sets (3) alpha shapes (4) vector stepping and (5) the rolling probe method (covered in Chapter 3).

B.1 Space Overlapping

“Space overlapping” is the algorithm used by the program, SURFNET (Laskowski, 1995). SURFNET works by placing a probe equidistant between two atoms then increasing the radius of the probe until it intersects the VDW radius of any other atom (Figure B.1). The solvent volume is the union of the probes found for all pairs of atoms (Figure B.2).

Unfortunately, this method does not do a good job distinguishing interior and exterior solvent because it uses the equivalent of the convex hull to define the exterior of the object, which is never a good idea (see Figure 3.2, page 54). Second, it cannot be used to identify and characterize a particular channel *e.g.* the exit tunnel, unless another program is used. And finally, because the algorithm analyzes the volume between pairs of atoms the running time of the computer grows as

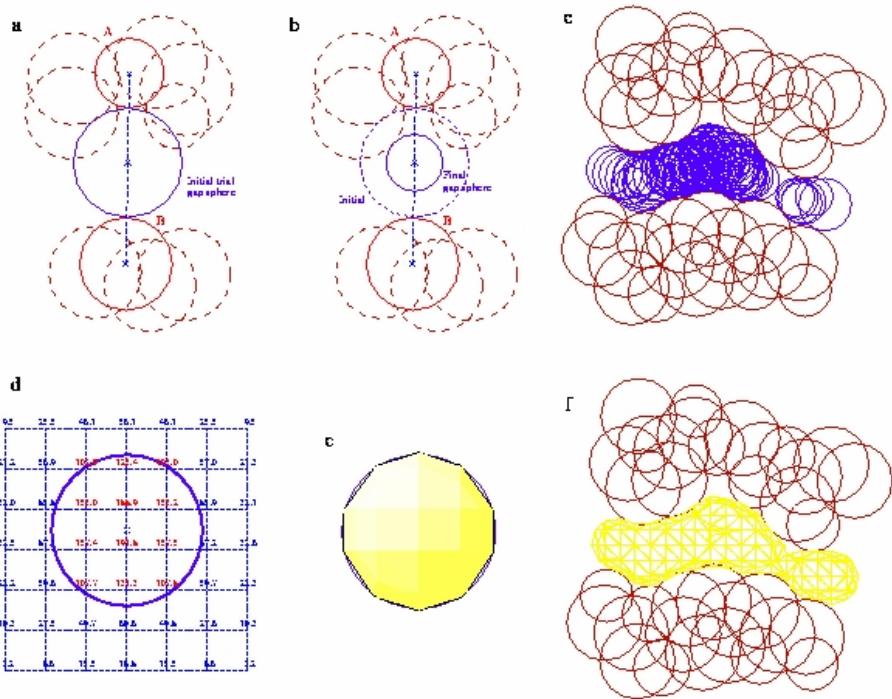


Figure B.1: The space overlapping method. Gap regions are defined by first filling the region between the two molecules with gap-spheres and then using these to compute a 3D density map which, when contoured, defines the surface of the gap region. (a) Two atoms, A and B, have a trial gap sphere placed midway between their van der Waals (VDW) surfaces and just touching each one. (b) Neighboring atoms are then considered in turn. If any penetrate the gap sphere its radius is reduced until it just touches the intruding atom. The process is repeated until all the neighboring atoms have been considered. If the radius of the sphere falls below some predetermined minimum limit (usually 1.0Å) it is rejected. Otherwise, the final gap sphere is saved. (c) The procedure is continued until all pairs of atoms have been considered and the gap region is filled with spheres. (d) The spheres are then used to update points on a 3D array of grid-points using a Gaussian function. (e) The update is such that, when the grid is contoured at a contour level of 100.0, the resultant 3D surface corresponds to each gap sphere. (f) When all the spheres have updated the grid, the final 3D contour represents the surface of the interpenetrating gap spheres, and hence defines the extent of the gap region between the two molecules. (Figure taken from SURFNET website, Laskowski, 1995)

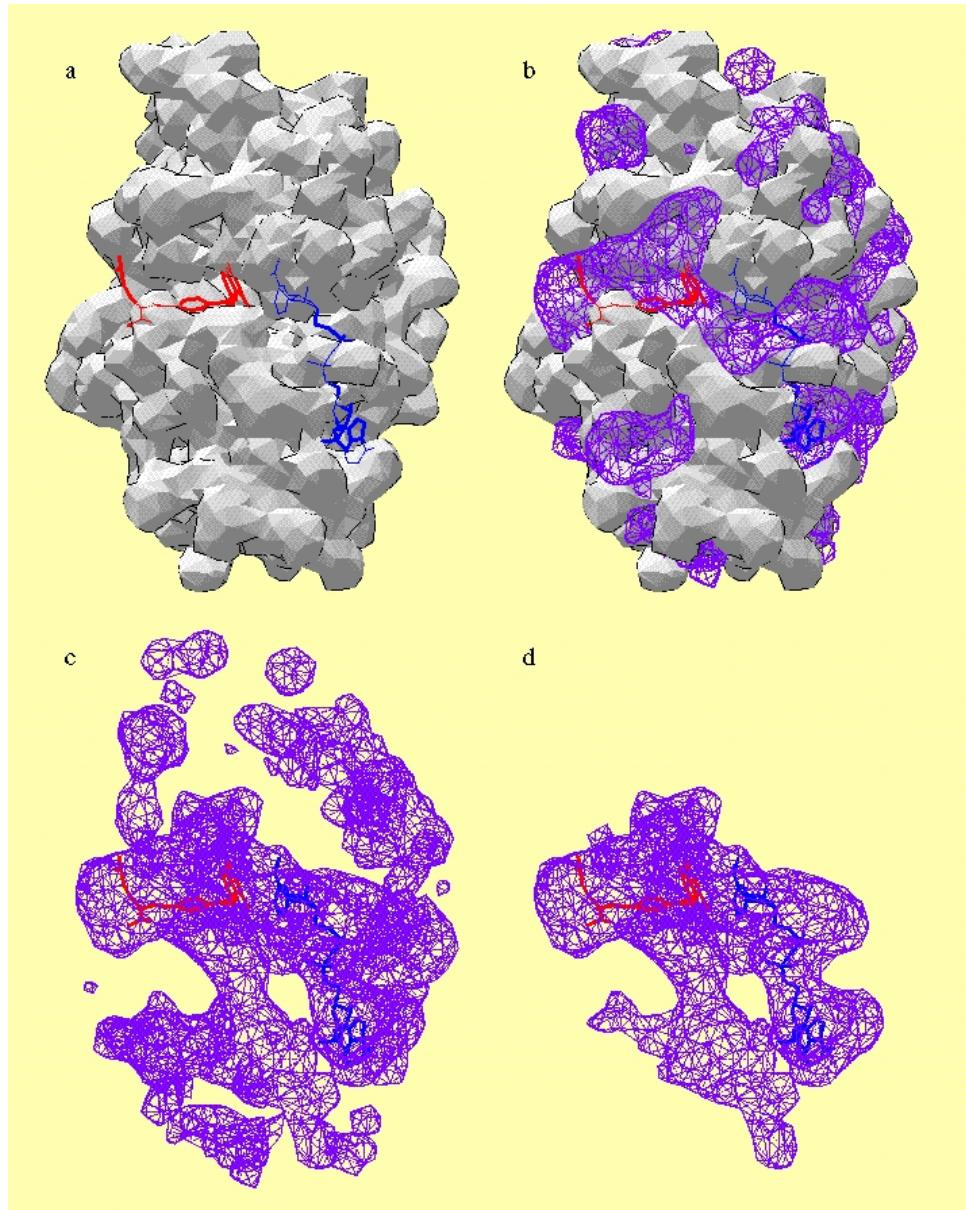


Figure B.2: Example channels from the space overlapping method. The example shown above is the ternary complex of dihydrofolate reductase, methotrexate and NADPH (from PDB code 3dfr). (a) The surface of the protein is shown as a shaded sketch, with the methotrexate molecule (on the left) and NADPH molecule (on the right) emerging into visibility here and there. The invisible parts of the two molecules are represented by dotted lines. (b) All the gap regions (cavities and grooves) in the protein are shown by the purple wire cage surface. They are generated as described here. (c) Just the gap regions, with the protein removed. The two bound molecules can be seen within the largest of the gap regions. (d) The largest of the gap regions, corresponding to the protein's binding site, is shown on its own, extracted from all the rest by the program MASK.

the square of the number of atoms ($O(n^2)$). So, while SURFNET and the space overlapping method is a good technique for finding channels in small macromolecules, it does not scale well to objects the size of the ribosome.

B.2 Active Contours and the Level Set Method

“The level set method” is a numerical technique for isolating interfaces and shapes invented by Osher & Fedkiw (1988). The level set method works by defining an initial surface and maximal curvature allowed by that surface. The level set algorithm then moves (expands or contracts) in the normal direction by the Hamilton-Jacobi equation ($\phi_t = v|\nabla\phi|$) to its surface until it reaches a “road block” or atom in this case. The level set is then forced to expand around the atoms until the maximal curvature is reached or the obstacle is passed. For example, if the maximal curvature is set to zero and the surface is wrapped around the ribosome, then the convex hull of the ribosome will be the result (see Figure 3.2, page 54). The level set method is requires very few parameters: initial surface and curvature. Second, the level set method is very robust, in theory it could be used to define the exit tunnel rather well. Unfortunately, the level set method is difficult to implement because it is very complex and since it is new there are few available software packages that use it and none have been developed for biological applications.

The important feature within Figure B.3 is the outline of the artery. The method looks for places where there is a big jump in intensity between neighboring pixels. Selecting how big of an increase in pixel value is a difficult decision. On one hand, if a small value is used, extra boundaries are created. On the other hand, if too large a value is used, then nothing is considered to be a boundary. Large spikes of noise can also cause other problems. A different approach by Malladi *et al.* (1995) comes from initializing a small circle inside the region of interest, and allowing it to grow outwards until it reaches the desired boundary. The fast marching method was used to propagate the initial seed point outwards, followed by a level set method to fine tune the result. The curve moves outwards with a speed that depends on the steepness of the image gradient. When the curve passes over places where the image gradient is small, the curve front expands quickly. In

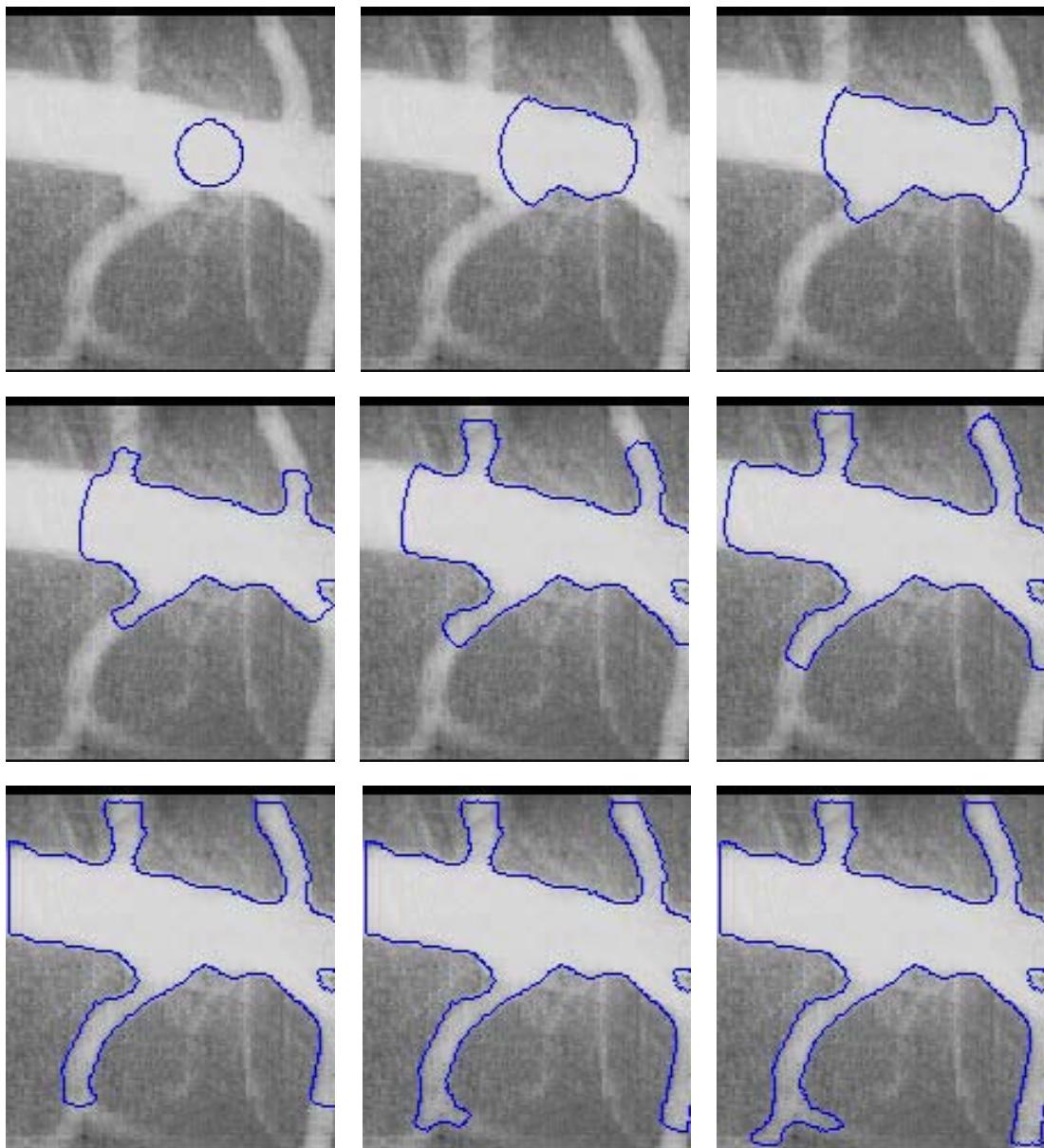


Figure B.3: Example channels from the space overlapping method. initializing a small circle inside the region of interest, and allowing it to grow outwards until it reaches the desired boundary. (Malladi *et al.*, 1995)

contrast, when the curve passes over places where the image gradient is large, the curve front slows down Figure B.3.

B.3 Alpha Shapes

“Alpha shapes” are a common method in computational geometry to extract cavities from proteins (Edelsbrunner *et al.*, 1995; Liang *et al.*, 1998; Edelsbrunner *et al.*, 1998). Alpha shapes are very similar to the Voronoi method employed in Chapter 2, but instead of defining a volume the alpha shapes are the voids in a Voronoi construction (Figure B.5). Alpha shapes rely on the accessible surface of the macromolecule defined by the rolling probe (see Figure 3.1, page 51) to define cavities and channels (Figure B.5, middle right). One drawback to this method is that the result is not a volume that touches the VDW surface of the atoms, but rather a collection of atom centers which define the cavity (Figure B.5, middle right). The convex hull issues may again be a problem, but it is not straightforward. Unfortunately, the two alpha-shapes software packages available do not support large structures like the ribosome.

B.4 Vector Stepping

“Vector stepping” is a method that uses a vector along the a particular channel and a point inside the channel to extract the entire channel. Vector stepping is most popular in membrane proteins using the program, HOLE (Smart *et al.*, 1993). HOLE is a program that is used to extract ion channels from membrane protein structures by stepping through them. The method then has only two steps, first calculate the cross-sectional area from the start point and second move along the vector to a new point (see Figure B.6A). Starting at the initial point, it calculates the surface area of the channel perpendicular to vector. There are a couple of ways to determine the cross-sectional area. The first method used by the program HOLE, extends a probe of a given radius out radially in several directions until each touches another atom. It then takes all the radial values to construct a polygon and determines the area. A second method to calculate the area is to slice the macromolecule in two-dimensions and then use a two-dimensional rolling probe technique to calculate the area

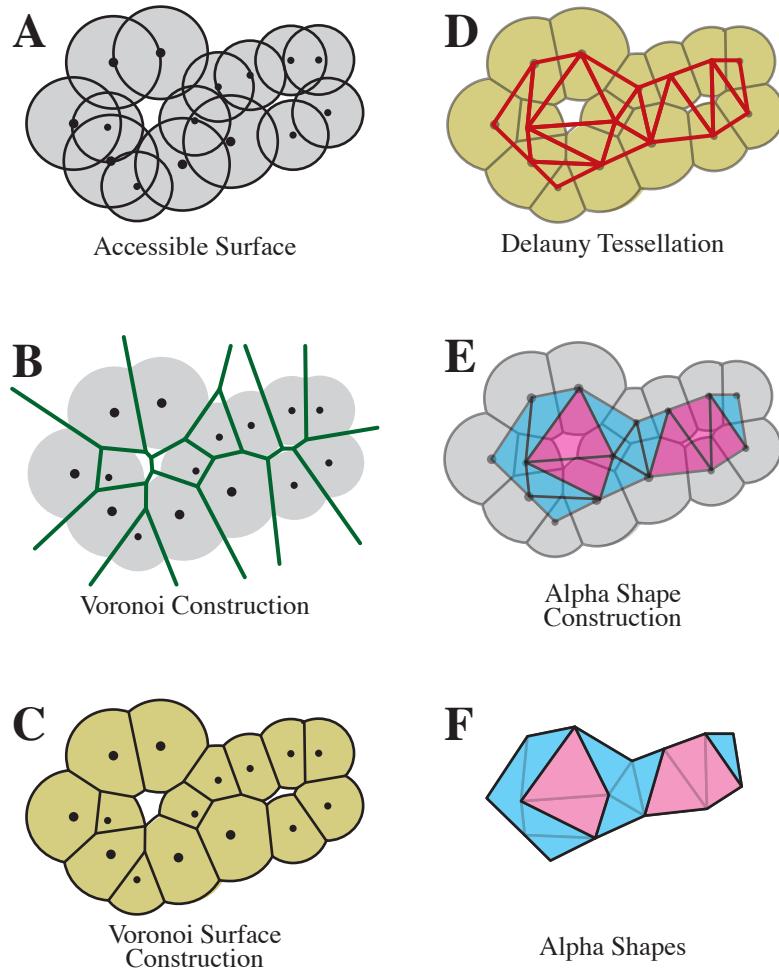


Figure B.4: Defining cavities and channels using alpha shapes. (A) The accessible surface defined by Lee & Richards (1971) (see Figure 3.1, page 51). (B) The Voronoi construction defined in Chapter 2 (see Figure 2.1, page 12). (C) By taking the accessible surface in part (A) and the Voronoi planes in part (B), the Voronoi surface can be constructed. Whenever the Voronoi plane is contained by the accessible surface, use the plane otherwise, use the accessible surface as the boundary. (D) The Delaunay tessellation is constructed by connecting a line between all atoms that share a Voronoi plane in part (B). (E) Alpha shapes are then constructed by taking the Delaunay tessellation and determining which shapes (triangles in this two-dimensional case) are contained within the accessible surface (blue) or those that contain a cavity (pink). (F) The resulting alpha shapes from this example molecule, it contains two cavities (pink).

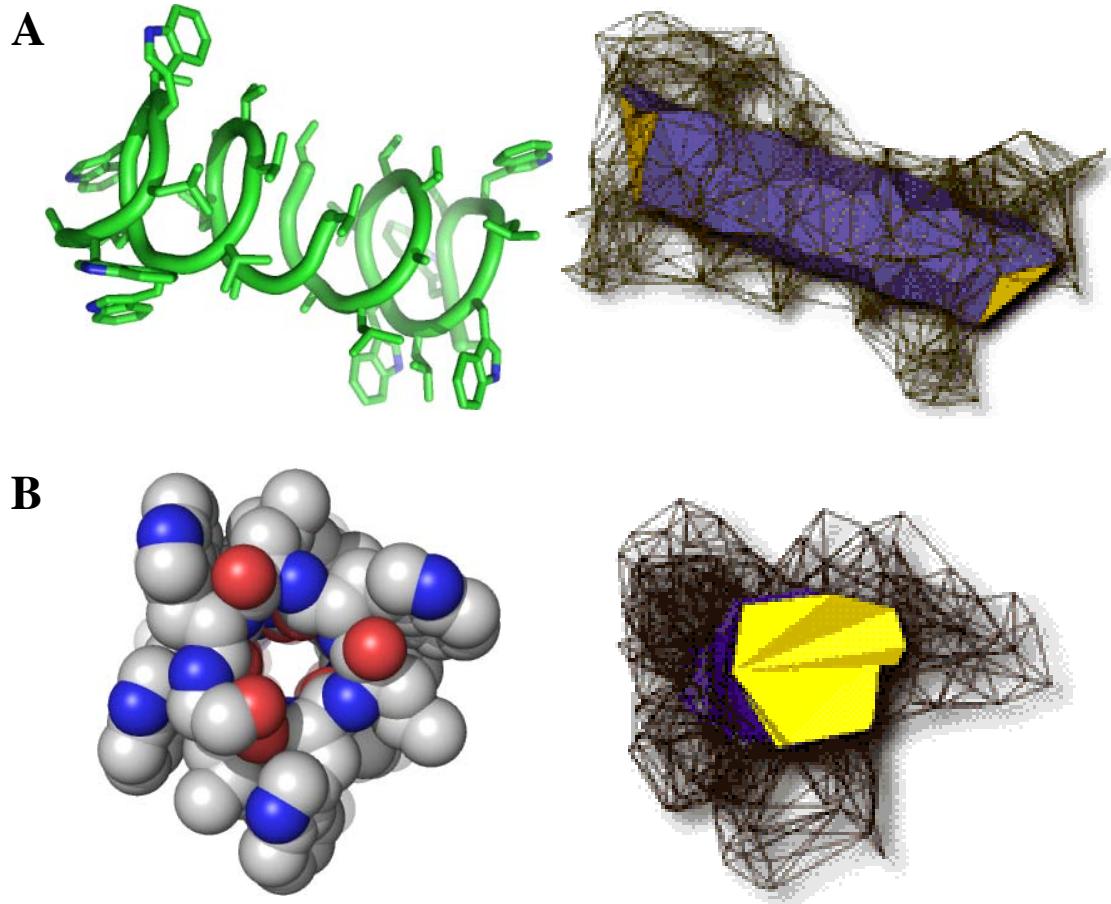


Figure B.5: Using alpha shapes to extract interior channel of gramicidin. (A) Side view of gramicidin secondary structure (left) and the extracted interior channel using alpha shapes (right). (B) Top view of the CPK sphere model of gramicidin showing the central channl (left) and the extracted alpha shape channel (right). Right images taken from Edelsbrunner *et al.* (1998), left images were generated from PDB code: 1GRM (Lomize *et al.*, 1992) using PyMOL (DeLano, 2002).

(Figure B.6C). After the area is calculated the program needs to step along the vector. The program can either move from the initial point or the center of the calculated area. This method is easy to visualize and understand, takes a long time to parse the entire tunnel and this method is plagued with problems for channels without high symmetry of membrane proteins. When this method was attempted on the tunnel, it consistently runs into dead ends and provide unrealistic cross-sectional areas (Figure B.6B). To work around this the program was designed to “wobble” at each point. That is the program will not only calculate the surface area perpendicular to the vector, but also do a tilt series (usually about 15° in 6 directions). The tilt series allows the program to “see ahead” and anticipate dead ends. Unfortunately, this was still not enough. Sometime the tunnel cross-section can look peanut-shaped and thus provided two different large passages for the program to step through (Figure B.6C). To prevent this, I switch to the other method of calculating the cross-sectional area, slice the molecule in two-dimensions and use a two-dimensional rolling probe. This performed better with some slight drawbacks if the probe was too small, it tended to enter side branches and sometimes reach the exterior. Finally, when a macromolecule was introduced the programs would dart down the wrong passage. At this point, I decided instead of using a two-dimensional rolling probe with several parameters and restraints it was best to use a three-dimensional rolling probe with fewer parameters and no restraints.

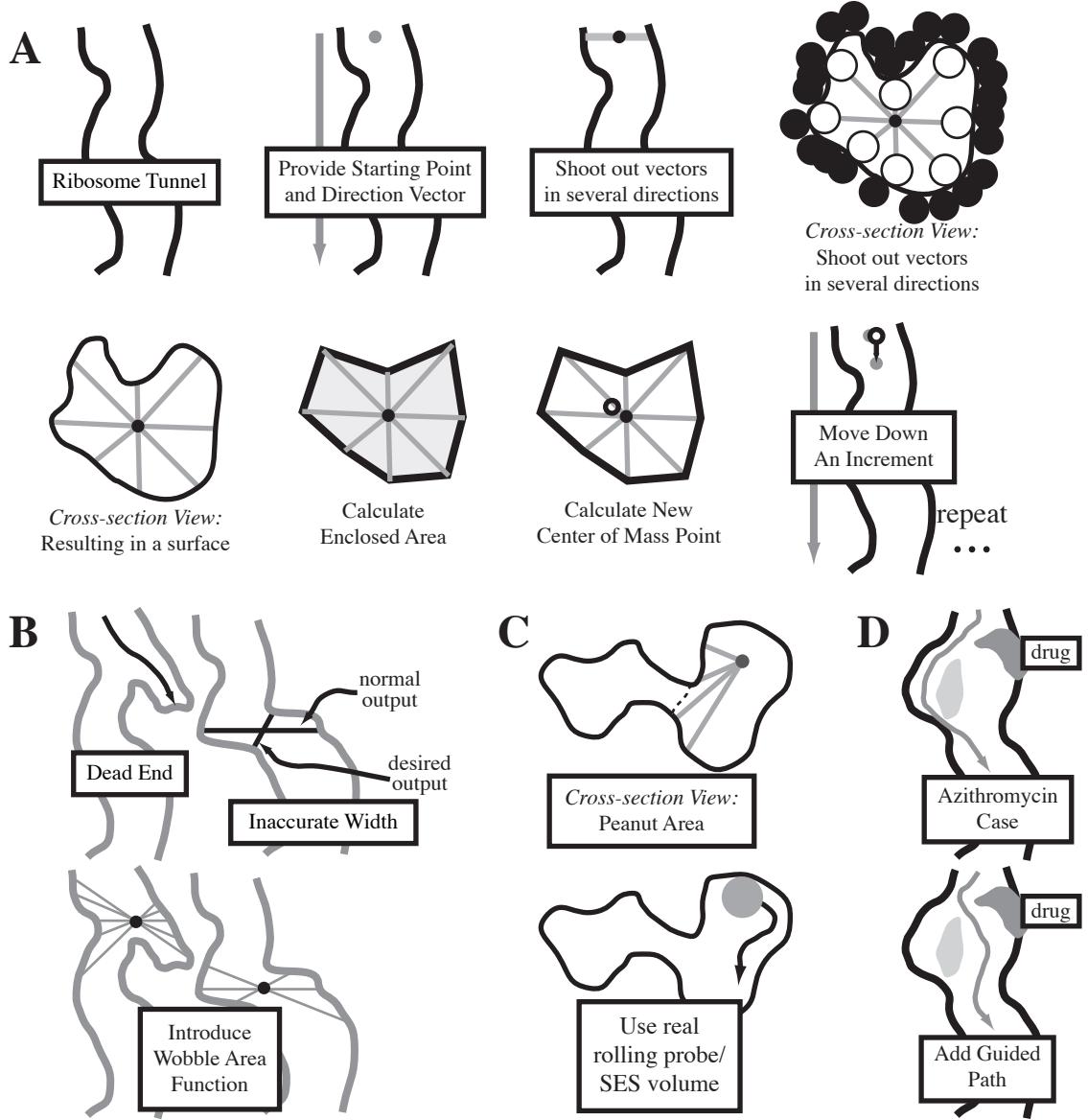


Figure B.6: Extracting the exit tunnel by vector stepping. (A) Given a central point and a vector, start at that point and step along a vector, calculate the cross-sectional area and obtain a new central point. (B) The method shown in part A suffers from running into dead ends and not always providing a realistic cross-sectional area of a narrow section of the exit tunnel. To prevent this the “wobble” function was introduced. The “wobble” function samples the cross-sectional area at various tilts which allows it to see a dead end coming and provide a more realistic cross-sectional area. (C) Despite fixing the dead end issue, another issue of non-circular cross-sectional areas still played a role in unrealistic areas getting calculated. To correct for this instead of shooting out vector at given angle the excluded surface was calculated in two-dimensions. (D) Finally, after fixing the previous two problems, the algorithm had a tendency to follow the smaller path when a peanut-shaped area (shown in part C) divides the area into two separate paths. To correct this last problem a guided path was introduced into the program based on previous runs.

B.5 References

- D. Bakowies and W. F. van Gunsteren (2002). “Water in protein cavities: A procedure to identify internal water and exchange pathways and application to fatty acid-binding protein”. *Proteins-Structure Function And Genetics*, **47** (4): pp. 534–545.
- W. L. DeLano (2002). “The PyMOL molecular graphics system”. URL <http://www.pymol.org>.
- H. Edelsbrunner, M. Facello, P. Fu, and J. Liang (1995). “Measuring proteins and voids in proteins.” In “Proc. 28th Ann. Hawaii Internat. Conf. System Sci.”, pp. 256–264.
- H. Edelsbrunner, M. A. Facello, and J. Liang. (1998). “On the definition and the construction of pockets in macromolecules.” *Discrete Appl Math*, **88**: pp. 83–102.
- G. J. Kleywegt and T. A. Jones (1994). “Detection, delineation, measurement and display of cavities in macromolecular structures”. *Acta Cryst D*, **50**: pp. 178–185.
- G. J. Kleywegt, J. Y. Zou, M. Kjeldgaard, and T. A. Jones (2001). “Around O”. In M. G. Rossman and E. Arnold, eds., “International tables for crystallography”, volume F: Crystallography of biological macromolecules, pp. 353–356, 366–367. Kluwer Academic Publishers, The Netherlands.
- R. A. Laskowski (1995). “SURFNET: A program for visualizing molecular-surfaces, cavities, and intermolecular interactions”. *Journal of Molecular Graphics*, **13** (5): pp. 323–330.
- R. A. Laskowski, N. M. Luscombe, M. B. Swindells, and J. M. Thornton (1996). “Protein clefts in molecular recognition and function”. *Protein Science*, **5** (12): pp. 2438–2452.
- B. Lee and F. M. Richards (1971). “The interpretation of protein structures: Estimation of static accessibility”. *J Mol Biol*, **55** (3): pp. 379–400. [http://dx.doi.org/10.1016/0022-2836\(71\)90324-X](http://dx.doi.org/10.1016/0022-2836(71)90324-X).
- J. Liang, H. Edelsbrunner, and C. Woodward (1998). “Anatomy of protein pockets and cavities: Measurement of binding site geometry and implications for ligand design”. *Protein Science*, **7** (9): pp. 1884–1897.
- A. L. Lomize, V. I. Orekhov, and A. S. Arsen’ev (1992). “[Refinement of the spatial structure of the gramicidin A ion channel]”. *Bioorg Khim*, **18** (2): pp. 182–200.
- R. Malladi, J. Sethian, and B. Vemuri (1995). “Shape modeling with front propagation: A level set approach”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17** (2): pp. 158–175. URL http://math.berkeley.edu/~sethian/2006/Applications/Medical_Imaging/artery.html.
- S. J. Osher and R. P. Fedkiw (1988). “Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton–Jacobi Formulations”. *J Comp Phys*, **79**: pp. 12–49.
- O. S. Smart, J. M. Goodfellow, and B. A. Wallace (1993). “The pore dimensions of gramicidin A”. *Biophys J*, **65** (6): pp. 2455–60.

Appendix C

Additional Ribosomal Exit Tunnels from Other Structures

C.1 Available structures

At the time of submission, Voss *et al.* (2006) had only one complete structure of the ribosome available, PDB code 1JJ2 (Klein *et al.*, 2001). There were two other models of the large subunit, PDB codes 1GIY from *T. thermophilus* (Yusupov *et al.*, 2001) and 1NWK from *D. radiodurans* (Harms *et al.*, 2001), which were not complete structures. 1GIY was low resolution (5.5 Å) and contains only phosphorus (P) and alpha-carbon (CA) locations. 1NWK was more complete than 1GIY, but still did not contain any protein side-chain information, only alpha-carbon (CA) locations. Since the tunnel wall is 82% RNA it was still possible to get a model of the exit tunnel for 1NWK, but it could not be trusted for any useful information about the properties of the exit tunnel.

Since the submission of Voss *et al.* (2006), four more complete structures from two organisms (*T. thermophilus* and *E. coli*) have been independently published on the large subunit of the ribosome. Also, the *E. coli* structure contained two different ribosomes in the crystal.

C.2 Volumes of the Exit Tunnels

Figure C.1 shows the volumes of the exit tunnel as a function of probe radius. It is clear that the 1JJ2 structure has the most compact volume of all the exit tunnels. The appearance of the different exit

Organism	PDB code(s)	Date	resol'n	R_{free}	# atoms	info	Reference
<i>H. marismortui</i>	1FFK	07/25/2000	2.4	26.1	64,268	CA only	Ban <i>et al.</i> (2000)
<i>H. marismortui</i>	1JJ2	07/03/2001	2.4	22.2	90,418	full	Klein <i>et al.</i> (2001)
<i>T. thermophilus</i>	1GIY	03/30/2001	5.5	—	5,747	P, CA only	Yusupov <i>et al.</i> (2001)
<i>D. radiodurans</i>	1NKW	01/05/2003	3.1	27.4	65,300	CA only	Harms <i>et al.</i> (2001)
<i>H. marismortui</i>	1S72	01/28/2004	2.4	22.2	90,985	extra	Klein <i>et al.</i> (2004)
<i>T. thermophilus</i>	1YL3	01/19/2005	5.5	35.6	92,097	full	Jenner <i>et al.</i> (2005)
<i>E. coli</i>	2AW4,2AWB	08/31/2005	3.5	33.1	89,690	full	Schuwirth <i>et al.</i> (2005)
<i>T. thermophilus</i>	2J01,2J03	07/31/2006	2.8	31.3	89,408	full	Selmer <i>et al.</i> (2006)
<i>T. thermophilus</i>	1VS9	08/14/2006	3.7	34.7	91,299	full	Korostelev <i>et al.</i> (2006)

Table C.1: All 50S subunit structures available and their characteristics. Date is the date the coordinates were deposited in the PDB. The number of atoms (# atoms) refers to the number of RNA and protein atoms. Solvent atoms are not included. For the completeness column: “CA only” means that there is no side chain information for proteins and only alpha-carbon coordinates. “P, CA only” means there are only phosphorus and alpha-carbon coordinates in the file. “full” means that all side chain information is included. “extra” means the file contains all atom information and also base modifications.

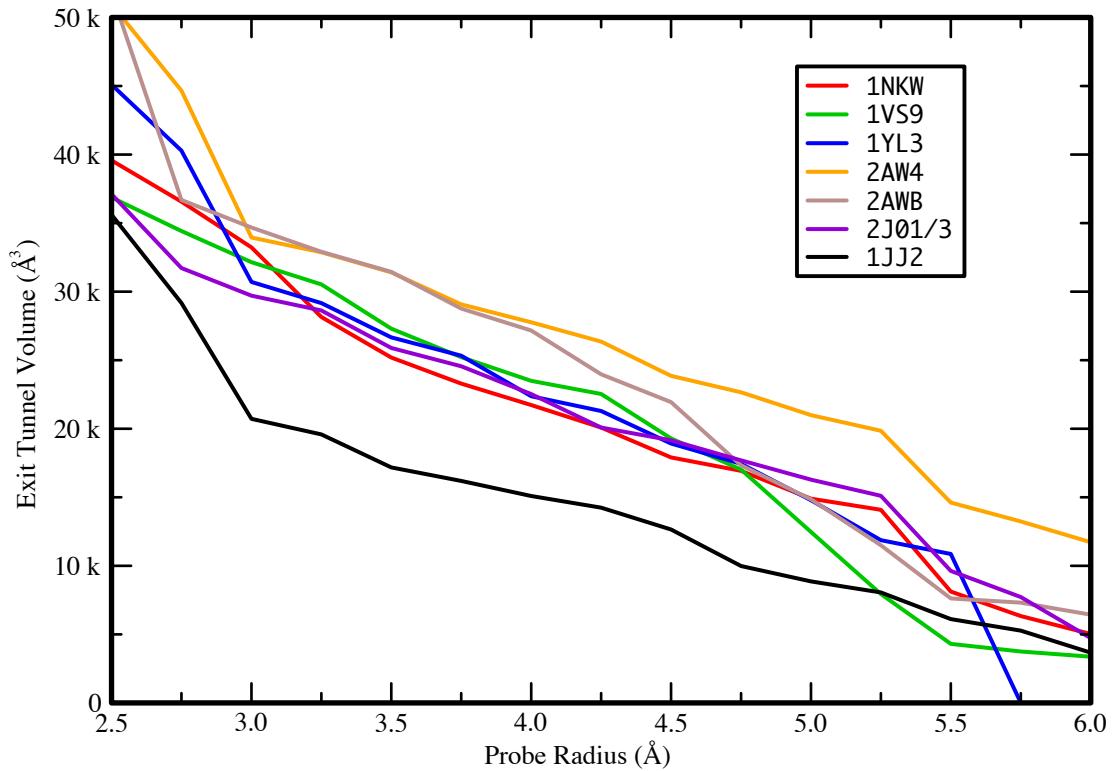


Figure C.1: Volume as a function of probe radius for the different exit tunnels. PDB files 2J01 and 2J03 produced the exact same results, so only 2J01 is shown. PDB files 1JJ2 and 1S72 produced the very similar results, so only 1JJ2 is shown. PDB file 1NKW is the only structure included without protein side chain information. Structures from PDB files 1FFK and 1GIY are not included. PDB codes correspond to the structures listed in Table C.1.

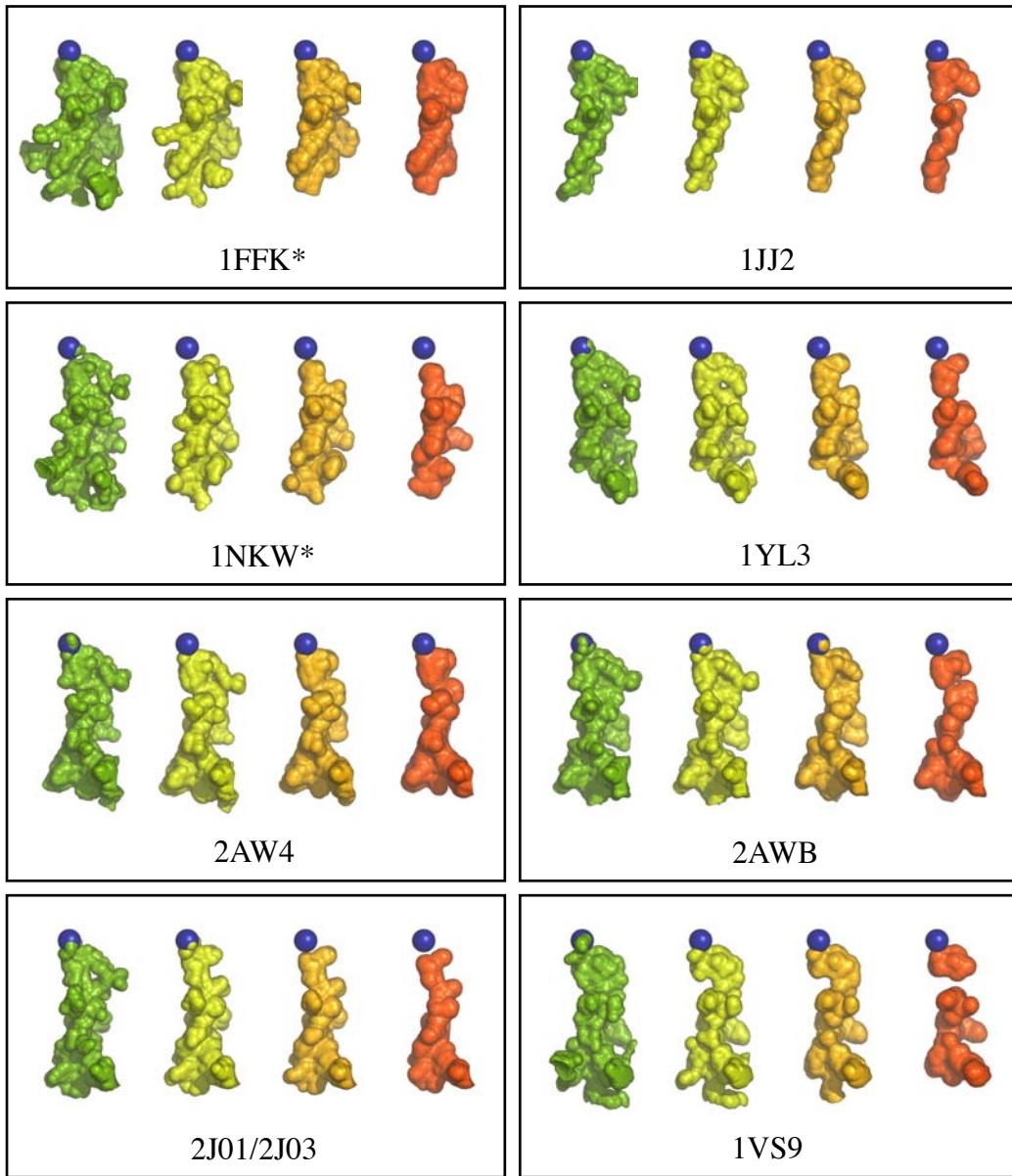


Figure C.2: Images of the different exit tunnels. The probe radius in the images increases from left to right: 3.0 Å (green), 3.5 Å (yellow), 4.0 Å (orange), and 4.5 Å (red). PDB files 2J01 and 2J03 produced the exact same results, so only 2J01 is shown. PDB files 1JJ2 and 1S72 produced the very similar results, so only 1JJ2 is shown. PDB files 1FFK and 1NWK are the only structures included without protein side chain information and are marked with an asterisk (*). The structures from PDB file 1GIY fill the entire area and are not included. All images are drawn to scale. PDB codes correspond to the structures listed in Table C.1.

tunnels is dramatically different as well (Figure C.2). All the tunnel possess the narrow cylindrical structure, but do so in different ways. Several of the tunnel still possess branches at 3.0 Å as well.

C.3 Rotations

PDB files 1FFK, 1JJ2, and 1S72 of the *H. marismortui* large subunit all have the same rotation. All other structures were rotated to the *H. marismortui* coordinates using the program LSQMAN (Kleywegt & Jones, 1993) resulting in the following transformation matrices:

$$\mathbf{PDB}_{IJJ2} = \begin{bmatrix} -0.149430 & -0.152442 & -0.976950 \\ -0.986960 & 0.082787 & 0.138043 \\ 0.059835 & 0.984839 & -0.162825 \end{bmatrix} \cdot \left(\mathbf{PDB}_{IGIY} + \begin{bmatrix} -227.174 \\ 196.088 \\ -156.483 \end{bmatrix} \right) \quad (\text{C.1})$$

$$\mathbf{PDB}_{IJJ2} = \begin{bmatrix} 0.631943 & -0.463245 & 0.621331 \\ 0.681478 & -0.049673 & -0.730151 \\ 0.369102 & 0.884837 & 0.284301 \end{bmatrix} \cdot \left(\mathbf{PDB}_{INKW} + \begin{bmatrix} 83.776 \\ -50.659 \\ -122.938 \end{bmatrix} \right) \quad (\text{C.2})$$

$$\mathbf{PDB}_{IJJ2} = \begin{bmatrix} 0.987881 & -0.109579 & -0.109927 \\ -0.118820 & -0.078214 & -0.989831 \\ 0.099867 & 0.990896 & -0.090286 \end{bmatrix} \cdot \left(\mathbf{PDB}_{IVS9} + \begin{bmatrix} -254.304 \\ -193.655 \\ -97.725 \end{bmatrix} \right) \quad (\text{C.3})$$

$$\mathbf{PDB}_{IJJ2} = \begin{bmatrix} -0.132479 & -0.154505 & -0.979070 \\ 0.988572 & -0.092288 & -0.119201 \\ -0.071939 & -0.983672 & 0.164965 \end{bmatrix} \cdot \left(\mathbf{PDB}_{IYL3} + \begin{bmatrix} -34.866 \\ -260.889 \\ -373.948 \end{bmatrix} \right) \quad (\text{C.4})$$

$$\mathbf{PDB}_{IJJ2} = \begin{bmatrix} -0.299957 & -0.850685 & 0.431695 \\ 0.773061 & -0.481907 & -0.412483 \\ 0.558930 & 0.209999 & 0.802183 \end{bmatrix} \cdot \left(\mathbf{PDB}_{2AW4} + \begin{bmatrix} -185.248 \\ -195.616 \\ -117.168 \end{bmatrix} \right) \quad (\text{C.5})$$

$$\mathbf{PDB}_{IJJ2} = \begin{bmatrix} -0.217730 & -0.938801 & 0.266920 \\ -0.469109 & -0.139161 & -0.872108 \\ 0.855880 & -0.315098 & -0.410100 \end{bmatrix} \cdot \left(\mathbf{PDB}_{2AWB} + \begin{bmatrix} -285.869 \\ -124.399 \\ -187.053 \end{bmatrix} \right) \quad (\text{C.6})$$

$$\mathbf{PDB}_{IJJ2} = \begin{bmatrix} 0.169808 & 0.927499 & -0.333033 \\ -0.036609 & -0.331771 & -0.942649 \\ -0.984797 & 0.172261 & -0.022382 \end{bmatrix} \cdot \left(\mathbf{PDB}_{2J01} + \begin{bmatrix} -246.467 \\ -159.089 \\ -208.565 \end{bmatrix} \right) \quad (\text{C.7})$$

$$\mathbf{PDB}_{IJJ2} = \begin{bmatrix} 0.177173 & 0.927832 & -0.328234 \\ 0.984179 & -0.167565 & 0.057575 \\ -0.001581 & -0.333241 & -0.942840 \end{bmatrix} \cdot \left(\mathbf{PDB}_{2J03} + \begin{bmatrix} -141.251 \\ -191.014 \\ -82.947 \end{bmatrix} \right) \quad (\text{C.8})$$

Using these rotation matrices RMSD values based on the phosphate and backbone 4' carbons are shown in Table C.2. These values are significantly smaller than those reported by Selmer *et al.* (2006), presumably because I was less stringent on the sequence alignment of the particles and allowed several gaps in the structure.

Source:	<i>H. ma</i>			<i>D. ra</i>	<i>T. th</i>			<i>T. th</i>		<i>E. co</i>	
	1FFK	1JJ2	1S72		1NKW	1GIY	1VS9	1YL3	2J01	2J03	2AW4
1FFK	.	0.058	0.060	0.572	0.682	0.659	0.557	0.512	0.510	0.545	0.540
1JJ2	0.058	.	0.015	0.572	0.685	0.659	0.559	0.514	0.513	0.545	0.537
1S72	0.060	0.015	.	0.573	0.761	0.662	0.559	0.513	0.512	0.546	0.541
1NKW	0.572	0.572	0.573	.	0.718	0.590	0.478	0.717	0.716	0.799	0.800
1GIY	0.682	0.685	0.759	0.718	.	0.748	0.651	0.649	0.649	0.692	0.700
1VS9	0.659	0.659	0.662	0.590	0.747	.	0.553	0.539	0.537	0.595	0.595
1YL3	0.557	0.560	0.559	0.478	0.651	0.553	.	0.525	0.525	0.913	0.910
2J01	0.512	0.514	0.513	0.717	0.649	0.539	0.525	.	0.032	0.486	0.574
2J03	0.510	0.512	0.512	0.716	0.649	0.537	0.525	0.032	.	0.487	0.575
2AW4	0.545	0.546	0.546	0.799	0.692	0.595	0.913	0.486	0.487	.	0.106
2AWB	0.540	0.537	0.541	0.800	0.700	0.595	0.910	0.574	0.575	0.106	.

Table C.2: RMSD differences between the PDB files. RMSD values are based upon the phosphate (P) and 4' carbon of the sugar backbone (C4*). PDB files are grouped by organism (listed at the top) and crystal form. PDB codes correspond to the structures listed in Table C.1.

C.4 References

- N. Ban, P. Nissen, J. Hansen, P. B. Moore, and T. A. Steitz (2000). “The complete atomic structure of the large ribosomal subunit at 2.4 Å resolution”. *Science*, **289** (5481): pp. 905–20.
- J. Harms, F. Schluelzen, R. Zarivach, A. Bashan, S. Gat, I. Agmon, H. Bartels, F. Franceschi, and A. Yonath (2001). “High resolution structure of the large ribosomal subunit from a mesophilic eubacterium”. *Cell*, **107** (5): pp. 679–88.
- L. Jenner, P. Romby, B. Rees, C. Schulze-Briese, M. Springer, C. Ehresmann, B. Ehresmann, D. Moras, G. Yusupova, and M. Yusupov (2005). “Translational operator of mRNA on the ribosome: how repressor proteins exclude ribosome binding”. *Science*, **308** (5718): pp. 120–123.
- D. J. Klein, P. B. Moore, and T. A. Steitz (2004). “The roles of ribosomal proteins in the structure assembly, and evolution of the large ribosomal subunit”. *J Mol Biol*, **340** (1): pp. 141–77.
- D. J. Klein, T. M. Schmeing, P. B. Moore, and T. A. Steitz (2001). “The kink-turn: A new RNA secondary structure motif”. *Embo J*, **20** (15): pp. 4214–21. <http://dx.doi.org/10.1093/emboj/20.15.4214>.
- G. J. Kleywegt and T. A. Jones (1993). “LSQMAN”. URL <http://xray.bmc.uu.se/usf/rave.html>.
- A. Korostelev, S. Trakhanov, M. Laurberg, and H. F. Noller (2006). “Crystal structure of a 70S ribosome-tRNA complex reveals functional interactions and rearrangements”. *Cell*, **126** (6): pp. 1065–1077.
- B. S. Schuwirth, M. A. Borovinskaya, C. W. Hau, W. Zhang, A. Vila-Sanjurjo, J. M. Holton, and J. H. D. Cate (2005). “Structures of the bacterial ribosome at 3.5 Å resolution”. *Science*, **310** (5749): pp. 827–834.
- M. Selmer, C. M. Dunham, F. V. t. Murphy, A. Weixlbaumer, S. Petry, A. C. Kelley, J. R. Weir, and V. Ramakrishnan (2006). “Structure of the 70S ribosome complexed with mRNA and tRNA”. *Science*, **313** (5795): pp. 1935–1942.
- N. R. Voss, M. Gerstein, T. A. Steitz, and P. B. Moore (2006). “The geometry of the ribosomal polypeptide exit tunnel”. *J Mol Biol*, **360** (4): pp. 893–906. <http://dx.doi.org/10.1016/j.jmb.2006.05.023>.
- M. M. Yusupov, G. Z. Yusupova, A. Baucom, K. Lieberman, T. N. Earnest, J. H. Cate, and H. F. Noller (2001). “Crystal structure of the ribosome at 5.5 Å resolution”. *Science*, **292** (5518): pp. 883–96.

Appendix D

Data Files

D.1 Data Files for the Voronoi Programs

Two figures D.1 and D.2 contain the distribution for all 17 atom types using both method B and the radical plane method.

Also there are seven data files associated with the volume set of which three are needed to run the Voronoi program.

- atom-defs.dat: definitions relating the 49 kinds of RNA atoms to their respective atom types.
- atom-vols-prot.dat: Atomic volumes for proteins.
- atom-vols-rna.dat: Atomic volumes for RNA.
- bond-lengths.dat: Bond lengths data file that was used to calculate the bonded radii for the atom types.
- resi-vols.dat: residue volumes for RNA bases and nucleotides and protein amino acids.
- stdvol258.dat: the volume of all 258 kinds of atoms for both protein and RNA.
- stdvol-extended.dat: the same as the ‘stdvol258.dat’ file with extended information such as the median, mode, and skew of the distributions.

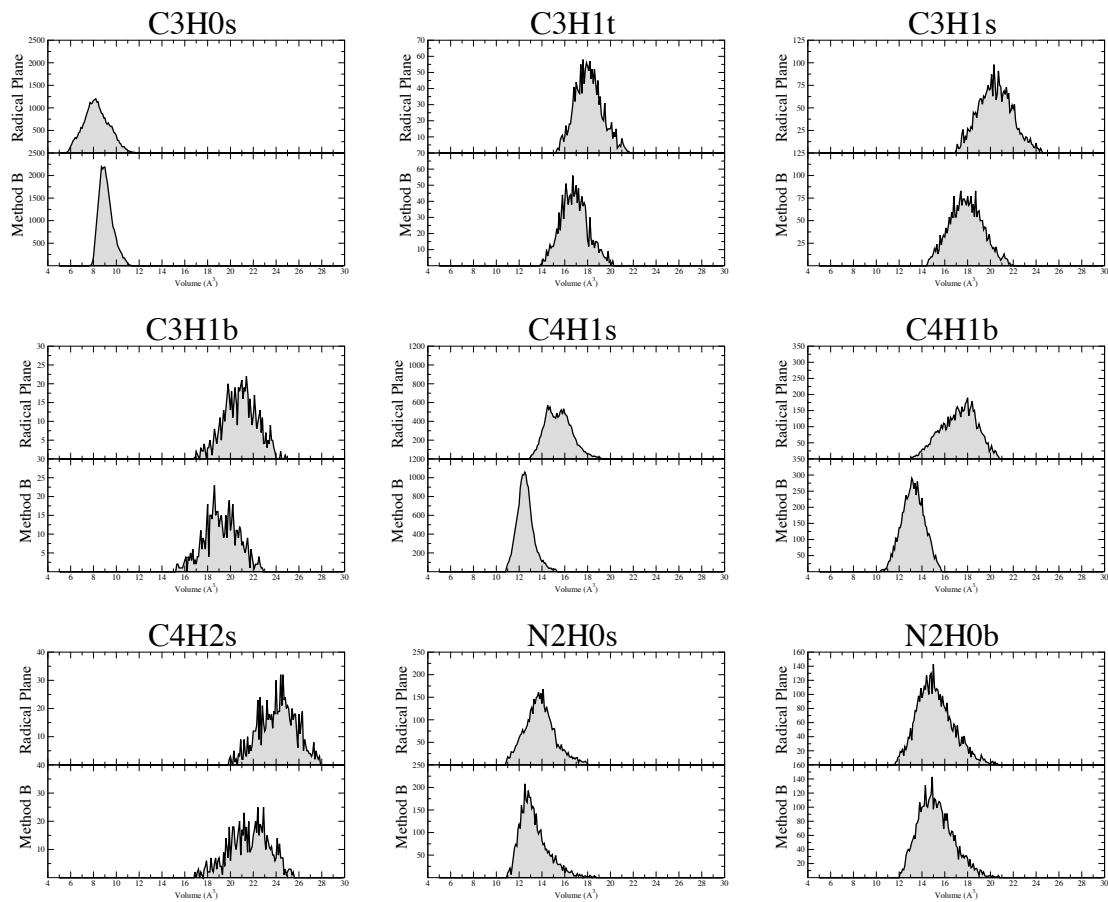


Figure D.1: Distribution of atom type volumes for both plane positioning methods, part 1.
The atom type is listed above each graph. The distribution of volumes from radical plane method is on top and distribution of volumes from method B is on bottom. All graphs have the same x-axis range from 4–30 Å³. The y-axis varies for each graph, but is consistent for both methods.

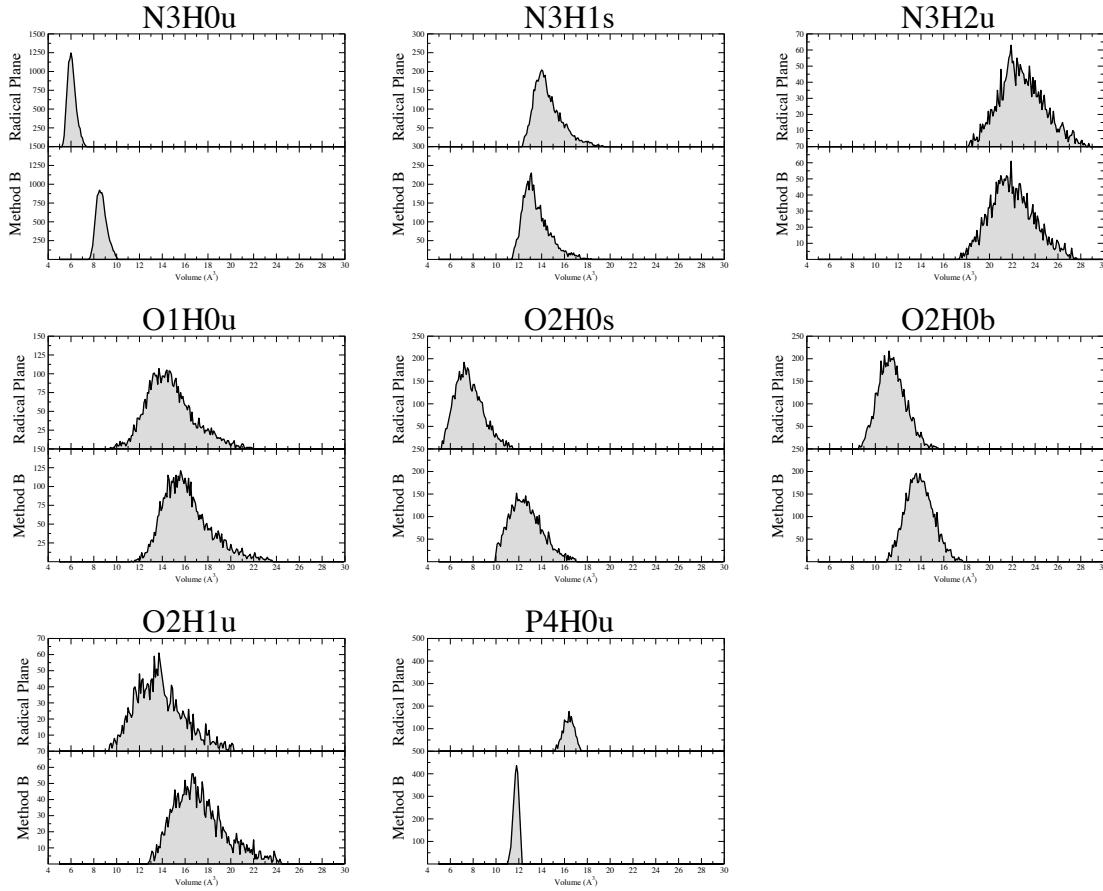


Figure D.2: Distribution of atom type volumes for both plane positioning methods, part 2.
The atom type is listed above each graph. The distribution of volumes from radical plane method is on top and distribution of volumes from method B is on bottom. All graphs have the same x-axis range from 4–30 Å³. The y-axis varies for each graph, but is consistent for both methods.

C3H0s	1.61	0.72
C3H0b	1.61	0.72
C3H1t	1.76	0.68
C3H1s	1.76	0.68
C3H1b	1.76	0.68
C3H2u	1.76	0.77 #not refined
C4H1s	1.88	0.78
C4H1b	1.88	0.78
C4H2s	1.88	0.76
C4H2b	1.88	0.76
C4H3u	1.88	0.74
N2H0s	1.64	0.64
N2H0b	1.64	0.64
N3H0u	1.64	0.69
N3H1s	1.64	0.67
N3H1b	1.64	0.67
N3H2u	1.64	0.61
N4H3u	1.64	0.73
O1H0u	1.42	0.52
O2H0s	1.50	0.65
O2H0b	1.62	0.65
O2H1u	1.46	0.65
O2H2u	1.46	0.66 #not refined
S2H0u	1.77	1.04
S2H1u	1.77	1.04
P4H0u	1.82	0.97
O1N3HH	1.60	0.68 #not refined
Z2ION	0.74	0.74 #not refined
FE	1.70	0.70 #not refined
MGION	1.59	0.70 #not refined
DEFAULT	1.70	0.77 #not refined

group	radii	volume	sd	num
-----	-----	-----	----	---
C3H0s	1.61	8.72	0.58	20
C3H0b	1.61	9.70	0.72	13
C3H1s	1.76	20.44	1.74	8
C3H1b	1.76	21.28	1.85	8
C4H1s	1.88	13.71	0.96	18
C4H1b	1.88	14.35	1.30	6
C4H2s	1.88	23.19	2.28	20
C4H2b	1.88	24.26	2.06	7
C4H3u	1.88	36.73	3.24	9
N3H0u	1.64	8.65	0.59	1
N3H1b	1.64	15.72	1.51	4
N3H1s	1.64	13.62	0.99	20
N3H2u	1.64	22.69	2.15	4
N4H3u	1.64	21.41	1.15	1
O1H0u	1.42	15.91	1.29	27
O2H1u	1.46	17.98	1.73	3
S2H0u	1.77	29.17	2.65	2
S2H1u	1.77	36.75	4.16	1

```

# NucProt type set of Atom Volumes
#
# Generated: Mon Sep 27 17:29:34 2004
# Input file simplify.out
# v 3.2 2/26/04
# Output file atomvol-simp.dat
# Atom file atom-defs.dat
# Type 'rna'
#

```

type	count	%OK	rad	median	mode	mean	sd	10%ile	90%ile	skew	kurt
C3H0s	7592	91.9%	1.61	9.14	8.97	9.231	0.58	8.59	9.98	0.99	1.80
C3H1b	319	25.4%	1.76	19.57	18.70	19.598	1.49	17.80	21.57	0.19	0.30
C3H1s	1122	47.5%	1.76	18.26	17.44	18.274	1.51	16.37	20.19	0.22	0.29
C3H1t	568	45.3%	1.76	17.09	16.53	17.216	1.21	15.79	18.88	0.48	0.42
C4H1b	2680	46.6%	1.88	13.44	13.25	13.471	0.94	12.32	14.70	0.07	0.22
C4H1s	4478	77.9%	1.88	12.72	12.64	12.822	0.75	12.03	13.74	1.04	2.75
C4H2s	445	15.5%	1.88	22.01	22.68	21.956	1.81	19.79	24.23	-0.18	0.27
N2H0b	1818	56.0%	1.64	15.13	14.98	15.276	1.48	13.55	17.11	0.65	1.04
N2H0s	1198	81.4%	1.64	13.31	12.59	13.594	1.22	12.37	15.27	1.45	3.55
N3H0u	2670	92.8%	1.64	8.75	8.85	8.780	0.41	8.30	9.32	0.48	0.45
N3H1s	1222	87.0%	1.64	13.43	13.16	13.688	1.07	12.60	15.07	1.39	2.72
N3H2u	1077	45.8%	1.64	22.09	21.55	22.214	1.86	20.01	24.61	0.37	0.27
O1H0u	2604	31.0%	1.42	15.80	15.48	16.085	2.05	13.88	18.63	1.00	2.13
O2H0b	2344	40.8%	1.62	13.89	13.96	13.980	1.22	12.51	15.48	0.47	0.65
O2H0s	1762	61.3%	1.50	12.64	12.83	12.749	1.31	11.22	14.42	0.68	0.94
O2H1u	1204	41.9%	1.46	16.86	16.74	17.103	1.92	14.92	19.48	0.85	1.22
P4H0u	1079	37.6%	1.82	11.85	11.79	11.837	0.22	11.53	12.12	-0.41	0.17

! Phos. - combined RNA/DNA statistics used

BOND	P4H0u	O1H0u	1400.000	1.485
BOND	P4H0u	O1H0u	1400.000	1.485
BOND	P4H0u	O2H0b	220.000	1.593
BOND	P4H0u	O2H0b	220.000	1.607
! RNA Sugars				
BOND	C4H2S	C4H2S	580.000	1.425
BOND	C4H1S	C4H1S	540.000	1.510
BOND	C4H1S	C4H1S	540.000	1.524
BOND	C4H1S	C4H1S	540.000	1.525
BOND	C4H1S	C4H1S	540.000	1.528
BOND	C4H1S	C4H1S	540.000	1.528
BOND	O2H0b	C4H1S	580.000	1.414
BOND	O2H0b	C4H1S	580.000	1.453
BOND	C4H1S	C4H1S	580.000	1.423
BOND	C4H1S	O2H1lu	580.000	1.413
! Base-sugar joint BONDS				
BOND	C4H1S	N3H0u	648.000	1.469
BOND	C4H1S	N3H0u	648.000	1.470
BOND	C4H1S	N3H0u	648.000	1.459
BOND	C4H1S	N3H0u	648.000	1.462
! cytosine				
BOND	C3H0S	O1H0u	1580.000	1.240
BOND	C3H0S	N3H2u	1224.000	1.335
BOND	C3H0S	C3H0S	1044.000	1.397
BOND	N3H0u	C3H1t	1032.000	1.367
BOND	C3H0S	N2H0s	1128.000	1.353
BOND	N2H0s	C3H0S	1232.000	1.335
BOND	C3H0S	C3H1s	1008.000	1.425
BOND	C3H0S	C3H1t	1496.000	1.339
! adenine				
BOND	N2H0s	C3H1t	1308.000	1.339
BOND	C3H1t	N2H0s	1308.000	1.331
BOND	N2H0b	C3H0S	1232.000	1.344
BOND	C3H0S	C3H0S	1380.000	1.383
BOND	C3H0S	C3H0S	1176.000	1.406
BOND	C3H0S	N2H0s	1224.000	1.351
BOND	C3H0S	N2H0b	956.000	1.388
BOND	N2H0b	C3H1s	1416.000	1.311
BOND	C3H1s	N3H0u	1030.000	1.373
BOND	N3H0u	C3H0S	1044.000	1.374
BOND	C3H0S	N3H2u	1224.000	1.335
! guanine				
BOND	C3H1t	N3H1t	1008.000	1.373
BOND	C3H1t	N2H0b	1222.000	1.323
BOND	N2H0b	C3H0S	1144.000	1.350
BOND	C3H0S	C3H0S	1380.000	1.379
BOND	C3H0S	C3H0S	1088.000	1.419
BOND	C3H0S	N3H1s	972.000	1.391
BOND	C3H0S	N2H0b	936.000	1.388
BOND	N2H0b	C3H1s	1416.000	1.305
BOND	C3H1s	N3H0u	1000.000	1.374
BOND	C3H0S	C3H2u	1044.000	1.375
BOND	C3H0S	O1H0u	1224.000	1.341
BOND	C3H0S	O1H0u	1580.000	1.237
! uracil				
BOND	C3H0S	O1H0u	1580.000	1.219
BOND	C3H0S	O1H0u	1580.000	1.232
BOND	N3H0u	C3H0S	972.000	1.381
BOND	N3H0u	C3H1t	1082.000	1.375
BOND	C3H0S	N3H1s	972.000	1.373
BOND	N3H1s	C3H0S	972.000	1.380
BOND	C3H0S	C3H1s	940.000	1.431
BOND	C3H1s	C3H1t	1496.000	1.337
! proteins				
! ala				
BOND	N3H1s	C4H1b	1639.889	1.458
BOND	C4H1b	C3H0S	1342.404	1.525
BOND	C3H0S	O1H0u	1480.000	1.231

BOND	C4H1b	C4H3u	543.618	1.521
! arg				
BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525
BOND	C3H0s	O1H0u	1480.000	1.231
BOND	C4H1s	C4H2s	1480.000	1.530
BOND	C4H2s	C4H1s	1480.000	1.520
BOND	C4H2s	C4H2s	657.778	1.520
BOND	C4H2s	C4H1s	657.778	1.520
BOND	N3H1b	C3H0b	1827.161	1.460
BOND	C3H0b	O1H0u	3020.408	1.329
BOND	C3H0b	N3H2u	1827.161	1.326
BOND	C3H0b	N3H2u	1827.161	1.326
! asn				
BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525
BOND	C3H0s	O1H0u	1480.000	1.231
BOND	C4H1s	C4H2s	1480.000	1.530
BOND	C4H2s	C4H1s	1480.000	1.516
BOND	C3H0b	O1H0u	1480.000	1.231
BOND	C3H0b	N3H2u	1342.404	1.328
! asp				
BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525
BOND	C3H0s	O1H0u	1480.000	1.231
BOND	C4H1s	C4H2s	1480.000	1.530
BOND	C4H2s	C4H1s	1480.000	1.516
BOND	C3H0b	O1H0u	1639.889	1.249
BOND	C3H0b	O1H0u	1639.889	1.249
! cys				
BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525
BOND	C3H0s	O1H0u	1480.000	1.231
BOND	C4H1s	C4H2s	1480.000	1.530
BOND	C4H2s	C4H1s	1480.000	1.516
BOND	S2H1u	C4H1s	543.618	1.808
! gln				
BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525
BOND	C3H0s	O1H0u	1480.000	1.231
BOND	C4H1s	C4H2s	1480.000	1.530
BOND	C4H2s	C4H1s	1480.000	1.516
BOND	C3H0b	O1H0u	1480.000	1.231
BOND	C3H0b	N3H2u	1342.404	1.328
! glu				
BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525
BOND	C3H0s	O1H0u	1480.000	1.231
BOND	C4H1s	C4H2s	1480.000	1.530
BOND	C4H2s	C4H1s	1480.000	1.516
BOND	S2H1u	C4H1s	543.618	1.808
! his				
BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525
BOND	C3H0s	O1H0u	1480.000	1.231
BOND	C4H1s	C4H2s	1480.000	1.530
BOND	C4H2s	C4H1s	1480.000	1.516
BOND	C3H0b	O1H0u	1639.889	1.249
BOND	C3H0b	O1H0u	1639.889	1.249
! gly				
BOND	N3H1s	C4H2s	2312.500	1.451
BOND	C4H2s	C3H0b	1827.161	1.516
BOND	C3H0b	O1H0u	1480.000	1.231
! ile				

BOND	N3H1s	C4H1s	1639.889	1.458	
BOND	C4H1s	C3H0s	1342.404	1.525	
BOND	C4H1s	O1H0u	1480.000	1.231	
BOND	C4H1s	C4H1b	842.071	1.540	
BOND	C4H1b	C4H2b	1480.000	1.530	
BOND	C4H1b	C4H3u	543.618	1.521	
BOND	C4H1b	C4H3u	389.218	1.513	
!_leu	BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525	
BOND	C4H1s	O1H0u	1480.000	1.231	
BOND	C4H1s	C4H2s	1480.000	1.530	
BOND	C4H2s	C4H1b	1480.000	1.521	
BOND	C4H2s	C4H3u	543.618	1.521	
BOND	C4H1b	C4H3u	543.618	1.521	
!_lys	BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525	
BOND	C4H1s	O1H0u	1480.000	1.231	
BOND	C4H1s	C4H2s	1480.000	1.530	
BOND	C4H2s	C4H1b	1480.000	1.521	
BOND	C4H1b	C4H3u	543.618	1.521	
BOND	C4H1b	C4H3u	389.218	1.513	
!_met	BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525	
BOND	C4H1s	O1H0u	1480.000	1.231	
BOND	C4H1s	C4H2s	1480.000	1.530	
BOND	C4H2s	C4H1b	1480.000	1.520	
BOND	C4H2s	C4H2b	657.778	1.520	
BOND	C4H2s	C4H3u	657.778	1.520	
BOND	C4H2b	C4H1b	657.778	1.520	
BOND	C4H2b	N4H3u	657.778	1.520	
!_phe	BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525	
BOND	C4H1s	O1H0u	1480.000	1.231	
BOND	C4H1s	C4H2s	1480.000	1.530	
BOND	C4H2s	C4H1b	1480.000	1.520	
BOND	C4H2s	S2H0u	542.111	1.803	
BOND	S2H0u	C4H3u	170.066	1.791	
BOND	C4H1b	C4H3u	1639.889	1.458	
BOND	C4H1s	C3H0s	1342.404	1.525	
BOND	C4H1s	O1H0u	1480.000	1.231	
BOND	C4H1s	C4H2s	1480.000	1.530	
BOND	C4H2s	C4H1b	1480.000	1.520	
BOND	C4H2s	C4H2b	657.778	1.520	
BOND	C4H2s	C4H3u	657.778	1.520	
BOND	C4H2b	C4H1b	657.778	1.520	
BOND	C4H2b	C4H2d	236.800	1.492	
BOND	C4H2d	C4H1b	657.778	1.520	
!_pro	BOND	N3H0u	C4H1b	2631.111	1.466
BOND	C4H1b	C3H0s	1342.404	1.525	
BOND	C4H1b	O1H0u	1480.000	1.231	
BOND	C4H1b	C3H1b	1342.404	1.384	
BOND	C3H0s	C4H2s	3020.408	1.473	
BOND	C3H1b	C3H1b	657.778	1.382	
BOND	C4H1b	C4H2b	1480.000	1.530	
BOND	C4H2b	C4H2d	236.800	1.492	
BOND	C4H2d	C4H1b	657.778	1.520	
!_ser	BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525	
BOND	C4H1s	O1H0u	1480.000	1.231	
BOND	C4H1s	C4H2s	1480.000	1.530	
BOND	C4H2s	O2H1u	1480.000	1.417	
!_thr	BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525	
BOND	C4H1s	O1H0u	1480.000	1.231	
BOND	C4H1s	C4H2s	1480.000	1.530	
BOND	C4H2s	O2H1u	1480.000	1.417	
!_trp	BOND	N3H1s	C4H1s	1639.889	1.458
BOND	C4H1s	C3H0s	1342.404	1.525	

res	volume	sd
---	-----	----
URI	286.25	6.37
ADE	315.44	6.61
GUA	323.02	6.50
CYT	291.28	6.25
GLY	63.76	2.69
ALA	89.27	3.45
VAL	138.16	4.78
LEU	163.09	5.81
ILE	163.01	5.26
MET	165.82	5.41
PRO	121.29	3.73
HIS	157.46	4.25
PHE	190.84	4.83
TYR	194.63	4.86
TRP	226.38	5.33
CYH	112.84	5.49
CYS	102.50	3.50
SER	93.50	3.93
THR	119.61	4.23
ASN	122.35	4.60
GLN	146.91	4.29
GLU	138.81	4.32
ASP	114.43	3.87
LYS	165.08	6.89
ARG	190.33	4.73

```

# NucProt type set of 258 Volumes
#
# Generated: Mon Sep 27 15:23:05 2004
# Input file ALL-206766-edit.out
# Resi file resi-defs.dat
#
#          atom   Count  Mean volume    SD   SDC()  Minimum  Maximum
C          1641     8.801   0.461   5.241   7.821   10.113
C          1649     9.202   0.630   6.845   7.992   11.138
C          1519     16.605   1.944   11.709   12.885   23.166
C          1575     9.538   13.915   1.315   9.449   11.605
C          439      16.825   2.069   1.535   6.530   8.354
C          237      19.135   1.501   1.217   1.535   8.024
C          545      16.983   1.504   1.216   1.504   7.166
C          762      13.216   1.504   1.216   1.504   7.166
C          1499     12.701   0.841   6.624   10.964   15.379
C          1181     12.633   0.611   4.833   11.059   14.596
C          460      13.290   1.055   0.791   1.055   7.911
C          438      17.297   2.266   1.300   13.025   24.242
C          360      14.001   1.326   0.947   1.326   9.470
C          817      12.686   1.343   1.056   1.343   10.585
C          149      21.398   1.828   1.221   1.828   8.543
C          613      13.913   0.227   0.227   0.227   1.918
C          420      11.848   0.227   0.227   0.227   1.918
C          141      16.126   2.258   1.403   2.258   14.003
C          146      16.140   2.167   1.343   2.167   13.427
Total     15240   286.255   6.379   2.228

```

	atom	Count	Mean volume	SD	SDC()	Minimum	Maximum	
C	N1	1641	8.801	0.461	5.241	7.821	10.113	C8
	C2	1649	9.202	0.630	6.845	7.992	11.138	N9
	C3	1519	16.605	1.944	11.709	12.885	23.166	C1
	C4	1575	9.538	13.915	1.315	9.449	11.605	C2
	C5	439	16.825	2.069	1.535	6.530	8.354	C3
	C6	237	19.135	1.501	1.217	1.535	8.024	C4
	C7	545	16.983	1.504	1.216	1.504	7.166	C5
	C8	762	13.216	1.504	1.216	1.504	7.166	C6
	C9	1499	12.701	0.841	6.624	10.964	15.379	C7
	C10	1181	12.633	0.611	4.833	11.059	14.596	C8
C11	460	13.290	1.055	0.791	1.055	7.911	C9	
C12	438	17.297	2.266	1.300	13.025	24.242	C10	
C13	360	14.001	1.326	0.947	1.326	9.470	C11	
C14	817	12.686	1.343	1.056	1.343	10.585	C12	
C15	149	21.398	1.828	1.221	1.828	8.543	C13	
C16	613	13.913	0.227	0.227	0.227	1.918	C14	
C17	420	11.848	0.227	0.227	0.227	1.918	C15	
C18	141	16.126	2.258	1.403	2.258	14.003	C16	
C19	146	16.140	2.167	1.343	2.167	13.427	C17	
Total	15240	286.255	6.379	2.228			Total	

LEU	0	715	15.957	1.233	7.728	12.453	19.813	CB	317	23.426	2.161	9.226	2.884	29.421
C	1040	8.781	0.568	6.474	7.153	11.923	CG	518	9.695	0.628	6.478	8.236	13.946	
CA	905	13.055	0.827	6.336	10.522	15.954	CD1	325	20.057	1.102	8.484	16.520	25.352	
CA	984	13.517	0.881	6.517	10.692	19.657	CD2	309	20.578	1.631	7.927	16.348	26.402	
N	698	22.818	2.182	9.565	4.902	28.870	CE1	220	20.534	1.708	8.319	16.373	28.543	
CG	915	14.704	1.267	8.617	4.515	19.817	CE2	210	20.577	1.778	8.640	16.849	26.511	
CD1	566	37.235	3.364	9.035	19.102	47.591	CZ	397	9.888	0.684	6.922	7.999	12.329	
CD2	496	37.020	3.568	9.637	10.739	48.835	OH	111	18.541	1.707	9.204	13.640	24.609	
Total	6319	163.087	5.807	3.561			Total	4099	194.633	4.862	2.498			
ILE	0	507	15.930	1.205	7.566	12.369	20.021	TRP	0	159	15.797	1.272	8.053	13.432
C	788	8.445	0.534	6.326	6.985	10.674	CA	223	8.687	0.667	7.674	7.512	10.764	
CA	623	12.946	0.390	7.646	10.734	16.296	N	169	13.323	0.890	6.679	11.370	17.390	
N	684	13.493	0.855	6.297	9.891	17.091	CB	148	23.826	1.821	7.643	11.243	16.415	
CB	623	14.146	1.145	8.094	3.279	18.783	CG	224	9.915	0.667	7.25	8.503	12.159	
CG1	511	24.071	1.877	7.797	19.425	30.851	CD1	120	20.597	2.029	9.850	10.214	27.057	
CG2	386	35.763	3.154	8.888	7.616	44.492	NE1	122	16.723	1.448	8.659	13.008	21.982	
CD1	371	38.219	3.054	7.990	30.552	48.568	CD2	208	10.068	0.705	7.006	8.550	12.190	
Total	4493	163.014	5.263	3.229			CE2	203	9.848	0.738	7.488	8.033	12.266	
PRO	0	235	15.856	1.312	8.277	12.756	20.177	CE3	174	20.383	1.977	9.701	11.543	25.618
C	560	8.768	0.436	4.971	7.749	10.487	CZ3	142	21.429	1.149	8.164	17.801	26.278	
CA	322	13.828	1.031	7.456	10.897	16.417	CZ2	107	20.931	1.755	8.387	17.162	26.251	
N	581	8.650	0.594	6.869	7.135	10.490	CH2	122	21.219	1.911	9.005	16.939	26.508	
CB	127	25.314	2.073	8.189	20.702	31.594	Total	2308	226.384	5.333	2.356			
CG	118	25.480	1.791	7.030	21.639	31.080	SER	0	469	15.860	1.353	8.530	8.680	
CD	151	23.390	1.750	7.482	18.654	29.741	C	822	8.858	0.886	6.618	3.793	11.169	
Total	2094	121.285	3.728	3.074			CA	485	13.351	1.115	8.350	3.254	16.818	
MET	0	174	16.002	1.382	8.639	12.866	20.826	N	570	13.808	1.030	7.462	3.911	17.190
C	251	8.756	0.583	6.658	7.648	10.773	CB	228	23.599	2.870	12.161	4.028	30.962	
CA	214	13.194	1.021	7.755	10.697	18.186	OG	245	18.021	1.650	9.157	13.980	23.787	
N	230	13.405	0.918	6.847	11.150	16.277	Total	2819	93.497	3.929	4.202			
CB	171	23.418	1.686	7.201	18.912	27.823	THR	0	451	15.795	1.260	7.974	12.619	
CG	163	23.830	1.802	7.563	19.002	28.746	C	870	8.685	0.370	6.561	6.970	20.144	
SD	162	30.207	2.398	7.939	23.949	36.746	CA	553	13.025	0.900	6.911	11.218	20.543	
CE	108	37.003	3.645	9.849	25.817	50.849	N	692	13.544	1.275	8.775	11.081	21.357	
Total	1473	165.815	5.409	3.262			CB	328	14.687	1.275	8.682	9.862	20.415	
PHE	0	408	15.961	1.212	7.594	12.657	21.717	OG1	203	17.610	1.749	9.932	14.499	23.506
C	583	8.697	0.638	7.335	7.312	10.801	CG2	180	36.265	3.098	8.544	24.755	46.100	
CA	438	13.371	0.960	7.182	11.396	16.507	Total	3267	119.613	4.225	3.532			
N	510	13.524	0.918	6.790	11.156	16.689	ASN	0	278	15.857	1.273	8.028	12.751	
CB	371	23.623	1.862	7.881	18.782	30.259	C	572	8.853	0.611	6.900	7.385	21.292	
CG	614	9.684	0.635	6.558	8.137	12.619	CA	345	13.052	0.859	6.582	9.973	17.336	
CD1	406	20.325	1.683	8.282	16.120	26.870	N	555	13.525	0.993	7.344	10.868	16.962	
CD2	414	20.948	1.674	7.989	17.430	27.179	CB	162	22.756	2.024	12.832	31.832		
CE1	366	21.532	1.868	8.674	16.182	28.524	CG	309	9.537	0.765	8.026	4.363	11.974	
CE2	374	21.625	1.781	8.237	17.212	28.819	OD1	151	16.247	1.840	11.328	10.481	22.936	
CZ	364	21.555	1.864	8.645	16.959	28.882	ND2	85	22.525	2.206	9.705	16.703	28.777	
Total	4848	190.843	4.825	2.528			Total	2457	122.353	4.595	3.755			
TYR	0	337	15.901	1.159	7.286	12.315	19.028	GLN	0	230	15.767	1.326	8.410	11.827
C	514	8.714	0.630	7.232	7.344	10.902	C	431	8.744	0.598	6.838	7.473	11.383	
CA	400	13.249	0.950	7.171	9.530	16.972	CA	278	13.231	0.955	7.217	10.590	16.396	
N	441	13.473	0.959	7.120	11.203	16.559	N	399	13.449	0.971	7.238	11.173	16.455	
							CB	142	23.059	1.846	8.004	19.186	28.112	

CG	133	23.248	1.655	7.129	19.335	26.926	C	544	8.779	0.625	7.125	3.431	10.771
CD	187	9.618	0.674	7.006	8.129	12.136	CA	379	13.310	0.995	7.473	11.125	16.716
OE1	84	16.571	1.376	8.301	13.524	20.525	N	455	13.486	0.967	7.173	10.623	16.858
NE2	58	23.255	2.442	10.499	16.772	30.231	CB	200	22.833	1.638	7.174	18.446	27.809
Total	1942	146.913	4.292	2.922			CG	194	23.273	2.139	9.190	7.482	29.048
CYS	0	64	16.382	1.355	8.268	13.680	20.134	CD	131	22.849	1.763	18.509	28.264
C	93	8.786	0.513	5.894	7.648	10.374	NE	141	15.019	1.077	7.168	11.909	20.045
CA	73	13.583	1.153	8.488	11.901	16.712	CZ	210	9.678	0.708	7.311	8.335	12.292
N	85	13.865	1.153	12.447	11.957	16.697	NH1	62	22.056	1.485	6.731	18.559	26.141
CB	52	23.471	2.921	12.447	10.577	27.841	NH2	45	23.132	2.119	9.161	19.492	28.817
SG	48	36.748	4.156	11.308	22.869	46.139	Total	2673	190.331	4.726	2.483		
CSS	0	82	16.093	1.300	8.076	13.462	LYS	0	335	15.818	1.118	7.068	13.086
C	145	8.742	0.584	6.677	7.538	10.172	C	708	8.696	0.576	6.619	7.127	19.726
CA	129	13.081	0.894	6.836	9.753	15.470	CA	356	13.217	0.598	6.793	10.990	16.590
N	119	13.631	0.808	5.927	12.159	15.563	N	566	13.429	1.030	7.668	10.059	
CB	80	23.447	2.044	8.718	19.583	30.500	CB	158	22.578	3.443	15.249	3.876	28.192
SG	101	27.507	2.134	7.758	22.691	34.501	CG	140	22.847	3.507	15.350	2.895	28.765
Total	656	102.500	3.495	3.410			CD	69	23.365	3.880	16.604	2.982	31.253
HIS	0	163	15.855	1.253	7.903	12.442	CE	14	23.720	1.882	7.935	19.103	26.122
C	277	8.760	0.565	6.447	7.122	10.692	NZ	8	21.413	1.153	5.383	19.680	23.588
CA	166	13.335	1.011	7.583	11.630	16.200	Total	2354	165.083	6.894	4.176		
N	235	13.532	1.058	7.821	10.920								
CB	120	23.443	1.819	7.758	19.616								
CG	213	9.870	0.668	6.768	8.599								
ND1	115	15.483	1.576	10.176	10.209								
CD2	101	20.938	1.803	8.610	16.895								
CE1	51	20.491	1.712	8.354	16.288								
NE2	68	15.758	1.261	8.003	13.114								
Total	1509	157.464	4.247	2.697									
GLU	0	279	15.765	1.240	7.868	12.513							
C	622	8.631	0.529	6.150	7.417	10.656							
CA	345	13.284	0.304	6.803	11.144	16.790							
N	514	13.461	0.943	7.008	10.838	16.833							
CB	157	23.214	1.748	7.529	19.259	28.227							
CG	124	23.304	2.691	11.550	3.556	28.818							
CD	152	9.437	0.738	7.816	5.524	11.327							
OE1	68	15.497	1.312	8.465	9.874	18.286							
OE2	50	16.243	1.609	9.921	13.079	22.335							
Total	2311	138.805	4.321	3.113									
ASP	0	368	15.757	1.183	7.505	11.100							
C	736	8.750	0.579	6.613	6.632	11.012							
CA	389	13.254	0.978	7.377	10.983	17.720							
N	573	13.654	0.949	6.951	11.073	17.257							
CB	142	23.022	2.157	9.371	9.509	26.698							
CG	264	9.336	0.879	9.419	2.607	11.090							
OD1	183	15.078	1.538	10.200	9.252	19.548							
OD2	86	15.582	1.890	12.129	9.288	22.577							
Total	2741	114.433	3.867	3.379									
ARG	0	312	15.916	1.139	7.158	12.805							

```
# NucProt type set of 258 Volumes  
# Generated: Sun Mar 7 16:03:09 2004  
# Input file ALL-200766-edit.out  
# Resi file ../../spec-dets.dat  
#
```

# NucProt type set of 258 Volumes												
# Generated: Sun Mar 7 16:03:09 2004												
# Input file ALI-206766-edit.out												
# Resi File ../../spec-defs.dat												
URI	atom	type	count	% OK	mode	mean	% SD	min	max	mode	mean	% SD
N1	N3H0u	C2	1641	74.7	8.71	8.801	0.46	5.24	7.821	10.113	12.306	15.537
C2	C3H0s	C3H1t	1649	75.1	9.01	9.202	0.63	6.84	7.992	11.138	12.306	15.537
O1H	O1H0u	O1H1t	849	38.6	16.15	16.605	1.94	11.7	12.885	23.166	24.234	25.500
N3	N3H1s	N3H2t	1519	69.1	13.18	13.915	1.31	9.45	11.605	18.385	19.423	20.598
C4	C4H0s	C4H1t	1375	62.6	9.35	9.538	0.62	6.53	8.354	11.414	12.224	13.714
O4	O1H0u	O1H1t	439	20.0	15.81	16.825	2.07	12.3	12.979	23.689	24.234	25.500
C5	C3H1t	C3H2t	237	10.8	18.89	19.135	1.54	8.02	15.431	22.862	23.689	25.500
C6	C3H1t	C3H2t	545	24.8	16.42	16.983	1.22	7.17	14.238	20.297	21.234	22.555
TOTAL					8254	47.0		107.5	111.00	3.82	91.31	141.06

GUA											
atom	type	count	%	OK	mode	mean	sd	%	SD	min	max
N1	N3H1s	3292	77.8	13.01	13.489	1.07	7.90	11.592	17.418	-	-
C2	C3H0s	3696	87.4	8.78	9.033	0.58	6.41	7.982	10.891	-	-
N2	N3H0s	1158	27.4	21.16	21.736	1.91	8.80	17.547	27.339	-	-
N3	N2H0b	1490	35.2	14.19	14.961	1.47	9.83	12.164	19.489	-	-
C4	C3H0s	3631	85.8	8.86	9.030	0.55	6.12	8.008	10.859	-	-
C5	C3H0s	3485	82.4	9.05	9.239	0.55	5.96	8.175	11.040	-	-
C6	C3H0s	3538	83.6	9.10	9.265	0.53	5.75	8.205	10.961	-	-
O6	O1H0u	895	21.2	15.88	16.410	1.88	11.5	12.960	22.975	-	-
N7	N2H0b	624	14.7	15.37	15.888	1.38	8.71	12.672	20.115	-	-
C8	C3H1s	732	17.3	18.40	18.213	1.38	7.60	14.447	22.032	-	-
N9	N3H0u	3499	82.7	8.69	8.765	0.43	4.96	7.857	10.014	-	-
TOTAL		26040	56.0	142.5	146.04	3.97	121.81	183.13	-	-	-
CYT											
atom	type	count	%	OK	mode	mean	sd	%	SD	min	max
N1	N3H0u	2871	83.8	8.68	8.811	0.45	5.15	7.831	10.123	-	-
C2	C3H0s	2922	85.3	9.01	9.311	0.64	6.85	8.134	11.264	-	-
O2	O1H0u	1174	34.3	14.28	15.744	1.88	11.9	12.152	21.595	-	-
N3	N2H0s	2765	80.7	12.78	13.082	1.00	7.65	11.056	16.656	-	-
N3H0s	N2H0s	2688	78.4	9.26	9.406	0.55	5.83	8.242	11.144	-	-
C4	C3H0s	438	12.8	21.78	22.475	1.85	8.22	18.362	27.281	-	-
C5	C3H1b	349	10.2	20.12	19.446	1.48	7.39	15.832	22.715	-	-
C6	C3H1t	842	24.6	16.96	16.920	1.18	6.97	14.039	20.203	-	-
TOTAL		14049	51.2	112.9	115.19	3.53	95.96	140.98	-	-	-

D.2 PDB Files for the Exit Tunnel

- beta-turn.pdb: coordinates of a beta-turn peptide in the exit tunnel.
- tungsten-11.pdb: coordinates of the tungsten-11 clusters (W11) lodged in the exit tunnel.

		ATOM	ATOM	ATOM	HB2 ALA	HB2 ALA	N
		ATOM	ATOM	ATOM	HB3 ALA	HB3 ALA	H
		END					H
1	N	AMN	73.738	126.286	95.198	1.00	0.00
2	H1	AMN	72.977	126.873	95.352	1.00	0.00
3	H2	AMN	73.976	125.715	96.024	1.00	0.00
4	H3	AMN	74.609	126.870	94.995	1.00	0.00
5	C	CXL	68.544	122.781	95.573	1.00	0.00
6	O1	CXL	68.073	123.291	96.588	1.00	0.00
7	O2	CXL	68.490	121.563	95.416	1.00	0.00
8	CA	ALA	73.545	125.402	94.051	1.00	0.00
9	C	ALA	71.444	125.175	94.513	1.00	0.00
10	O	ALA	72.506	123.946	95.282	1.00	0.00
11	CB	ALA	73.324	127.428	92.848	1.00	0.00
12	HA	ALA	72.319	124.527	94.366	1.00	0.00
13	HB1	ALA	72.740	126.225	90.453	1.00	0.00
14	HB2	ALA	71.335	123.842	93.527	1.00	0.00
15	HB3	ALA	70.446	126.201	91.030	1.00	0.00
16	N	ALA	72.944	125.278	89.257	1.00	0.00
17	CA	ALA	72.937	124.515	91.698	1.00	0.00
18	C	ALA	73.445	127.066	90.362	1.00	0.00
19	O	ALA	72.762	125.828	88.321	1.00	0.00
20	CB	ALA	72.235	124.440	89.333	1.00	0.00
21	H	ALA	73.973	124.891	89.264	1.00	0.00
22	HA	ALA	71.142	127.817	89.601	1.00	0.00
23	HB1	ALA	73.402	128.341	89.400	1.00	0.00
24	HB2	ALA	69.933	127.249	88.844	1.00	0.00
25	HB3	ALA	67.747	127.210	89.130	1.00	0.00
26	N	ALA	69.878	129.489	88.376	1.00	0.00
27	CA	ALA	71.142	127.817	89.601	1.00	0.00
28	C	ALA	69.390	128.713	90.350	1.00	0.00
29	O	ALA	68.868	129.890	88.200	1.00	0.00
30	CB	ALA	70.291	129.106	87.431	1.00	0.00
31	H	ALA	70.528	130.285	88.766	1.00	0.00
32	HA	ALA	69.557	126.348	88.060	1.00	0.00
33	HB1	ALA	68.838	125.207	87.528	1.00	0.00
34	HB2	ALA	70.531	126.436	87.851	1.00	0.00
35	HB3	ALA	68.070	125.543	86.815	1.00	0.00
36	N	ALA	69.341	123.425	86.396	1.00	0.00
37	CA	ALA	68.205	124.414	88.636	1.00	0.00
38	C	ALA	67.087	123.953	88.466	1.00	0.00
39	O	ALA	67.332	124.868	85.994	1.00	0.00
40	CB	ALA	69.856	124.305	86.808	1.00	0.00
41	H	ALA	70.531	126.436	87.851	1.00	0.00
42	HA	ALA	68.343	123.471	90.856	1.00	0.00
43	HB1	ALA	69.166	123.591	92.105	1.00	0.00
44	HB2	ALA	70.623	123.979	87.527	1.00	0.00
45	HB3	ALA	68.274	121.985	90.464	1.00	0.00
46	N	ALA	69.817	124.665	89.896	1.00	0.00
47	CA	ALA	68.911	124.254	89.773	1.00	0.00
48	C	ALA	67.323	123.827	91.063	1.00	0.00
49	O	ALA	67.831	121.408	91.289	1.00	0.00
50	CB	ALA	69.290	121.614	90.261	1.00	0.00
51	H	ALA	67.654	121.871	89.562	1.00	0.00
52	HA	ALA	68.482	123.510	93.262	1.00	0.00
53	HB1	ALA	69.183	123.646	94.525	1.00	0.00
54	HB2	ALA	69.118	125.119	94.967	1.00	0.00
55	HB3	ALA	67.490	123.392	93.271	1.00	0.00
56	CB	ALA	70.234	123.349	94.402	1.00	0.00
57	CA	ALA	69.655	125.240	95.918	1.00	0.00
58	CB	ALA					
59	H	ALA					
60	HA	ALA					
61	HB1	ALA					

ATOM	28	0	W11	8	43.383	154.644	70.225	1.00	0.00	
ATOM	29	0	W11	8	40.071	154.541	66.890	1.00	0.00	
ATOM	30	0	W11	8	40.058	152.965	65.392	1.00	0.00	
ATOM	31	0	W11	8	38.545	152.999	66.924	1.00	0.00	
ATOM	32	0	W11	8	43.425	148.164	67.068	1.00	0.00	
ATOM	33	0	W11	8	43.426	149.664	65.491	1.00	0.00	
ATOM	34	0	W11	8	44.957	149.708	67.037	1.00	0.00	
ATOM	35	0	W11	8	40.091	148.235	70.374	1.00	0.00	
ATOM	36	0	W11	8	40.078	149.809	71.871	1.00	0.00	
ATOM	37	0	W11	8	38.547	149.767	70.331	1.00	0.00	
ATOM	38	0	W11	8	40.378	154.222	69.907	1.00	0.00	
ATOM	39	0	W11	8	38.934	152.765	69.936	1.00	0.00	
ATOM	40	0	W11	8	40.382	152.810	71.396	1.00	0.00	
ATOM	41	0	W11	8	43.095	148.638	70.059	1.00	0.00	
ATOM	42	0	W11	8	43.087	150.130	71.475	1.00	0.00	
ATOM	43	0	W11	8	44.544	150.102	70.030	1.00	0.00	
ATOM	44	0	W11	8	40.414	148.559	67.362	1.00	0.00	
ATOM	45	0	W11	8	40.416	149.976	65.870	1.00	0.00	
ATOM	46	0	W11	8	38.956	150.005	67.319	1.00	0.00	
ATOM	47	0	W11	8	43.095	152.675	65.811	1.00	0.00	
ATOM	48	0	W11	8	44.543	152.718	67.271	1.00	0.00	
ATOM	49	0	W11	8	43.081	154.166	67.225	1.00	0.00	
ATOM	2	0	W11	9	61.506	138.343	78.940	1.00	0.00	
ATOM	3	0	W11	9	61.456	137.174	77.726	1.00	0.00	
ATOM	4	0	W11	9	57.664	138.432	75.078	1.00	0.00	
ATOM	5	0	W11	9	58.850	137.234	75.107	1.00	0.00	
ATOM	6	0	W11	9	65.037	130.990	74.965	1.00	0.00	
ATOM	7	0	W11	9	63.851	132.186	75.030	1.00	0.00	
ATOM	8	0	W11	9	57.629	131.020	75.224	1.00	0.00	
ATOM	9	0	W11	9	58.827	132.207	75.206	1.00	0.00	
ATOM	10	0	W11	9	61.171	130.941	78.806	1.00	0.00	
ATOM	11	0	W11	9	61.229	132.154	77.635	1.00	0.00	
ATOM	12	0	W11	9	61.234	138.482	71.532	1.00	0.00	
ATOM	13	0	W11	9	61.271	137.268	72.701	1.00	0.00	
ATOM	14	0	W11	9	61.486	131.076	71.393	1.00	0.00	
ATOM	15	0	W11	9	61.442	132.245	72.607	1.00	0.00	
ATOM	16	0	W11	9	65.044	134.765	78.886	1.00	0.00	
ATOM	17	0	W11	9	57.656	134.941	71.458	1.00	0.00	
ATOM	18	0	W11	9	65.070	134.620	71.476	1.00	0.00	
ATOM	19	0	W11	9	63.874	134.649	72.664	1.00	0.00	
ATOM	20	0	W11	9	57.633	134.513	78.855	1.00	0.00	
ATOM	21	0	W11	9	58.829	134.576	77.669	1.00	0.00	
ATOM	22	0	W11	9	63.016	133.104	71.944	1.00	0.00	
ATOM	23	0	W11	9	64.547	133.065	73.487	1.00	0.00	
ATOM	24	0	W11	9	59.708	137.944	73.561	1.00	0.00	
ATOM	25	0	W11	9	59.706	136.437	71.997	1.00	0.00	
ATOM	26	0	W11	9	58.166	136.416	73.529	1.00	0.00	
ATOM	27	0	W11	9	63.002	131.535	73.452	1.00	0.00	
ATOM	28	0	W11	9	64.547	133.065	73.487	1.00	0.00	
ATOM	29	0	W11	9	59.708	137.867	76.890	1.00	0.00	
ATOM	30	0	W11	9	59.667	131.490	76.763	1.00	0.00	
ATOM	31	0	W11	9	59.669	132.999	78.328	1.00	0.00	
ATOM	32	0	W11	9	58.139	133.032	76.784	1.00	0.00	
ATOM	33	0	W11	9	60.015	137.490	76.558	1.00	0.00	
ATOM	34	0	W11	9	58.553	136.041	76.527	1.00	0.00	
ATOM	35	0	W11	9	60.003	136.007	77.987	1.00	0.00	
ATOM	36	0	W11	9	62.714	137.527	73.388	1.00	0.00	

Appendix E

Perl and Shell Scripts

E.1 General Perl and Shell Scripts

- fastnet.pl: reads a PDB of any type and removes atoms within a certain distance of each other effectively creating a net of the original atoms. This program was used to reduce the number of atoms exported by the ³V package into something graphic software can use.
- filetools.pl: a perl library used to manipulate files.
- fixnum.pl: a perl library to export a fixed number of decimal places. Good for export numbers in PDB format of 4.3.

```

#!/bin/perl -w
use strict;
use Getopt::Std;
require '/home/vossman/lib/fixnum.pl';
print STDERR "\n\n";
my %args;
getopt('ior',\%args);
my $pdbfile = $args{i} || "tunnel-1ij2.pdb";
my $outfile = $args{o} || "small-$pdbfile";
my $currend = $args{r} || 0.6; #angstroms
print STDERR "Reading Spdbfile and creating net of radius $currend\n",
"Then writing to $outfile\n";
open(PDB, "< $pdbfile") or die;
open(OUT, "> $outfile") or die;
my %GRID;
my $roughgrid = $currend*1.0;
my $scutdist = $currend**2;
my $count;
my $atoms;
my $atomsdef;
while(defined(my $line = <PDB>)) {
    if(length($line) > 60 && ($line =~ /^HETATM/ || $line =~ /\^ATOM/ )) {
$atoms++;
my ($x, $y, $z) = getCoord($line);
my $key = xyz2key($x,$y,$z);
if($key && !$GRID{$key}) {
$GRID{$key} = 1;
$count++;
print OUT $line;
if($count % 50 == 0) { print STDERR "..."; }
if($count % 3800 == 0) { print STDERR "\n"; }
}
}
close(OUT); close(PDB);
print STDERR "\nCreated $count atoms of $atoms into netfile $outfile\n\n";
sub getCoord {
my ($line) = @_;
my ($x,$y,$z);
my $x = substr($line,30,8);
my $y = substr($line,38,8);
my $z = substr($line,46,8);
return ($x,$y,$z);
}
sub list2vec {
my (@list) = @_;
my $vec;
foreach my $item (@list) {
$vec .= $item." ";
}
return $vec;
}
sub vec2list {
my ($vec) = @_;
return split(/\s+/, $vec);
}
sub varyKey {
my ($key) = @_;
my @keys;

```

```

#!/usr/bin/perl -w
use strict;
my ($in) = @_;
system("wc -l $in > temp.txt");
open(IN, '< temp.txt') or die;
my $num = <IN>;
close(IN);
my @chunks = split(' ', $num);
$num = $chunks[0];
$num =~ s/^\s+//; $num =~ s/\s+$/;;
return $num;
};

#####
sub getNumLines {
    my ($in) = @_;
    system("wc -l $in > temp.txt");
    open(IN, '< temp.txt') or die;
    my $num = <IN>;
    close(IN);
    my @chunks = split(' ', $num);
    $num = $chunks[0];
    $num =~ s/^\s+//; $num =~ s/\s+$/;;
    return $num;
};

#####
sub printBar {
    my ($num) = @_;
    my $mult=1;
    while($num/$mult > 80) {
        $mult++;
    }
    $num = int($num/$mult+0.5);
    print STDERR "#";
    for(my $i=1;$i<=$num-2;$i++) {
        if($i==int(($num-2)/2+0.501)) { print STDERR "\n"; }
        elsif($i==int(($num-2)/4+0.501)) { print STDERR "#"; }
        elsif($i==int(3*($num-2)/4+0.501)) { print STDERR "#"; }
        elsif($i%5==0) { print STDERR "#"; }
        else { print STDERR "#"; }
    }
    if($i%100==0) { print STDERR "\n"; }
    print STDERR "\n";
    return $mult;
};

#####
sub clean {
    my ($str) = @_;
    $str =~ s/^\s+//; $str =~ s/\s+$/;;
    return $str;
};

#####
sub list2vec {
    my (@list) = @_;
    my $vec;
    foreach my $item (@list) {
        $vec .= $item." ";
    }
    return $vec;
};

#####
sub vec2list {
    my ($vec) = @_;
    return split(' ', $vec);
};

```

```

#!/usr/bin/perl -w

# ALL PURPOSE GENERAL PROGRAM
sub fixnum {
    # num - number to edit
    # dig - number of preceding digits
    # dec - number of decimal places
    # type - one of
    #   r - regular integer
    #   z - zerofill
    #   p - leading plus or minus
    my $debug = 0;
}

my ($num,$dig,$dec,$type) = (@_, "#SAVE ORIGINAL REQUEST");
print STDERR "FIXNUM: $num NOT A NUMBER\n";
return $num;

#Setup
my ($num,$dig,$dec) = ($num,$dig,$dec);
if($debug) { print STDERR "start num: $num\n"; }
if($debug) { print STDERR " digits: $dig\n"; }
if($debug) { print STDERR " decimals: $dec\n"; }
if($num) {
    if($type) {
        $num =~ s/\$/s$/;;
    }
    $type = substr($type,0,1);
    if($type eq 'z') { $spacer = "0"; }
    else { $spacer = "+"; }
    if($debug) { print STDERR " spacer: '$spacer'\n"; }
    # $num = int($num*(10**($dec+1))+0.5*signFunc($num))/(10**($dec+1));
}

#Start modifications
#round
$num = int($num*(10**$dec))+0.5*signFunc($num));

#copy to final product
my $num2 = $num;
if($debug) { print STDERR " rounded: '$num2'\n"; }

$add = $dec + $dig - length($num);
if($debug) { print STDERR " need to add $add spaces\n"; }

# $add--;
# if($add < 0) { print STDERR " neg: need to add $add spaces\n"; }
# if($add < 0) { #too many digits for this mode
#     $add is negative, so really minus
#     $dec+$add;
#     #re-round
#     if($dec < 0) { $dec++; }
#     $num = int($num*(10**$dec)+0.5*signFunc($num));
#     $num2 = $num; # copy to final product
#     #re-copy to final product
#     while($add > 0) {
#         if($add > 0) { #not enough digits
#             if($debug) { print STDERR " adding $add spaces\n"; }
#             $add digit in front
#             if($add > $num >= 0) {
#                 $num2 = "0" . $num2;
#                 elsif($add > $dig-1 && $num < 0) {
#                     $num2 =~ s/-0//;
#                     $num2 = "-". $num2;
#                 }
#             }
#         }
#     }
# }

#SPECIAL CHECK FOR P-MODE
if($type eq "p") {
    $num2 =~ s/^\+/\+/g;
    $num2 =~ s/\+\+/\+/g;
    $num2 =~ s/\+\-/ \-/g;
}
#Make sure decimals are filled
if($dec > 0) {
    substr($num,-1*$dec,0) .= ".";
}
if($debug) { print STDERR " after dec mod: '$num2'\n"; }

#RETURN!
return $num2;
}

#Old Function IDs
sub fixnum0 { my ($num)=@_; fixnum($num,1,0,'r'); };
sub fixnum11 { my ($num)=@_; fixnum($num,1,1,'r'); };
sub fixnum12 { my ($num)=@_; fixnum($num,1,2,'r'); };
sub fixnum20 { my ($num)=@_; fixnum($num,2,0,'r'); };
sub fixnum21 { my ($num)=@_; fixnum($num,2,1,'r'); };
sub fixnum22 { my ($num)=@_; fixnum($num,2,2,'r'); };
sub fixnum30 { my ($num)=@_; fixnum($num,3,0,'r'); };
sub fixnum31 { my ($num)=@_; fixnum($num,3,1,'r'); };
sub fixnum32 { my ($num)=@_; fixnum($num,3,2,'r'); };
sub fixnum40 { my ($num)=@_; fixnum($num,4,0,'r'); };
sub fixnum41 { my ($num)=@_; fixnum($num,4,1,'r'); };
sub fixnum42 { my ($num)=@_; fixnum($num,4,2,'r'); };
sub fixnum50 { my ($num)=@_; fixnum($num,5,0,'r'); };
sub fixnum51 { my ($num)=@_; fixnum($num,5,1,'r'); };
sub fixnum52 { my ($num)=@_; fixnum($num,5,2,'r'); };

return 1;
}

```

E.2 Perl Scripts for the Voronoi Calculations

- bond-lengths.pl: is a program that takes all the defined bond lengths for RNA and protein and finds the best bonded VDW radii for the combined atoms.
- find_min-stdev.pl: is a program that varies the non-bonded radii for one or two atoms and finds the total percent standard deviation or self-consistency, Φ . This was used to find the non-bonded radii for the *P4H0u*, *O2H0s*, and *O2H0u* atom types.
- gen_atomvol.pl: is a program that takes the Voronoi volume output files and generates the **atom-vols.dat** file
- gen_basesize.pl: is a program that takes the Voronoi volume output files and generates the **resi-vols.dat** file
- gen_histogram.pl: is a program that takes the Voronoi volume output files and generates a histogram based on residue atom kind or atom type.
- gen_stdvol-classic.pl: is a program that takes the Voronoi volume output files and generates the **stdvol258.dat** file.
- gen_stdvol-extended.pl: is a program that takes the Voronoi volume output files and generates the **stdvol-extended.dat** file.
- rm_extreme-pts.pl: is a program that takes the Voronoi volume output files and removes the extreme volumes from each distribution for a more self-consistent set.
- rm_symmetry.pl: is a program that takes the Voronoi volume output files and the original pdb file and removes all symmetry-generated atom from the output file leaving only the original atoms in the output file.
- run_voronoi.pl: is a program that takes a PDB file and runs the entire Voronoi volume process on it.
- superpak8.pl: is a program that reads the PDB header and generates all molecules near the original molecule ignoring the crystallographic unit cell. The program is useful for exploring crystal packing interactions not only between molecules in the same unit cell, but those from other unit cells.

```

#!/usr/bin/perl
use strict;
require '/home/vossmann/lib/fixnum.p1';

strand (12);

my $min=10000;
my $rna=1;
my %isRNA;
my $max=10;
my %isType;
my %fix;
my @types;
my @rules;

open(IN, "< atom-defs.fix") or die;
while(defined(my $line = <IN>)) {
    if(substr($line, 0, 4) eq "PRO") { $rna = 0; }
    if(substr($line, 0, 4) eq "BOND") {
        my @chunks = split(" ", $line);
        my $print = "BOND ";
        #print $DERR "BOND ", fill($chunks[1],7), fill($chunks[2],7),
        #fixnum43($chunks[3]), fill($chunks[4],7);
        my $type = substr($chunks[1],0,4); my $type2 = substr($chunks[2],0,4);
        my $rule = $type." ".$type2." ".$chunks[3]." ".$chunks[4];
        push(@rules,$rule);
        if($rna) { $isRNA{$type} = 1; }
        if($isType{$type}) {
            push(@types,$type);
            $isType{$type}=rand(1);
        }
        if($rna) { $isRNA{$type2} = 1; }
        if($isType{$type2}) {
            push(@types,$type2);
            $isType{$type2}=rand(1);
        }
    }
}
close(IN);

my $oldterr=1; my $terr=0;
for(my $i=0;$i<=$max;$i++) {
    $olderr = $terr;
    $terr=0;
    print STDERR "initier $in\n";
    @types = shuffle(@types);
    if($i==$max) { @types = sort { $a cmp $b } @types; }

    foreach my $type (@types) {
        if($isRNA{$type}) { print STDERR "$type\n"; }
        my $sum=0; my $count=0; my $sumsq=0; my $num=0;
        print STDERR "$type\n";
        foreach my $rule (@rules) {
            my @bits = split(" ", $rule);
            foreach my $type (@types) {
                if($bits[0] eq $type) { $oth = $bits[1]; }
                else { $oth = $bits[3]-$isType{$type}*$bits[2]; }
                $sum += ($bits[3]-$isType{$type}-$isType{$oth})*$bits[2];
                $sumsq += ($isType{$type}+$isType{$oth})*$bits[3];
                $count+= $bits[2];
            }
            $num++;
        }
        my $err = sqrt($sumsq/$count)*100;
        print STDERR fixnum25($isType{$type}), "\t",
        fixnum25($terr), "\t$terr\n";
        $terr += $err;
    }
    print STDERR "total error ----- \"$terr,\" best $min\n";
    sub shuffle {
        my (@list) = @_;
        my @newlist;
        while(@list >= 0) {
            my $num = int(rand(@list)+1);
            push(@newlist,$list[$num]);
            $list[$num]=@list[$#list];
            $#list--;
        }
        return @newlist;
    };
    sub fill {
        my ($str,$num) = @_;
        while(length($str) < $num) {
            $str .= " ";
        }
        return $str;
    };
}
my $err = $terr;
print STDERR fixnum25($terr), "\t$terr\n";
$terr += $err;

```

```

#!/usr/bin/perl
use Getopt::Std;
require "ctime.pl";
use strict;
my $version = 3.0;
print STDERR "$^V\n";
my $bondrard = 0.65;
my ($type1,$type2,$atomdef,$residef,$stdvol,$outfile,
    $min,$max,$incr,$debug,$spec,$listfile,$type2) = getINPUT();
writefile($outfile);
if($debug) { print STDOUT "Start $min\nStop $max\nIncr $incr\nSpec $spec\nList $listfile
\n"; }
foreach my $phos (1.54..1.64..1.74) {
    foreach my $phos2 (1.54..1.64..1.74) {
        for my $phos=-1.40;$phos<=1.60;$phos+=0.01) {
            for my $phos2=-1.52;$phos2<=1.72;$phos2+=0.01) {
                my $start = getTime();
                if($debug) { getTime(); }
                my $types = editAtomDefs($atomdef,$phos,$phos2);
                if($debug) { print STDERR "Getting PDB IDs\n"; }
                my @pds = getpdbIDs();
                my $outf = "PHOS-$fixnum12($phos).-$temp-$spec.out";
                if($debug) { print STDERR "Getting Volumes into $outf\n"; }
                foreach my $pdid (@pds) {
                    my $temp = "temp-$spec.out";
                    if($debug) { print STDERR "Get volume for ".lc($pdid),"\n"; }
                    if($debug) { print STDERR "perl getvolumes.pl -a $types -r $pdid -o $temp\n"; }
                    system("perl getvolumes.pl -a $types -r $pdid -o $temp -u $spec");
                    if($stat("temp")) { die "no temp file\n"; }
                    system("cat $temp > $outf");
                    system("rm -f $temp");
                }
                if($debug) { print STDERR "Starting analysis\n"; }
                my ($totdev,$totperdev,$rnaatoms,$count,$totatoms)
                    = AnalyzeVol($outf);
                if($debug) { print STDERR "Outputting Info\n"; }
                printOutInfo($outfile,$phos,$start,$totdev,$totperdev,
                    $rnaatoms,$count,$totatoms,$spec);
            }
            if($debug) { print STDERR "Optimizing\n"; }
            getopt("s:ptnxdulv",\%args);
            if($args{v}) { print STDERR "\nOptimize, version $version\n\n"; die; }
            my $min = $args{in} | 1.2;
            my $max = $args{ix} | 2.4;
            my $incr = $args{is} | 0.05;
            my $debut = $args{id} | 1;
            my $spec = $args{u} | int(rand(900000)+100000);
            my $listfile = $args{l} | "pdbs.txt";
            my $type = $args{t} | "Pdbs";
            my $type1 = "02H0s";
            my $type2 = "02H0p";
            my $path = getPath();

```

```

#min
$min = $times2[4] - $times1[4];
if($min < 0) { $times2[3]--; $min+=60; }

#hour
$hour = $times2[3] - $times1[3];
if($hour < 0) { $times2[2]--; $hour+=24; }

#days - grossly inaccurate
$day = $times2[2] - $times1[2];
if($day < 0) { $times2[1]--; $day+=30; }

#months
$month = $times2[1] - $times1[1];
if($month < 0) { $times2[0]--; $month+=12; }

#year
$month = $times2[0] - $times1[0];
if($debg > 1) { print STDERR "Year,$month,$day,$hour,$min,$sec"; }

return "$year year(s), $month month(s)";

} elsif($month) {
    return "Month month(s), $day day(s)";
} elsif($day) {
    return "$day day(s), $hour hour";
} elsif($hour) {
    return "$hour hour, $min min";
} elsif($min) {
    return "$min min, $sec sec";
} elsif($sec) {
    return "$sec sec";
}
return 0;
}

sub mon2num {
    my ($mon) = @_;
    my %mons = ("Jan", "Feb", "Mar", "Apr", "May", "Jun",
               "Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
    for(my $i=0; $i<= #mons; $i++) {
        if($mon eq $mons[$i]) {
            return $i+1;
        }
    }
    return 0;
}

sub getPdbIDs {
    my @pdbs;
    if($debg) { print STDERR "Reading $listfile for pdbs\n"; }
    open(PDB, "< $listfile") or die;
    while(defined(my $line = <PDB>)) {
        $line =~ s/^(\S+); $line =~ s/\s+$//;
        if($line && length($line) == 4) {
            push(@pdbs,$line);
        }
    }
    if(@pdbs < 0) { die "not enough pdbid in pdbs.txt"; }
    return @pdbs;
}

my $SRESI;
if($debg) { print STDERR "open $outf for reading:\n"; }
open(OUTF, "< $outf") or die "Can't open $outf for reading:\n";
my $minin; my %max; my $rotations = 0;
{
    open(OUTF, "< $outf") or die "Can't open $outf for reading:\n";
    my @keys; my %KEY;
    my %list;
    while(defined(my $line = <OUTF>)) {
        my @chunks = split("\t", $line);
        foreach my $test (@chunks) { $test =~ s/^$s+$//; $test =~ s/^\s+//; }

        sub AnalyzeVol {
            my ($outf) = @_;
            #PASS ONE -- get central limits 5% and 95%
            my $minin; my %max; my $rotations = 0;
            {
                open(OUTF, "< $outf") or die "Can't open $outf for reading:\n";
                my @keys = $chunks[3];
                my $SRESI = $chunks[3];
                my $RNA = $RNA($SRESI) && $chunks[13] eq "\{OK\}";
                $rotations++;
                my $key = gerkey($RESI, $ATOM);
                my $vol = $chunks[10];
                push(@list{$key}, $vol);
                if($isKEY{$key}) {
                    push(@keys, $key);
                    $isKEY{$key} = 1;
                }
            }
            close(OUTF);
            if($debg) { print STDOUT "RES-ATM count min max mincut maxcut median\n"; }
            foreach my $key (@keys) {
                my $sa = $list{$key};
                my $sb = $list{$key};
                my $vol = $chunks[10];
                my $max = $list{$key}[int(0.95*$#list{$key})];
                my $min = $list{$key}[int(0.05*$#list{$key})];
                $max++;
                if($debg) {
                    #print STDOUT "\$key\t",fixnum22($list{$key})+1," ",fixnum22($list{$key})+1," ",fixnum22($list{$key})+1," ",fixnum22($list{$key})+1," ";
                    print "#\t",fixnum22($max{$key}),"\t",fixnum22($list{$key})+1," ",fixnum22($list{$key})+1," ",fixnum22($list{$key})+1," ";
                }
            }
        }
    }
    #close PASS ONE
}

#PASS TWO
open(OUTF, "< $outf") or die "Can't open $outf for reading:\n";
my @keys; my %vol; my %okay; my %volold; my %KEY;
my $numatoms = 0;
while(defined(my $line = <OUTF>)) {
    my @chunks = split("\t", $line);
    foreach my $test (@chunks) { $test =~ s/^$s+$//; $test =~ s/^\s+//; }

    sub AnalyzeVol {
        my ($outf) = @_;
        #PASS ONE -- get central limits 5% and 95%
        my $minin; my %max; my $rotations = 0;
        {
            open(OUTF, "< $outf") or die "Can't open $outf for reading:\n";
            my @keys = $chunks[3];
            my $SRESI = $chunks[3];
            my $RNA = $RNA($SRESI) && $chunks[13] eq "\{OK\}";
            $rotations++;
            my $key = gerkey($RESI, $ATOM);
            my $vol = $chunks[10];
            if($vol > $minin{$key} && $vol < $max{$key}) {
                $numatoms++;
                if($isKEY{$key}) {
                    push(@keys, $key);
                    $isKEY{$key} = 1;
                    $okay{$key} = 1;
                    $volold{$key} = $vol;
                    $volold{$key} = ($vol*1**2);
                } else {
                    $okay{$key}++;
                    $volold{$key} += $vol;
                }
            }
        }
    }
}
```

```

$ovolsq{$key} += ($vol*x2);
}

close(OUTF);

system('rm -f $outf');

my %stiderv; my %mean; my $totdev=0; my $count=0;
foreach my $key (@keys) {
    if($okay{$key} > 10) {
        $count++;
        $stiderv{$key} = sprintf("(%ovolq{$key}-($ovol1{$key}*x2))/($okay{$key}-1));";
        $mean{$key} = $ovol1{$key} / $okay{$key};
        $stiderv += $stdev{$key};
        $totdev += $stdev{$key} / $mean{$key};
        $totperdev += $stdev{$key} / $mean{$key};
        #print STDERR "$key\n";
        #print fixnum32($stiderv),"\n";
    }
    print STDERR "$key\n";
}
return ($totdev,$totperdev,$numatoms,$count,$totatoms);
};

print OUTF " ";
print (STDERR,"$totdev,$totperdev,$numatoms,$count,$totatoms);

#####
#####

sub printOutInfo {
    my ($outfile, $phos, $start, $totdev, $totperdev,
        $numatoms, $count, $totatoms, $phos2) = @_;
    open(OUT, '> $outfile') or die "Couldn't open $outfile for writing: $!\n";
    print OUT fixnum12($phos), " ", fixnum12($totperdev), "\n";
    print OUT fixnum12($phos2), " ", fixnum37($totperdev), "\n";
    print OUT fixnum12($phos), " ", fixnum37($totperdev), "\n";
    print OUT "det " . fixnum12($phos), " ", fixnum30($count);
    print OUT "tot dev " . fixnum35($totdev), " ", fixnum25($totperdev), " ", tot%dev " ;
    print OUT fixnum25($totperdev), "%", "\%\n";
    print OUT "time: " . substr($start, getime0), "\n";
    print OUT "t", "fixnum25($totperdev/$count), "\n";
    print OUT "t", "fixnum32($numatoms/$totatoms), "\n";
    print OUT "xx $type1 xx $type2\n";
    close(OUT);
};

#####
#####

sub getKey {
    my ($res,$atm) = @_;
    if($atm eq "p" || $atm eq "OIP" || $atm eq "02P") {
        return "SUG-".$atm;
    } elsif(length($atm) == 3 && substr($atm,2,1) eq "\") {
        return "SUG-".$atm;
    }
    return $res."-".$atm;
};

#####
#####

sub writeTime {
    ($outfile) = @_;
    open(OUT, '> $outfile') or die "Couldn't open $outfile for reading: $!\n";
    print OUT gettime(), "\n";
    close(OUT);
};

#####
#####

sub editAtoms {
    my ($atomdef, $phos, $phos2) = @_;
    open(ATOM, "< $atomdef") or die "Couldn't open $atomdef for reading: $!\n";
    my $types = $atomdef;
    substr($types,3) = "temp.$spec";
    open(TYPES, "> $types");
    while(defined(my $line = <ATOM>)) {
        my @chunks = split( ' ', $line);
        if($chunks[0] eq $type1) {
            print TYPES "$type1 ", fixnum12($phos), " $bondrad\n";
        } elsif($chunks[0] eq $type2) {
            print TYPES "$type2 ", fixnum12($phos2), " $bondrad\n";
        } else {
            print TYPES "$line";
        }
    }
    close(ATOM);
    close(TYPES);
    return $types;
};

#####
#####

```

```

#!/usr/bin/perl -w
use Getopt::Std;
use strict;
require "ctime.pl";
require '/home/vossman/lib/fixnum.pl';
my %args;
getopt("ioart", \%args);
#GLOBAL HASHES
my %atomrad;
my %kistype;
my %typefor;
my %typebad;
my %typeExist;
my $globaltype = "rna";
my $globaltype = "prot";
my $out2 = $args{o} || './atom-vols-$globaltype-$time.dat';
my $out2 = substr($out2,-4) = "-MORE.dat";
main();
#END
sub main {
    #OPEN FILES
    open(OUT, "> $out2")
    or die "Couldn't open $out2 for writing: $!\n";
    #open(OUT2, "> $out2")
    # or die "Couldn't open $out2 for writing: $!\n";
    print "Writing to $out2...\n";
    my $header = "# NucProt type set of Atom Volumes\n# Generated: " .
        " ("ctime)." . "# Input file $in#\n";
    $header .= "# Output file $out#\n";
    $header .= "# Atom file $atom#\n";
    $header .= "Type \"$globaltype\"\n# $atom#\n";
    $header .= "Mean $atomrad\n";
    $header .= "StdDev $atomrad\n";
    $header .= "Min $atomrad\n";
    $header .= "Max $atomrad\n";
    print OUT $header;
    #print OUT2 $header;
    my @atomdefs = parseAtom($atom);
    @atomdefs = sort { $a cmp $b } @atomdefs;
    print STDERR "done parsing atom file (" . $#atomdefs + 1 .
        " types found)\n\n";
}

```

```

#!/usr/bin/perl -w
my @res = parseResi($resi);
#print STDERR "done parsing resi file (" . $#res + 1 . " found)\n";
my $i = -1;
my $count = $found.$notresi.$nottype.$notgood = (0,0,0,0);
my $total = numLines($in);
my $carrot = $total/50;
open(IN, "< $in");
or die "Couldn't open $in for reading: $!\n";
print STDERR "Parsing in file ($total lines)\n";
while(defined(my $line = <IN>)) {
    my @chunks = split("\t", $line);
    foreach my $test (@chunks) { $test =~ s/^\s+//; $test =~ s/\s+$//; }
    #CHECK IF VALIDITY
    if($isResi{@chunks[6]}) { $nottype++; }
    elsif($isResi{@chunks[3]}) { $notresi++; }
    #MODIFY FOR PROTEIN
    elsif($isResi{@chunks[3]} & $isTypef{@chunks[6]}) {
        $typeExist{@chunks[6]}=1;
        if(@chunks[13] eq "OK") {
            push(@{$chunks[6]}, $chunks[10]);
            if($typeOK{@chunks[6]}) { $typeOK{@chunks[6]}=0; }
            $typeOK{@chunks[6]}++;
            $found++;
        } elsif($chunks[13] eq "possible") {
            if($typePoss{@chunks[6]}) { $typePoss{@chunks[6]}=0; }
            $typePoss{@chunks[6]}++;
            $found++;
        } elsif($chunks[13] eq "bad") {
            if($typeBad{@chunks[6]}) { $typeBad{@chunks[6]}=0; }
            $typeBad{@chunks[6]}++;
            $found++;
        } else { $notgood++; }
        if($count > $carrot) { print STDERR "\n"; $carrot += ($total/50); }
        $count++;
    }
    if($count > $carrot) { print STDERR "\n"; $carrot += ($total/50); }
    close(IN);
    print STDERR "\nFound $total parsed $count\n";
    # print STDERR "\n";
    print STDERR "\n";
    #print STDERR $header;
    #print STDERR $header;
    foreach my $type (@atomdefs) {
        my $line;
        my $perc = fixnum21(100*$typeOK{$type}/$typeOK{$type}+$typePos{$type}+$typeBad{$type});
        $line = "$type\t".fixnum50($typeOK{$type}) ."\t" .$perc ."\t";
        my ($mean,$median,$stdev,$min,$max,$mode,$skew,$kurt) = analyzeVol($type);
        $line .= fixnum22($atomrad{$type}) ."\t" .fixnum32($median) ."\t" .fixnum22($mean) ."\t" .fixnum22($stdev) ."\t" .fixnum22($min) ."\t" .fixnum22($max) ."\t" ;
        print OUT $line;
        my (@atoms) = parseResi($resi,$type);
        my (@atoms) = 0;
        $line := fixnum30($atoms);
        #print STDERR $line " atoms printed to file\n";
        foreach my $atom (@atoms) { $line .= "$atom;" }
        substr($line,-1) = "\n";
        $line .= "\n";
        print OUT $line;
    }
}
close(OUT);
#close(OUT2);
}; # end main()

```

```

sub isResi {
    my ($res) = @_;
    if($globaltype == "prot") {
        return isProt($res);
    } elsif($globaltype == "rna") {
        return ISRNA($res);
    }
    return 0;
}

sub isRNA {
    my ($res) = @_;
    my @RNA = ("A", "C", "U", "G", "ADE", "CYT", "URI", "GUA");
    foreach my $base (@RNA) {
        if($res eq $base) {
            return 1;
        }
    }
    return 0;
}

sub isProt {
    my ($res) = @_;
    my @PROT = ("GLY", "ALA", "MET", "GLUT", "ASP", "TRP", "THR", "TYR",
                "ILE", "LEU", "VAL", "CYS", "PRO", "ARG", "SER", "LYS", "PHE", "HIS",
                "CHI", "CSS");
    foreach my $aa (@PROT) {
        if($res eq $aa) {
            return 1;
        }
    }
    return 0;
}

sub mode {
    my (@list) = @_;
    my $modemax;
    my $p = 1.689700434; # precision
    my @evals;
    foreach my $vol (@list) {
        my $v = int($vol * (10**$p)+0.5);
        if($modemax < $modemax) { $modemax = $v; }
        else { push(@evals,$v); $modemax=1; }
    }
    my ($mode,$modemax) = (@_,@evals);
    my @crash;
    my $range = 50;
    foreach my $v (@evals) {
        my $avemode=0;
        for(my $k=$range;$k<=$range;$k++) {
            my $v2 = int((($v - $k*$p)*1000 + $p)/1000);
            if($modemax{$v2}) {
                my $avemode = $modemax{$v2};
                my $factor = exp(-0.05*($k**2));
                $factor = int($factor*1000)/1000;
                $avemode += $modemax{$v2}*$factor;
            }
        }
        if($avemode) { #???
            if($avemode > $modemax) {
                $mode = $v;
                $modemax = $avemode;
                @crash = ($v);
            } elsif($avemode == $modemax) {
                push(@crash,$v);
            }
        }
    }
    if($#crash > 0) {
        print STDERR "Crash on Mode \[$modemax\]: ";
        foreach my $v (@crash) {
            print STDERR fixnum($v,2,int(-1*log($p)/log(10)+0.95),",",".");
        }
        print STDERR "\n";
        return $mode;
    }
}

sub parseResi {
    my ($resi,$type) = @_;
    my $list;
    open(RESI, "< $resi");
    for(my $i=0; $i<$num; $i++) {
        $str.= "=";
    }
    return $str;
}

if($#chunks>0) {
    my $res;
    while(defined(my $line = <RESI>)) {
        my @chunks = split(" ", $line);
        if(@chunks) {
            if($chunks[1]) {
                $res=$chunks[0];
            }
            $i=$chunks[1];
        }
    }
}

```

```

} elsif($res) {
    if($chunks[0] && $chunks[1] eq $type) {
        push(@list,$res." ".$chunks[0]);
        $i--;
    }
    close(RSI);
    return @list;
}

sub parseAtom {
    (@atom) = @_;
    my @atomdefs;
    open(ATOM, "< $atom");
    or die "Couldn't open $atom for reading: $!\n";
    while(defined(my $line = <ATOM>)) {
        if(length($line) > 3) {
            my @chunks = split(' ', $line);
            if(@#chunks > 1) {
                push(@atomdefs, $chunks[0]);
                $listtype{$chunks[0]}=1;
                $atomrad{$chunks[0]}=$chunks[1];
            } else {
                print STDERR "E";
            }
        }
        close(ATOM);
        return @atomdefs;
    }
}

sub numLines {
    my ($in) = @_;
    open(IN, "< $in")
    or die "Couldn't open $in for reading: $!\n";
    my $count=0;
    while(defined(my $line=<IN>)) {
        $count++;
    }
    close(IN);
    return $count;
}

```

```

sub analyzeVol {
    my ($type, @vol) = @_;
    @vol = sort { $a <= $b } @vol;
    my $cut = int(0.10*$typeeqK($type));
    my ($min, $max) = ($vol[$cut], $vol[$#vol-$cut]);
    my ($mean, $stdev) = meanstdev(@vol);
    my $median = median(@vol);
    my $mode = mode2(@vol);
    my $skew = skew($mean, $stdev, @vol);
    my $kurt = kurtosis($mean, $stdev, $min, $max, $mode, $skew, $kurt);
    return ($mean, $median, $stdev, $min, $max, $mode, $skew, $kurt);
}

sub meanstdev {
    my (@nums) = @_;
    my ($sum, $sumsq) = (0,0);
    foreach my $num (@nums) {
        $sum += $num;
        $sumsq += ($num**2);
    }
    my $mean = ($sum/($#nums+1));
    my $stdev = sqrt(($sumsq-$sum**2/($#nums+1))/$#nums);
    return ($mean, $stdev);
}

```

```

sub median {
    #assume sorted
    my ($vol) = @_;
    my $median;
    if($#vol%2 == 0) {
        $median = ($vol[int($#vol)/2] + $vol[int($#vol)/2+1])/2;
    } else {
        $median = $vol[int($#vol+1)/2];
    }

    sub skew {
        my ($mean, $stdev, @list) = @_;
        my $skew;
        foreach my $v (@list) {
            $skew += ($v-$mean)**3 / (@list + 1);
        }
        $skew /= $stdev**3;
        return $skew;
    }

    sub kurtosis {
        my ($mean, $stdev, @list) = @_;
        my $skurt;
        foreach my $v (@list) {
            $skurt += ($v-$mean)**4 / (@list + 1);
        }
        $skurt = ($skurt/$stdev**4)-3;
        return $skurt;
    }
}

```

```

#!/usr/bin/perl -w
use strict;
use Getopt::Std;
require "ctime.pl";
require '/home/yossman/lib/fxnum.pl';

#####
my %args;
getopt("jor", \%args);
my %isRNA = ('CYT' ,1,'URI' ,1,'GUA' ,1,'ADE' ,1);
my %isATOM;
my %isSUG;
my %atomVol;
my $times = split(' ', &ctime(time));
my $time = $times[1] . '-' . $times[2] . '-' . $times[4];
#####
my @args = ($args{file} || ".*.out");
my $resi = $args[r] || ".../spec-defs.dat"; #designed for resi-defs.dat
my $out = $args{o} || "base-sizes.$time.dat";
open(OUT, ">$out")
or die "Couldn't open $out for writing: $!\n";
print OUT "# NucProt volume set of base sizes\n# Generated: ", &ctime(time), "# Resi file $resi#\n";
#####
my $res = parseResiFile($resi);

#Open File
my $res = parseResiFile($resi);

print OUT "file\VADE\TCYT\тур1\тGIA\тSUG\тADE2\тCYT2\тUR12\тGIA2\n";
foreach my $inf (@in) {
    cleanAtomVol(@res);
    parseOutFile($inf);
    print STDERR "Calculating Statistics...\n";
    my %mean;
    foreach my $key (keys (%mean)) {
        my ($r,$a) = split(' ', $key);
        my ($m,$sdev) = meanstddev(@{$atomVol{$key}});
        $mean{$r,21+=}$m;
        $mean{$r,22+=}$m;
        if($r eq "SUG") {
            foreach my $r2 ("ADE", "CYT", "GIA", "UR1") {
                my $key2 = $r2 . " " . $sdev;
                ($m,$sdev) = meanstddev(@{$atomVol{$key2}});
            }
        }
    }
    my $inf2 = $inf;
    $inf2 =~ s/-edit.out$/;/;
    $inf2 =~ s/-d\d\d\d\d\d\d\d\d/;/;
    $inf2 =~ s/-voronoi//;
    print OUT "$inf2" fxnum32($mean{"ADE"}), "\t",
    fxnum32($mean{"CYT"}), "\t",
    fxnum32($mean{"GIA"}), "\t",
    fxnum32($mean{"UR1"}), "\t",
    fxnum32($mean{"SUG"}), "\t",
    fxnum32($mean{"AD22"}), "\t",
    fxnum32($mean{"CYT2"}), "\t",
    fxnum32($mean{"UR12"}), "\t",
}
my $vol = $chunks[10];
#####

```

```

fxnum32($mean{"GU2"}), "\n";
}
my %resi;
my $resi = $chunks[3];
my $status = $chunks[4];
my $key = $chunks[3] . "_" . $chunks[4];
if($status eq "OK") {
    my $vol = $chunks[10];
    #####
    sub parseResiFile {
        #opens resi file
        #returns list of residues and atoms in form of resi-atom
        #allow fills global hash %isatom
        my ($resi,
            open(RESI, "< $resifile")
            or die "Couldn't open $resi for reading: $!\n";
        my $i=0; my %curresi;
        while(defined(my $line = <RESI>)) {
            if($line =~ substr($line,0,1) && !$i) {
                $line =~ s/(S\S*)//; $line =~ s/^\s+//;
                my @chunks = split(' ', $line);
                foreach my $test (@chunks) { $test =~ s/\s+$/;/; $test =~ s/^\s+//; }
                if($chunks[1]) {
                    $curresi = $chunks[0];
                    $i=$chunks[1];
                }
            } elsif($curresi && $line =~ substr($line,2,1)) {
                $line =~ s/\s+$/;/;
                my @chunks = split(' ', $line);
                my $test = $chunks[0];
                $test =~ s/\s+$/;/;
                push(@res,$curresi . "_" . $test);
                if($curresi eq "SUG") {
                    $isatom{"ADE"} . $test = 1;
                    $isatom{"CYT"} . $test = 1;
                    $isatom{"GUA"} . $test = 1;
                    $isatom{"UR1"} . $test = 1;
                    $isatot{$URL} . $test = 1;
                    $isSUG{$test}=1;
                }
                $isAtom{$curresi."_" . $test} = 1;
                $i--;
            }
        }
        close(RESI);
        return @res;
    };
#####
sub parseOutFile {
    my ($in) = @_;
    my $total = getNumLines($in);
    print STDERR "(reading the file ($total) $in\n";
    my $count; my $carrot=0;
    printBar(76);
    open(IN, "< $in") or die;
    while(defined(my $line = <IN>)) {
        $count++;
        my @chunks = split("\t", $line);
        foreach my $test (@chunks) {
            $test =~ s/\s+$/;/;
        }
    }
    my $atom = $chunks[4]; my $resi = $chunks[3];
    my $status = $chunks[5];
    my $key = $chunks[3] . "_" . $chunks[4];
    if($status eq "OK") {
        my $vol = $chunks[10];
        #####
    }
}

```

```

if($isSUG{$atom} && $isRNA{$resi}) {
    my $key2 = "SUG";
    push(@{$atomVol{$key}}, $vol);
    push(@{$atomVol{$key}}, $vol);
}
if($count > $carrot) {
    print STDERR "<"; 
    $carrot = $carrot + $total / 76;
}
print STDERR "\n";
close(IN);
return;
};

#####
sub meanstddev {
    my (@nums) = @_;
    if (@nums < 1) { return 0; }
    my ($sum,$sumsq) = (0,0);
    foreach my $num (@nums) {
        $sum += $num;
        $sumsq += ($num**2);
    }
    my $mean = ($sum/($#nums+1));
    my $stdev = sqrt((($sumsq-$sum**2)/($#nums+1))/$#nums);
    return ($mean,$stdev);
}

#####
sub cleanAtomVol {
    #clean out %atomVol;
    my (%res) = @_;
    foreach my $key (@res) {
        my ($r, $a) = split(' ', $key);
        @{$atomVol{$key}} = 0;
        if($r eq "SUG") {
            foreach my $r2 ("ADE", "CYS", "GHA", "UR1") {
                my $key2 = $r2." ".$a;
                @{$atomVol{$key2}} = 0;
            }
        }
    }
}

#####
sub getNumLines {
    my ($in) = @_;
    my @chunks = split(' ', $in);
    open(IN, '< temp.txt') or die;
    my $num = <IN>;
    close(IN);
    my @chunks = split(' ', $num);
    $num = $chunks[0];
    $num =~ s/^\s+//; $num =~ s/\s+$//;
    return $num;
}

#####
sub printBar {
    my ($num) = @_;
    my $mult=1;
    while($num/$mult > 80) {
        $mult++;
        $num = int($num/$mult+0.5);
        print STDERR "[";
        for(my $i=1;$i<$num-2;$i++) {
            if($i==int(($num-2)/2+0.5)) { print STDERR "\n"; }
            elsif($i==int(3*$num-2)/4+0.501) { print STDERR "#"; }
            elsif($i==int(3*$num-2)/4+0.501) { print STDERR "\#"; }
            elsif($i%5==0) { print STDERR "+"; }
            else { print STDERR "-"; }
        }
        if($i%100==0) { print STDERR "\n"; }
        print STDERR "\n";
    }
    return $mult;
};

#####
sub clean {
    my ($str) = @_;
    $str =~ s/^\s+$/; $str =~ s/\s+$//;
    return $str;
};

#####
sub list2vec {
    my (@list) = @_;
    my $vec;
    foreach my $item (@list) {
        $vec .= $item." ";
    }
    return $vec;
};

#####
sub vec2list {
    my ($vec) = @_;
    my @vec;
    return split(' ', $vec);
};

#####
sub vec2pdb {
    #HETATM91176 0 HOH 496
    my ($vec, $count, $num, $resnum) = @_;
    my ($x, $y, $z, $r) = vec2list($vec);
    my $line = "HETATM";
    $line .= fixnum30($count);
    $line .= "O HO ";
    $line .= fixnum40($resnum);
    $line .= fixnum43($x+0.012);
    $line .= fixnum43($y+0.034);
    $line .= fixnum43($z+0.056);
    $line .= "1.00";
    if($num) {
        $line .= fixnum32($num);
    } else {
        $line .= fixnum32(0.0);
    }
    $line .= "\n";
    return $line;
};

#####

```

```

#!/usr/bin/perl
require '/home/vossman/lib/filetools.pl';
require '/home/vossman/lib/fixnum.pl';
require 'ctime.pl';
use Text::Tabs;
use Getopt::Std;
use constant FACTOR => 3.4519378473;
print "# HISTOGRAM MAKER, hist.pl v 0.9 (3/10/04)\n";
# creates histograms from packing out file
# requires:
# - one of atom name, atom type or residue specified
# - input packing out file
# outputs:
# - (x,y) data for histogram
# - input packing out file
# - output hst file (volume <tab> number of atoms)
# - Number of points on x-axis
# -H Number of points on displayed y-axis (does not affect output file)
# -r residue restriction (e.g. AIE)
# -a atom name restriction (e.g. N1)
# -t atom type restriction (e.g. NH1)
# -n minimum volumes value
# -N maximum volumes value
# -Y averaging factor (e.g. 1 means average +/- 1 point; default: 0)
# -b boxplot file name
# -R residue type RNA or PROT
# change log:
# v 0.8 add boxplot_info %ile,25%ile,75%ile,95%ile
# v 0.7 classic
my %args;
getopt("soatrhoxybR", \%args);
my %args;
getopt("soatrhoxybR", \%args);

#Set input variables
my %isRNA = ("ADE" ,1,"GUU",1,"CYT",1,"URU",1);
$residue_type = $args{R} || "rna";
#$residue_type = $args{R} || "prot";
if($residue eq "rna") {
    $in = $args{i} || "/home/vossman/packing/out/rnaonly_all_trun.out22222";
} elsif($residue eq "prot") {
    $in = $args{i} || "/home/vossman/packing/out/protein-edit.out";
} else {
    $in = $args{i};
}
$boxplot = $args{b} || "./boxplot.dat";
$type = $args{t}; # leave blank for unrestricted
$atom = $args{o}; # leave blank for unrestricted
$resi = $args{r}; # leave blank for unrestricted
$hist = $args{h};

if($args{fo}) {
    $out = $args{fo};
} else {
    my $info;
    foreach my $item ($resi,$atom,$type) {
        if($item =~ $info) {
            $out = $info;
            substr($out,-4) = "-$info.hst";
        }
    }
}
$info =~ s/-$/-/;

$out = $in;
substr($out,-4) = "-$info.hst";

$incr = fixnum52(($max - $min) / ($POINTS-$2));
if($defmin) { $min -= $incr*1.5; }

$POINTS = $args{P} || 50;
$HEIGHT = $args{H} || 20;
$defmin = $args{n} || 0;
$defmax = $args{xf} || 0;
$defincr = $args{s} || 0;
$aver = $args{y} || 0;

#Initialize Variables
$max = 0;
$min = 1500;
$numatoms = 0;
$vol = 0;
$volsq = 0;

my $numlines = getNumLines($in);
open(IN, "< $in");
my $count=0; my $carrot=$numlines/76;
print "$Reading $in...\n";
print "Reading $in...\n";
print "Counting $in...\n";
print "Carrot=$carrot\n";
print "Count=$count\n";
while( defined($line = <IN>) ) {
    $count++;
    if($count > $carrot) { print STDERR "^\n"; $carrot+=getNumLines/76; }
    #Read Variables
    @chunks = split("\t", $line);
    foreach $test (@chunks) { $test =~ s/\$/\$//; $test =~ s/\$/\$//; }
    if($chunks[13] eq "\\\OK\\\\") {
        if($type) {
            $TYPE = $chunks[6];
        }
        if($atom) {
            $ATOM = $chunks[4];
        }
        if($resi) {
            $RESI = $chunks[3];
        }
        if((!$type || $TYPE =~ $type) && (!$atom || $ATOM eq $atom) && (!$resi || $RESI eq "SUG" && $isRNA{$RESI})) {
            $numatoms++;
            push(@hist, $chunks[10]);
            if($chunks[10] > $max) { $max = $chunks[10]; }
            if($chunks[10] < $min) { $min = $chunks[10]; }
            $vol += $chunks[10];
            $volsq += $chunks[10]**$chunks[10];
        }
    }
}
close(IN);
print "\nFound $numatoms 'OK'" atoms...\\nContinuing with histogram info...\n";
open(OUT, "> $out");
my $time = `date`;
if($defmin) { $min=$defmin; }
if($defmax) { $max=$defmax; }
if($defincr) { $incr=$defincr; }
if($defmin) { $min = int((($max-$min)/$incr)+2); }
$max = $incr*$POINTS+$min;
} else {
    $incr = fixnum52(($max - $min) / ($POINTS-$2));
    if($defmin) { $min -= $incr*1.5; }

    $POINTS = $args{P} || 50;
    $HEIGHT = $args{H} || 20;
    $defmin = $args{n} || 0;
    $defmax = $args{xf} || 0;
    $defincr = $args{s} || 0;
    $aver = $args{y} || 0;
}

#if($type && !$atom && !$resi) {

```

```

}
print STDERR "max=$max\tmin=$min\n";
for($i=0; $i <= $POINTS; $i++) {
    push(@counts,0);
}

foreach $item (@hist) {
    $i = floor( ($item - $min)/$incr );
    $counts[$i]++;
}

#Output to file
if($aver) {
    for($i=0; $i < $POINTS; $i++) {
        printf OUT "%f\t%f\n", $min+$i*$incr,
    }
    else {
        for($i=0; $i < $POINTS; $i++) {
            for($j=($i-$aver);$j<=(($i+$aver);$j++) {
                $val+=$counts[$j];
            }
            if($val) {
                $val=($aver*$val);
                printf OUT "%f\t%f\n", $min+$i*$incr,$val;
            }
        }
    }
}

#Draw Graph
$maxcnt = 0;
$#counts = $POINTS;
foreach $num (@counts) {
    if($num > $maxcnt) { $maxcnt = $num; }
}

$incr2 = $maxcnt / $HEIGHT;
print "\n";
print "Atoms\t";
if($atom) { print "ATOM=$atom\n"; }
if($type) { print "TYPE=$type\n"; }
if($resi) { print "RESI=$resi\n"; }
if($hist) { print "HIST=$hist\n"; }
$temp = $HEIGHT*$incr2;
print "\n";
for($i=0; $i<=$POINTS-1; $i++) { print " "; }
for($i=$HEIGHT; $i>0; $i--) { print " "; }
print "\n";
foreach $num (@counts) {
    if($num != 0 && $num/$incr2 >= $i) {
        if($num/$incr2 <= $i+0.15 || ($num - $i*$incr2) < 1.5) { print "#"; }
        elsif($num/$incr2 <= $i+0.5) { print "*"; }
        else { print "#"; }
    } else { print " "; }
}
print "\n";
print "Atoms\t";
for($i=0; $i<=$POINTS-1; $i++) { print " "; }
for($i=0; $i<=$POINTS-1; $i++) {
    print "\n";
    for($j=0; $j<= $i; $j++) {
        print "\n";
        for($k=0; $k<=$j; $k++) {
            print "#";
        }
        print "\n";
    }
}
print "\n";
if($numatoms != 0) {
    if($hist) {
        printf("\n\t%.1f", $min);
    }
}

```

```

#!/usr/bin/perl -w
use Getopt::Std;
use Text::Tabs;
require "time.pl";
require "/home/vossman/lib/fixnum.pl";
getopt("ior", \%args);
$iresis{CYT}=L;
$iresis{URU}=L;
$iresis{GUA}=L;
$iresis{ADE}=L;
@times = split( ' ', &ctime(time));
$time = $times[1] . ". " . $times[2] . ". " . $times[4];
$Set input variables
$resi = $args{r} || "in.out";
$resi = $args{r} || ".../spec-defs.dat"; #designed for resi-defs.dat
if($args{o}) {
    $out = $args{o};
    open(OUT, ">$out");
    or die "Couldn't open $out for writing: $!\n";
} else {
    $out = "/dev/null";
    open(OUT, ">$out");
    or die "Couldn't open $out for writing: $!\n";
}
print "Writing to $out...\n";
#$header = " atom\type\tcount\t%OK\tmode\tmean\tsd\t%SD\tmin\tmax\n";
$header = "#$header";
$header2 = "#$header2";
print OUT "# NucProt type set of 258 Volumes\n# Generated: ".&ctime(time)
,"# Input file $resi file $resi.m\n";
print OUT " atom Count Mean volume SD SD() Minimum Maximum\n";
#Open Files
open(RESI, "<$resi");
or die "Couldn't open $resi for reading: $!\n";
$i=0;
while(defined($line = <RESI>)) {
    if($line && substr($line,0,1) && !$i) {
        $line =~ s/\s+$/ ; $line =~ s/\s+/;/;
        @chunks = split( ' ', $line);
        foreach $test (@chunks) { $test =~ s/\s+$/ ; $test =~ s/\s+//; }
        $i--;
    }
    elsif($curressi && $line && substr($line,2,1)) {
        $line =~ s/\s+$/ ; $line =~ s/\s+/;/;
        @chunks = split( ' ', $line);
        foreach $test (@chunks) { $test =~ s/\s+$/ ; $test =~ s/\s+//; }
        push(@{$curressi}, $chunks[0]);
        $i--;
    }
}
close(RESI);
foreach $RESI (@resi) {
    $tvol = 0;
    $tvolsq = 0;
    $okay = 0;
    $tposs = 0;
    $tbad = 0;
    if($out ne "stdout") { print STDERR "\n$RESI\t[ " ; }
    if($out ne "stdout") { print $tvol, " " ; }
    if($out ne "stdout") { print $tvolsq, " " ; }
    if($out ne "stdout") { print $okay, " " ; }
    if($out ne "stdout") { print $tposs, " " ; }
    if($out ne "stdout") { print $tbad, " " ; }
    if($out ne "stdout") { print "\n" ; }
}

```

```

$ok = -1;
print OUT "RESI\n" ; $header; $header2;
foreach $ATOM (@{$RESI}) {
    $k++;
    if($out ne "stdout") { print STDERR "$ATOM " ; }
    $okay = 0; $tposs=0; $tbad=0;
    $tvolsq = 0;
    $maxvol = 0;
    $minvol = 1500;
    open(IN, "<$in");
    or die "Couldn't open $in for reading: $!\n";
    while( defined($line = <IN>) ) {
        @chunks = split("\t", $line);
        foreach $test (@chunks) { $test =~ s/\s+$/ ; $test =~ s/\s+$/ ; $test =~ s/\s+$/ ; }
        if($ATOM eq $chunks[4] && ($RESI eq "SUG" && $iresis{$chunks[3]})) {
            if($chunks[13] eq "\OK") {
                $okay++;
                $vol = $chunks[10];
                $vol += $vol;
                $tvolsq = $tvolsq + $vol*$vol;
                if($vol > $maxvol) { $maxvol = $vol; }
                if($vol < $minvol) { $minvol = $vol; }
            }
            elsif($chunks[13] eq "possible") { $poss++; }
            elsif($chunks[13] eq "bad") { $bad++; }
        }
    }
    close(IN);
    $tvol += $okay;
    $tbad += $bad;
    $tposs += $poss;
    if($okay > 1) {
        $tvolsq = sqrt( ($tvolsq - $vol*$vol/$okay) / ($okay - 1) );
        $vol = $vol / $okay;
        $tvol += $vol;
        $tvolsq += $tvolsq*$tvolsq;
    }
    elsif ($okay == 1) {
        $tvolsq = 0;
    }
    if($okay > 1) {
        #FIND MODE
        $line = "$ATOM\t".fixnum50($okay)." " .
fixnum23($vol)." ".fixnum13($tvolsq)." " .
fixnum23($tvolsq*$vol*100)." ".fixnum23($maxvol)." ".fixnum23($minvol)." ".fixnum23($tvolsq)." " .
expand($line);
        print OUT $line;
    } else {
        print STDERR "\be ";
    }
}
if($out ne 'stdout') { print STDERR " ]" ; }
if($tvolsq > 1) {
    $tvolsq = sqrt($tvolsq);
    print OUT "\n Total ".fixnum50($tvolsq)." " .
fixnum33($tvol)." ".fixnum23($tvolsq)." " .
fixnum23($tvolsq/$tvol*100), "\n\n";
}
print STDERR "\n\nIf an '\e' follows the character; no \[OK\] atoms were found.\n\n";

```

```

#!/usr/bin/perl -w
use Getopt::Std;
require "ctime.p1";
require "/home/voszman/lib/fxnum.p1";
print STDERR "\n";
$gap=0.01;
$range=50;
getopt("ior", \%args);
@sresi{CIT}=>L;
@sresi{URI}=>L;
@sresi{GUA}=>L;
@sresi{ADE}=>L;
@sresi{SUG}=>L;

@times = split(' ', &ctime(time));
$time = $times[2].$times[1].fxnum20b($times[4]*100);

#Set input variables
$in = $args{fi} || ".in.out";
$resi = $args{r} || ".../spec-defs.dat"; #designed for resi-defs.dat
if($args{o}) {
    $out = $args{o};
} else {
    $out = "/stdvol258-ext.$time.dat";
}
if(stat($out)) {
    print STDERR "deleting $out...\n";
    sleep(5);
    system("rm -f $out");
    print STDERR "done\n\n";
}

#print OUT "<P><FONT SIZE=-2>Page generated on '$time'</FONT>\n";
$header = "atom\ttype\tcount\tv\tOK\tmode\tmean\t sd\tmin\tmax\n";
$header2 = "-----\t-----\t-----\t-----\t-----\t-----\t-----\n";
#Open File
open(RESI, "< $resi")
    or die "couldn't open $resi for reading: $!\n";
my %issug;
print STDERR "reading $resi file\n";
my $curresi; my $i=0;
while(defined(my $line = <RESI>)) {
    if($line && $line =~ /[A-Za-z]+/\$i) {
        $line =~ s/\$/\$//; $line =~ s/\$/\$+/;
        @chunks = split(' ', $line);
        foreach $test (@chunks) { $test =~ s/\$/\$//; $test =~ s/\$/\$+/; }
        push(@resi,$chunks[0]);
        print STDERR $chunks[0];
        $curresi = $chunks[1];
    }
    elsif($curresi && $line && $line =~ /\A-Za-z]+\+/) {
        $line =~ s/\$/\$//; $line =~ s/\$/\$+/;
        @chunks = split(' ', $line);
        foreach $test (@chunks) { $test =~ s/\$/\$//; $test =~ s/\$/\$+/; }
    }
}
foreach $test (@chunks) { $test =~ s/\$/\$//; $test =~ s/\$/\$+/; }

#Open OUT
open(OUT,">> $out")
    or die "couldn't open $out for writing: $!\n";
my OUT;
# NucProt type set of 258 Volumes\n# Generated: "&ctime(time)
# Input file $in#\ Resi file $resi\n#\n";
close(OUT);

print STDERR "writing to $out...\n";
foreach $RESI (@resi) {
    open(OUT,">> $out")
        or die "couldn't open $out for writing: $!\n";
    $two1 = 0;
    $two2 = 0;
    $tokay = 0; $tposs = 0; $tbad = 0;
    $modvol=0; $minvol=0; $maxvol=0;
    if($out ne "stdout") { print SIDER "\n$RESI\t[ \" ; } ;
    $k = -1;
    print OUT "$RESI\n", $header, $header2;
    foreach $ATOM (@$RESI) {
        foreach $ATOM (@$RESI) { }
    }
}

```

```

$ok++;
if($out ne 'stdout') { print STDERR "$ATOM " ; }

$okay = 0; $possp=0; $bad=0;
$ovol = 0;
$ovolsq = 0;
$maxvol = 0;
$minvol = 1500;

open(IN, '< $in');
# or die "Couldn't open $in for reading: $!\n";
while(defined($line = <IN>)) {
foreach my $line (@$RESI) {
    @chunks = split("\t", $line);
    foreach $test (@chunks) { $test =~ s/^\$/\$//; $test =~ s/^\$/\$//; }
    if($ATOM eq $chunks[4] && ($$RESI eq "$UG" && $$resi{$chunks[3]})) {
        if($chunks[13] ne "bad") {
            if($chunks[13] eq "\OKY") {
                $okay++;
                $vol = $chunks[10];
                $vol += $vol;
                $volt = int($vol/$gap)*$vol*$gap*10000+$gap/10)/10000;
                $modevol++; $volt++;
                $ovolsq = $ovolsq + $vol*$vol;
                if($vol > $maxvol) { $maxvol = $vol; }
                if($vol < $minvol) { $minvol = $vol; }
            }
            elsif($chunks[13] eq "possible") { $possp++; }
            elsif($chunks[13] eq "bad") { $bad++; }
        }
    }
}

#close(IN);
$okay += $okay;
$tbad += $bad;
$tposs += $possp;

if($okay > 1) {
    $ovolsq = sqrt( ($ovolsq - $ovol*$ovol/$okay) / ($okay - 1) );
    $vol += $ovol;
    $ovolsq += $ovolsq*$ovolsq;
}
elsif ($okay == 1) {
    $ovolsq = 0;
}

#FIND MODE
$modeemax=0;
for($l=$gap*($grange+1); $l<45; $l+= $gap) {
    $avgnode = 0;
    $l=int($l*1000+$gap/100)/1000;
    for($k=$grange; $k<=$range; $k++) {
        $l2 = int(($l - $k*$gap)*1000 + $gap/100)/1000;
        if($mode{$l2}) {
            if(abs($l2 - $l) > 0.3) {
                $factor = int(2**$gap/abs($l-$l2)*1000+$gap/100)/1000;
                $factor = int(exp(-5*(abs($l-$l2)-0.1)**2)*1000+$gap/100)/1000
            } else { $factor = 1; }
            $avgnode += $mode{$l2}*$factor;
        }
    }
    if($avgnode > $modeemax) {
        $modeemax=$avgnode;
        $modevol = $l;
    }
    $l2=$l-$gap*($range+1);
}

```

```

#!/usr/bin/perl -w
use strict;
require '/home/vossman/lib/fixnum.pl';

#amount (in %) to remove each of the data
#1--> keep 98% of data
my $cutoff;
my $debug = 0;
my @cutoffs = (1.25);
my @cutoffs = (0.0,0.01,0.1,0.2,0.4,0.8,1.0,1.1,1.2,1.3,1.4,1.6,1.8,2.0,4.8,16);
my @my @cutoffs = (0.0,0.1);

main();
sub main {
    my $exit;
    my @files = <ALL-*.*.out>;
    foreach my $file (@files) {
        foreach $cutoff (@cutoffs) {
            print STDERR $file, " --- $cutoff\n";
            $exit = AnalyzeVol($file,$cutoff);
        }
        if($exit) { print STDERR "good exit\n\n"; }
    }
}

sub AnalyzeVol {
    my ($outf,$cutoff) = @_;
    my $min; my $max;
    @keys = sort { $a cmp $b } @keys;
    my $count = 0;
    my $totatoms = readPDB($outf,\%list,\@keys);
    if($debug) {
        print STDERR "Found $totatoms total atoms\n";
        print STDERR "RES-ATM count [old --> new]\n";
        print STDERR "median\n";
    }
    my $drop,$bigrange = parseList(\%list,\@keys,\%min,\%max,$cutoff);
    print STDOUT "$cutoff\t$drop\t$bigrange\n";
    #return 1;
}

#PASS TWO --- edit the main file for limiters
my $chng = editFile($outf,\%max,\%min);
print STDERR "Changed $chng atoms\n\n";
my $bigchg=0;
my @files = <????-voronoi-?????.out>;
foreach my $file (@files) {
    $bigchg += editFile($file,\%max,\%min);
}
print STDERR "Changed $bigchg of expected $chng atoms\n\n";
return int($bigchg/$chng);
}

sub isRNA {
    my ($resi) = @_;
    $resi = uc($resi);
    $resi =~ s/(\^S+)/$resi =~ s/\^S//;
    my $hash = ('ADE',1,'GUA',1,'URC',1,'CYT',1);
    if($resi && $hash{$resi}) { return 1; }
    return 0;
}

sub getKey {
    my ($key) = @_;
    my $list = sort { $a <> $b } @{$$list{$key}};
    my $minInd = int($cutoff/100*\$#$list{$key})+0..5;
    my $maxInd = int((1-$cutoff/100)*(\$#$list{$key})); 
    my $minKey = $$list{$key}[$minInd];
    my $maxKey = $$list{$key}[$maxInd];
    my $oldrange = fixnum22(\$#$list{$key}) - $$list{$key}[0];
    my $newrange = fixnum22(\$#$list{$key}) - $$list{$key}[0];
    $drop += ($oldrange-$newrange);
    my $median = $$list{$key}[int(\$#$list{$key}/2)];
    $perdrop += $newrange/$median;
    $bigrange += $newrange;
    if($debug) {
}
}

```

```

print STDERR fillRight($key, 8), fixnum50((#{$$list{$key}})+1),
"[$oldrange, " .> " $newrange, " .> " $median), "\n";
} else { print STDERR " ";
print STDERR "$oldrange /=$count;
$drop /=$count;
return ($perdrop,$drop,$bigrange);
};

sub editFile {
#PASS TWO
my ($outf,$min) = @_;
my $count=0; my $carrot=1024;
my @keys; my %ovol; my %okay; my %ovolsq; my %isKEY;
my $unatoms=0; my $strations=0;
open(OUTF, '< $outf') or die "Can't open $outf for reading\n";
my $editf = $outf; my $chng=0;
substr($editf,-4) = ".edit.out";
print STDERR "analyzing $outf --> $editf\n";
open(EDITF, "> $editf") or die "Can't open $editf for writing\n";
while(defined(my $line = <OUTF>)) {
$count++;
if($count > $carrot) { print STDERR " "; $carrot*=1.25; }
my @chunks = split(' ', $line);
foreach my $test (@chunks) { $test =~ s/^\s+//; $test =~ s/\s+$/ /; }

my $REST = $chunks[3];
my $REST1 = $chunks[13];
if(isRNA($REST)) { $unatoms++; }
if(isRNA($REST1) && $chunks[13] eq "[OK]") {
my $key = getkey($REST,$ATOM);
my $vol = $chunks[10];
my $maxkey = $unatoms{$key} && $vol < $max{$key} ? $unatoms{$key} :
$unatoms++;
if(!$isKEY{$key}) {
push(@keys,$key);
$okay{$key} = 1;
$ovol{$key} = 1;
$ovolsq{$key} = ($vol*2);
} else {
$okay{$key}++;
$ovol{$key} *= $vol;
$ovolsq{$key} += ($vol**2);
}
else {
$chng++;
$line =~ s/^\s+//;
}
print EDITF $line;
}
close(OUTF);
close(EDITF);

#ANALYSIS --- ANALYSIS --- ANALYSIS --- ANALYSIS --- ANALYSIS
if($debug) {
print STDERR "key      mean    stdev  torperdev\n";
@keys = sort { $a cmp $b } @keys;
my %mean; my $totdev=0; my $count=0;
foreach my $key (@keys) {
if($okay{$key} > 10) {
$count++;
$stdev{$key} = sqrt((($ovolsq{$key}-($ovol{$key}**2))/($okay{$key}-1));
}
}
}
}

```

```

#!/usr/bin/perl -w
use Getopt::Std;
use Text::Tabs;

getopt("ios", \%args);

# Set input variables
if($args{'i'} || $args{'s'}) { $argst{'s'} = ! $args{'i'}; $in = $args{'i'}; $sym = $args{'s'}; $out = $args{'o'}; }

print "i\n$in\npub file: $in\n";
print "SYM input file: $sym\n";
system("wc -l $sym");

open(OUT, "> $out")
    or die "Couldn't open $out for writing\n";
print "Writing to: $out\n\n";
# Set input variables
# Open Files
print STDERR "Parsring $in file...\n";
open(IN, "< $in")
    or die "Couldn't open $in for reading\n";
$accept = 0;
$number_<resnum<chain<resname<atname
$protein<packeff<goodbad
$atom =~ $s+$s+$/; $atom =~ $s/\$/+;
$substr($line, 22, 4);
$resnum =~ $s/\$/+/; $resnum =~ $s/+/;
$substr($line, 17, 3);
$resnam =~ $s/\$/+/; $resnam =~ $s/+/;
$atnam =~ $s/\$/+/; $atnam =~ $s/+/;
$hash = $atom . "_" . $resnum . "_" . $resnam;
$exists{$hash}=1;
$accept++;
}
print "done (found $accept entries)\n\n";
close(IN);

while(defined($line = <IN>)) {
    if(substr($line, 0, 4) eq "ATOM") {
        $atom = substr($line, 6, 5);
        $atom =~ $s+$s+$/; $atom =~ $s/\$/+;
        $substr($line, 22, 4);
        $resnum =~ $s/\$/+/; $resnum =~ $s/+/;
        $substr($line, 17, 3);
        $resnam =~ $s/\$/+/; $resnam =~ $s/+/;
        $atnam =~ $s/\$/+/; $atnam =~ $s/+/;
        $hash = $atom . "_" . $resnum . "_" . $resnam;
        $exists{$hash}=1;
        $accept++;
    }
}

print "done (found $accept entries)\n\n";
close(IN);

print STDERR "Chopping $sym file and writing...\n";
open(SYM, "< $sym")
    or die "Couldn't open $sym for reading\n";
$carrot = int($accept/50);
print STDERR "$carrot\r";
while(defined($line = <SYM>)) {
    if($accept % $carrot == 0) {
        print STDERR "\r";
        $expand($line);
        $numb = $line + 1;
        @branks = $atom . "+" . $resnam . "+" . $atnam . "+" . $resnum;
        $atom = substr($line, 6, 5);
        $atom =~ $s+$s+$/; $atom =~ $s/\$/+;
        $substr($line, 22, 4);
        $resnum =~ $s/\$/+/; $resnum =~ $s/+/;
        $substr($line, 17, 3);
        $resnam =~ $s/\$/+/; $resnam =~ $s/+/;
        $atnam =~ $s/\$/+/; $atnam =~ $s/+/;
        $hash = $atom . "_" . $resnum . "_" . $resnam;
        $exists{$hash}=1;
        $accept++;
    }
}

```

```

$atom = $chunks[0];
$atom =~ s/\s+$/;; $atom =~ s/^\s+//;
$resnum = $chunks[1];
$resnum =~ s/\s+$/;; $resnum =~ s/^\s+//;
$resnam = $chunks[3];
$resnam =~ s/\s+$/;; $resnam =~ s/^\s+//;
$atnam = $chunks[4];
$atnam =~ s/\s+$/;; $atnam =~ s/^\s+//;
$hash = $atom . "-" . $resnum . "-" . $resnam . "-" . $atnam;
if($exists{$hash}) {
    print OUT $line;
    $count++;
    if($count % $carrot == 0) { print STDERR "^.";
    }
    if($exists{$hash} > 1) {
        print STDERR "\nhash collision... $hash\n";
    }
    $exists{$hash}++;
}
}
close(SYM);
print STDERR "\ndone (found $count of original $accept entries)\n\n";
close(OUT);

```

```

#!/usr/bin/perl -w
use Getopt::Std;
use strict;
my $debug = 0;
my $spec;
my $method; # 1 -- use symmetry, 2 -- no symm
main();
sub main {
    my $cmd;
    my ($atomdef,$residef,$stdvol,$symfile,$namfile,$outfile) = getINPUT();
    if($method == 1) {
        $cmd = mkCMD($atomdef,$residef,$stdvol,$symfile,In2out($symfile));
    } else {
        $cmd = mkCMD($atomdef,$residef,$stdvol,$namfile,$outfile);
    }
    if($debug) { print STDERR $cmd, "\n"; } else { system($cmd); }
    if($method == 1) {
        $cmd = Truncate2(In2out($symfile),$namfile,$outfile);
        if($debug) { print STDERR $cmd,".\n"; } else { system($cmd); }
        system("rm -f ".In2out($symfile));
    }
    print STDERR "\n\n\n$outfile created\n";
}
sub getUsername {
    system("uname > temp.$spec");
    open(TEMP, "< temp.$spec") or die;
    if(defined(my $line = <TEMP>)) {
        $line =~ s/^\s+$/; $line =~ s/\s+//;
        close(TEMP);
        system("rm -f temp.$spec");
        return $line;
    }
}
sub Truncate2 {
    my ($symfile,$infile,$outfile) = @_;
    my $path = getPath();
    my $execfile = $path."/truncate2.pl";
    if(!stat($execfile)) { die; }
    return "perl $execfile -s $symfile -i $infile -o $outfile";
}
sub Truncate {
    my ($symfile,$infile,$outfile) = @_;
    my $path = getPath();
    my $execfile = $path."/truncate.pl";
    if(!stat($execfile)) { die; }
    return "perl $execfile -s $symfile -i $infile -o $outfile";
}
sub In2out {
    my ($infile) = @_;
    my @chunks = split('/', $infile);
    substr(@chunks[$#chunks-4]) = "temp-$spec.out";
    return $chunks[$#chunks];
}
sub mkCMD {

```

```

#!/bin/perl -w
use Getopt::Std;
use constant PI => 3.14159265358979;

#SUPERPAK v0.8
# starting over...
# - read symmetry matrices from PDB file itself
# - ignore symop.lib altogether
# - perform trasnlation on generate unit cell
# - transforming single point; transform prism with center pt.
# - fix for overlap (x,y,z) < cut
# - only include close transformations
# - includes own matrix subroutines

getopt("o:isgcmrth", \%args);
$ourfile = $args{to} || "out.pdb"; #output file
$pdbfile = $args{is} || "in.pdb"; #input file
$Symfile = $args{sg} || "symop.lib"; #CPB4 symop lib file
$gen_zero = $args{g} || 1; #output original PDB x,y,z
if($gen_zero != 1){ $gen_zero = 0; }
$cutoff_plus = $args{c} || 25; #this isn't working...
$cutoff_mult = $args{m} || 20; #cutoff_mult=/=100;
$cutoff_mult = $cutoff_mult * 100;
print STDERR "cutoff = $cutoff_mult * size + $cutoff_plus\n";
$range = $args{r} || 3;
$mintrans = $args{t} || 1000;
$mintrans = $args{b} || 0;
print STDOUT "#####\n";
print STDERR "#####\n";
print STDERR "< $pdbfile\n";
open (PDB, "< $pdbfile");
or die "Failed opening $pdbfile for reading: $!\n";
$cryst=0; $fscal=0; $sintry=0;
while((&$args{t} && &$args{m}) && &$sintry) && defined($line=<PDB>)) {
    if(substr($line,0,19) eq "REMARK 290 SMTRY1") {
        print STDERR "Entering SMTRY loop... ";
        do {
            $chomp($line);
            $chomps=$chunks[3];
            $opnum = $chunks[3];
            print STDERR "[ $opnum ] ";
            print $chomps[4]; $chomps[5]; $chomps[6]; $chomps[7];
            $line=<PDB>;
            $chomp($line);
            $chomps=$chunks[3];
            $opnum2 = $chunks[3];
            $chomps[3];
            $chomps[4]; $chomps[5]; $chomps[6]; $chomps[7];
            $chomp($line);
            $chomps=$chunks[3];
            $opnum3 = $chunks[3];
            $chomps[4]; $chomps[5]; $chomps[6]; $chomps[7];
            $matrix[$opnum] = [ [ $x11, $x21, $x31 ], [ $x12, $x22, $x32 ], [ $x13, $x23, $x33 ] ];
            $vector[$opnum] = [ [ $y1 ], [ $y2 ], [ $y3 ] ];
            if($opnum != $opnum2 || $opnum != $opnum3) {
                die "opnums don't match on read in $opnum1 $opnum2 $opnum3 !";
            }
            push(@opers,$opnum);
            while(defined($line=<PDB>) && substr($line,0,18) eq "REMARK 290 SMTRY");
            print STDERR "done\n";
            print STDERR "found ". $#opers+1 ." of $opnum operations (including identity)\n";
            $sintry=1;
            if(substr($line,0,5) eq "CRYST") {
                $a = substr($line,6,8); $a =~ s/+/s++/; $aa =~ s/^\s+/s+/; $ab =~ s/^\s+/s+/;
                $b = substr($line,15,8); $b =~ s/^\s+/s+/;
            }
            $symp=$substr($line,67,3); $symp =~ s/+/s++/; $symp =~ s/^\s+/s+/;
            $symp2 = $symp;
            $sympm=$substr($line,55,10); $sympm =~ s/+/s++/; $sympm =~ s/^\s+/s+/;
            print STDOUT "a=$a, b=$b, c=$c, alpha=$alpha, beta=$beta, gamma=$gamma, symm=$symp2\n";
            $cryst=1;
            elsif(substr($line,0,5) eq "SCALE") {
                $chomp($line);
                @chunks=$split( ' ', $line );
                $x11=$chunks[1]; $x21=$chunks[2]; $x31=$chunks[3];
                $line=<PDB>;
                $chomp($line);
                @chunks=$split( ' ', $line );
                $x12=$chunks[1]; $x22=$chunks[2]; $x32=$chunks[3];
                $line=<PDB>;
                $chomp($line);
                @chunks=$split( ' ', $line );
                $x13=$chunks[1]; $x23=$chunks[2]; $x33=$chunks[3];
                $scale = [ [ $x11, $x21, $x31 ], [ $x12, $x22, $x32 ], [ $x13, $x23, $x33 ] ];
                print STDERR "Found SCALE matrix\n";
                matprint(mult($scale,[[$b,0],[0,$c]]));
                $invscale = matinv($scale);
                $fcale=1;
            }
            }
            #find average x,y,z
            $residues = 0;
            $atoms=0; $xmin=10000; $xmax=0; $zmin=10000; $zmax=0;
            do {
                if(substr($line,0,4) eq "ATOM" || substr($line,0,6) eq "HETATM") {
                    $x = $scale->[ [ $x11, $x21, $x31 ], [ $x12, $x22, $x32 ], [ $x13, $x23, $x33 ] ];
                    $y = $scale->[ [ $y1 ], [ $y2 ], [ $y3 ] ];
                    $z = $scale->[ [ $z1 ], [ $z2 ], [ $z3 ] ];
                    $res = substr($line,22,4);
                    $res =~ s/+/s++/; $res =~ s/^\s+/s+/;
                    if($x < $xmin) { $xmin = $x; $xmin = $y; $xmax = $z; }
                    if($x > $xmax) { $xmax = $x; $xmax = $y; $xmaxz = $z; }
                    if($y < $ymin) { $ymin = $y; $yminx = $y; $yminz = $z; }
                    if($y > $ymax) { $ymax = $y; $ymaxx = $y; $ymaxz = $z; }
                    if($z < $zmin) { $zmin = $z; $zminy = $y; $zmaxx = $x; }
                    if($z > $zmax) { $zmax = $z; $zmaxy = $y; $zmaxx = $x; }
                    $atoms++; $xavg+=-$x; $yavg+=-$y; $zavg+=-$z;
                }
            }
            if($residues < $res) { $residues = $res; }
        }
        while(defined($line=<PDB>));
        close(PDB);
        $xavg= int($xavg*1000/$atoms)/1000;
        $yavg= int($yavg*1000/$atoms)/1000;
        $zavg= int($zavg*1000/$atoms)/1000;
        print STDOUT "#atoms: $atoms #residues: $residues\n";
        print STDOUT "max ( x, y, z ) = ( $xmax, $ymax, $zmax )\n";
        print STDOUT "avg ( x, y, z ) = ( $xavg, $yavg, $zavg )\n";
        $xavg2= int(( $xmax*$ymin)*500)/1000;
        $yavg2= int(( $ymin*$zmin)*500)/1000;
        $zavg2= int(( $zmin*$xmax)*500)/1000;
        $length= int(sqrt( ( $xavg2-$xmin)**2 + ( $yavg2-$ymin)**2 + ( $zavg2-$zmin)**2 ))*10/10;
        $shift= int(sqrt( ( $xavg-$xav2)**2 + ( $yavg-$yav2)**2 + ( $zavg-$zav2)**2 ))*10/10;
        $percent = int($shift/$length*1000)/10;
    }
}

```

```

print STDOUT "av2 (x,y,z) = ($xav2, $yavg, $zavg2) -- [ $shift A ($percent%) ]\n";
print STDOUT "min (x,y,z) = ($xmin, $ymin, $zmin)\n"; #Find MAX RADIUS of pdb
$maxdist=1;
$i=0;
#insert points...
$mainpoint=[ $xavg ],[ $yavg ],[ $zavg ],[ ];
push(@points,$mainpoint);
push(@points,[ $xmin ],[ $ymin ],[ $zmin ],[ 1 ]);
push(@points,[ $xav2 ],[ $yavg ],[ $zavg2 ],[ 1 ]);
push(@points,[ $xmin ],[ $ymin ],[ $zmin ],[ 1 ]);
push(@points,[ $ymin ],[ $ymin ],[ $ymin ],[ 1 ]);
push(@points,[ $zmax ],[ $zmax ],[ $zmax ],[ 1 ]);
push(@points,[ $ymin ],[ $ymin ],[ $ymin ],[ 1 ]);
push(@points,[ $zmax ],[ $zmax ],[ $zmax ],[ 1 ]);
push(@points,[ $zmax ],[ $zmax ],[ $zmax ],[ 1 ]);
push(@points,[ $zmax ],[ $zmax ],[ $zmax ],[ 1 ]);

#FIND MAX DISTANCE
foreach $x2 ($xav2,$ymin,$zmin) {
    foreach $y2 ($yavg,$ymin,$zmin) {
        foreach $z2 ($zmax,$ymin,$zmin) {
            $dist=int(sqrt((($xav2-$x2)**2 + ($yavg-$y2)**2 + ($zavg-$z2)**2)*10)/10;
            if($dist > $maxdist) { $maxdist=$dist; }
        }
    }
}
$cut=0;
print STDOUT "max radius = [ $maxdist A ]\n";
$cutoff = $cutoff*mult*$maxdist + $shift + $cutoff_plus;
$percent = int($cutoff*($length*1000)/10;
print STDOUT "$CUTOFF: [ $cutoff A ($percent)% ]\n";
#Opnum=1 is identity, so start at 2, 0 is undefined
for($opnum=2;$opnum<=$opers+1;$opnum++) {
    $included=0;
    for($i=0;!$included && $i<=$#points;$i++) {
        $point = $points[$i];
        $newpoint = matadd(mult($matrix{$opnum}),$point),$vector{$opnum});
        foreach $point2 (@points) { #check if its withing distane of any points }
        #
        $point2=$mainpoint;
        $val = matsub($newpoint,$point2);
        $num = mmult(transpose($val),$val);
        $dist = sqrt($num);
        push(@unite1,$newpoint);
        print STDOUT "$Opnum: \" .int($dist). \"\t";
        if($dist < $cutoff) {
            #accepted, check for duplicates
            $dup=0;
            if($ingoodlist{$opnum}) { $dup=1; }
            if($dup) {
                push(@goodlist,$opnum);
                $ingoodlist{$opnum}=1;
                $included=1;
            }
        }
    }
}
print STDOUT "kept ".($#goodlist+1)." of $opers non-identity transformations\n";
if($#goodlist+1 < $opers) { $range=2; }
#create shifts
$changes=0;
do {
    $maxrefnum=0;
    for my $i=$range;$i<=$range;$i++) {
        for my $j=-$range;$j<=$range;$j++) {
            for my $k=-$range;$k<=$range;$k++) {
                $refnum=($i+$opnum)*($range*2+1)*($j+$range)*($range*2+1)+($k+$range)+1;
                if( ($i==0 && $j==0 && $k==0) || ($shifts[$refnum] == [[$i],[$j],[$k],]) ) {
                    $shifts[$refnum] = [$refnum];
                    if($refnum > $maxrefnum) { $maxrefnum = $refnum; }
                }
            }
        }
    }
}
print STDOUT "\n";
if($#goodlist > $opers) { $range++; }
#Create shifts
$changes=0;
do {
    $maxrefnum=0;
    for my $i=$range;$i<=$range;$i++) {
        for my $j=-$range;$j<=$range;$j++) {
            for my $k=-$range;$k<=$range;$k++) {
                $refnum=($i+$opnum)*($range*2+1)*($j+$range)*($range*2+1)+($k+$range)+1;
                if( ($i==0 && $j==0 && $k==0) || ($shifts[$refnum] == [[$i],[$j],[$k],]) ) {
                    $shifts[$refnum] = [$refnum];
                    if($refnum > $maxrefnum) { $maxrefnum = $refnum; }
                }
            }
        }
    }
}
print STDOUT "Now have ".($#goodlist+1)." of " .((#$opers+1)*$maxrefnum)
print STDOUT " non-identity molecules.\n";
print STDOUT "Time to generate new molecules.. .\n";
my $i=0;
if($gen_zero) {
    print STDOUT "Generating Identity Molecule as well\n";
    push(@goodlist,1);
}

```

```
$ingoodlist[1]=1;
} else {
    print STDERR "Ignoring Identity Molecule\n";
}
print STDOUT "$atoms atoms to generate\n";
$cut=int($atoms/50);

open (OUT, "> $outfile") or die "Failed opening $outfile for writing: $!\n";
my $struct=0;
foreach $id (@goodlist) {
    $struct++;
    print STDERR "Working on structure $struct of ".($#goodlist+1)." ID=$id\n";
    my $unique_id=$struct.".".substr($id,0,5);
    print STDERR "#Unique id:$refnum\n";
    my $opnum=$refnum->{"$unique_id:$refnum"}->{"$id"};
    ($refnum, $opnum)=split(' ', $id);
    if($opnum){ $opnum=$refnum; $refnum=0; }
    $refnum=multimatrix($shifts[$refnum]),$shifts[$refnum])->[0][0];
    open (PDB, "< $pdbfile");
    my $count=0;
    while(<PDB> && ($substr($line,0,4) eq "ATOM" || substr($line,0,6) eq "HETATM" )) {
        do {
            if(substr($line,0,4) eq "ATOM" || substr($line,0,6) eq "HETATM" ) {
                $x=$substr($line,30,8); $y=$substr($line,38,8); $z=$substr($line,46,8);
                $line =~ s/\$/\$//; $line =~ s/\$/\$//;
                # print STDERR $line . "\n";
                $x =~ s/\$/\$//; $x =~ s/\$/\$//;
                $y =~ s/\$/\$//; $y =~ s/\$/\$//;
                $z =~ s/\$/\$//; $z =~ s/\$/\$//;
                $res = substr($line,22,4);
                $res =~ s/\$/\$//; $res =~ s/\$/\$//;
                $vec=[[$x],[$y],[$z]];
                #symmetry operate (Opnum=1 => identity)
                $newvec = matadd($multimatrix[$opnum],$vec),$vector{$opnum};
                #scale and shift (Refnum=0 => identity)
                $newvec = matadd($multimatrix[$scale],$newvec);
                #transform back
                $newvec = mmult($invscale,$newvec);
                $x2=$newvec->[0][0];
                $z2=$newvec->[1][0];
                substr($line,46,8)=proper($x2);
                substr($line,38,8)=proper($z2);
                substr($line,30,8)=proper($x2);
                if( ($refnum==0 && $opnum==1) {
                    substr($line,56,1)=0;
                    $bfact = proper($rnum-$opnum/10);
                    substr($bfact,-1)=";
                    substr($bfact,0,2)="";
                    substr($line,61,5)=abtract;
                    $newres = proper((($resides*$struct + $res)%1000);
                    substr($newres,-4)="";
                    substr($line,22,4)=$newres;
                    my $label = makelabel($struct);
                    substr($line,20,1)=substr($label,0,1);
                }
                print OUT $line."\n";
            }
            $count++;
            if($count % $cut == 0) { print STDERR "\n"; }
        } while(defined($line=>PDB));
        print OUT "COMPND\n";
        print STDERR "\n";
    }
}
```

```
} print OUT "END\n";
}

sub makelabel {
    my $struct = $-[0];
    my $label;
    if($struct < 10) { $label=~tr/1-9/A-I/; }
    elsif($struct < 20) { $label=~tr/0-9/J-S/; }
    elsif($struct < 27) { $label=~tr/0-6/T-Z/; }
    elsif($struct < 37) { $label=~tr/0-9/k-t/; }
    elsif($struct < 47) { $label=~tr/0-9/a-j/; }
    elsif($struct < 53) { $label=~tr/0-47/l-b/; }
    else { $label=~tr/0-5/u-z/; }
    return $label;
}

sub proper {
    my ($num) = @_;
    $toler = 0.00001;
    $num = int($num*1000)/1000;
    $text="";
    if($num > 0) {
        if($num < 10) { $text = " ".$num; }
        elsif($num < 100) { $text = " ".$num; }
        elsif($num < 1000) { $text = " ".$num; }
        else { $text = $num; }
    } elsif($num < 0) {
        if($num < -10) { $text = " ".$num; }
        elsif($num > -100) { $text = " ".$num; }
        else { $text = $num; }
    }
    if(abs(int($num) - $num) < $toler) {
        if($text =~ /\Q000\Q/ ) {
            elsif(abs(int($num*10) - $num*10) < $toler) {
                if( (1 - abs(int($num*10) - $num*10)) < $toler )
                    { $text = $text."00"; }
                elsif(abs(int($num*10) - $num*10) < $toler) {
                    if( (1 - abs(int($num*100) - $num*100)) < $toler )
                        { $text = $text."00"; }
                    elsif(abs(int($num*100) - $num*100) < $toler) {
                        if( (1 - abs(int($num*100) - $num*100)) < $toler )
                            { $text = $text."0"; }
                        else { $text = $text."0"; }
                    }
                }
            }
        }
    }
    if(length($text) != 8) {
        if((int($num*100) - $num*100) == 0) { $text = $text."0"; }
        print STDERR "ERROR: Incorrect Length: '$text'\n";
        print "abs($num*100 - int($num*100)).\"\\n\";\n";
        return " 0.000";
    }
    return $text;
}

sub mult {
    my ($m1,$m2) = @_;
    my ($m1rows,$m1cols) = matdim($m1);
    my ($m2rows,$m2cols) = matdim($m2);
    if($m1cols != $m2rows && ($m1rows == 1 || $m2cols == 1) ) {
        print "multiply $m1rows x $m1cols by $m2rows x $m2cols error\\n";
        $m2 = transpose($m2);
        ($m2rows,$m2cols) = matdim($m2);
    }
    unless($m1cols == $m2rows) { # raise exception
        print "transposed: multiply $m1rows x $m1cols by $m2rows x $m2cols\\n";
    }
    unless($m1cols == $m2rows) { # raise exception
        print "print \"$m2rows\"; ";
        die "Cannot multiply these matrices bad array sizes: $m1cols != $m2rows $!";
    }
    my $result = [];
    my ($i,$j,$k);
}
```

```

for($i=0;$i<$m1rows;$i++) {
    for($j=0;$j<$m2cols;$j++) {
        for($k=0;$k<$m1cols;$k++) {
            $result->[$i][$j] += $m1->[$i][$k] * $m2->[$k][$j];
        }
    }
}
return $result;
}

sub matsub {
    my ($m1,$m2) = @_;
    my ($m1rows,$m1cols) = matdim($m1);
    my ($m2rows,$m2cols) = matdim($m2);
    if($m1cols != $m2cols && ($m2rows == 1 || $m2cols == 1) ) {
        print "multiply $m1rows x $m1cols by $m2rows x $m2cols error\n";
        $m2 = transpose($m2);
        ($m2rows,$m2cols) = matdim($m2);
        print "transposed: multiply $m1rows x $m1cols by $m2rows x $m2cols\n";
    }
    unless($m1cols == $m2cols && $m1rows == $m2rows) { # raise exception
        print "subtract $m1rows x $m1cols by $m2rows x $m2cols\n";
        die "Cannot subtract these matrices bad array sizes $!\n";
    }
    my $result = [];
    my ($i,$j);
    for($i=0;$i<$m1rows;$i++) {
        for($j=0;$j<$m2cols;$j++) {
            $result->[$i][$j] = $m1->[$i][$j] - $m2->[$i][$j];
        }
    }
}
return $result;
}

sub matadd {
    my ($m1,$m2) = @_;
    my ($m1rows,$m1cols) = matdim($m1);
    my ($m2rows,$m2cols) = matdim($m2);
    if($m1cols != $m2cols && ($m2rows == 1 || $m2cols == 1) ) {
        print "multiply $m1rows x $m1cols by $m2rows x $m2cols error\n";
        $m2 = transpose($m2);
        ($m2rows,$m2cols) = matdim($m2);
        print "transposed: multiply $m1rows x $m1cols by $m2rows x $m2cols\n";
    }
    unless($m1cols == $m2cols && $m1rows == $m2rows) { # raise exception
        print "subtract $m1rows x $m1cols by $m2rows x $m2cols\n";
        die "Cannot subtract these matrices bad array sizes $!\n";
    }
    my $result = [];
    my ($i,$j);
    for($i=0;$i<$m1rows;$i++) {
        for($j=0;$j<$m2cols;$j++) {
            $result->[$i][$j] = $m1->[$i][$j] + $m2->[$i][$j];
        }
    }
}
return $result;
}

sub veclen {
    my $ary_ref = $_[0];
    my $type = ref $ary_ref;
    if($type ne "ARRAY") { die "$type is bad array ref for $ary_ref $!"; }
    return scalar(@$ary_ref);
}

sub matdim {
    my $m = $_[0];
    my $rows = veclen($m);
    my $cols = veclen($m->[0]);
    return ($rows,$cols);
}

```

```

my $cols = veclen($m->[0]);
return ($rows,$cols);
}

sub matprint {
    my $m = $_[0];
    my ($rows,$cols) = matdim($m);
    print STDERR "Matrix size: $rows x $cols\n";
    for(my $i=0;$i<$rows;$i++) {
        print STDERR "[\\";
        for(my $j=0;$j<$cols;$j++) {
            print STDERR int($m->[$i][$j]*10000)/10000, "\\";
            print STDERR " ]\n";
        }
    }
}

sub matinv {
    #Assume 3x3
    my $m = $_[0];
    my ($rows,$cols) = matdim($m);
    if($rows == 3 && $cols == 3) { die "cannot invert non-3x3 matrix $!\n"; }
    my $det = matdet($m);
    if($det == 0) { die "matrix is singular, cannot invert $!\n"; }
    my $result = [];
    for(my $i=0;$i<$rows;$i++) {
        $result->[$i][0] = (
            $m->($i+1)%3*[$i][1]%3*[$i-1][2]%3] -
            $m->($i+2)%3*[$i+1][3]*[$i-2][($i+1)%3][($i+2)%3] -
            $result->[$i][1]*[$i][2];
        for(my $j=0;$j<$rows;$j++) {
            $result->[$i][$j] = $result->[$i][0]/$det;
        }
    }
    # matprint($result);
    return $result;
}

sub matdet {
    #Assume 3x3
    my $m = $_[0];
    my ($rows,$cols) = matdim($m);
    if($rows == 3 && $cols == 3) { die "cannot do non-3x3 matrix $!\n"; }
    my $det = $m->0][0]*[$m->1][1]*[$m->2][2] - $m->0][1]*[$m->1][2]*[$m->2][1]
    - $m->0][2]*[$m->1][0]*[$m->2][1] + $m->1][0]*[$m->0][1]*[$m->2][1];
    return $det;
}

sub transpose {
    my $m = $_[0];
    my ($rows,$cols) = matdim($m);
    my $result = [];
    my $rows_x_cols = $rows * $cols;
    # transpose $rows x $cols;
    for(my $i=0;$i<$cols;$i++) {
        for(my $j=0;$j<$rows;$j++) {
            $result->[$i][$j] = $m->[$j][$i];
        }
    }
    my ($rows2,$cols2) = matdim($result);
    # print "into $rows2 x $cols2\n";
    # result $result;
    return $result;
}

my $ary_ref = $_[0];
my $type = ref $ary_ref;
if($type ne "ARRAY") { die "$type is bad array ref for $ary_ref $!"; }
return scalar(@$ary_ref);

sub matdim {
    my $m = $_[0];
    my $rows = veclen($m);

```

E.3 Perl Scripts for the Solvent and Tunnel Calculations

- ccp42png.pl: automatically converts a series of CCP4 electron density maps into PNG images using pymol.
- get_close-atoms.pl: get atoms from one PDB file that are close to atoms into a second PDB file. This was used to extract atoms near the ribosomal exit tunnel.
- get_near-tunnel.pl: get atoms within a certain distance of a vector defined to be the tunnel. The resulting extraction is a cylinder.
- get_tunnel-dist.pl: for a given PDB determine where it is along a vector defining the exit tunnel. This was used to create the bar for each ribosomal protein and the azithromycin in Figure 4.5.
- pdb2pts.pl: converts a PDB into a PTS file suitable for use with ‘qhull’ a program that computes the convex hull of a set of pts. The coordinates cannot be extracted directly because the convex hull needs to be computed on points that lie on the VDW surface, so this program randomly distributes points on the VDW surface of each atom.
- rotate-pdb.pl: using a matrix calculated from LSQMAN this program rotates and translate a PDB.
- sugar-pucker.pl: calculates the sugar pucker of all nucleotides in a PDB file.
- tunnel-worm.pl: The program I wrote to implement the vector stepping method to extract the ribosomal exit tunnel.
- xyzr2pts.pl: similar to the ‘pdb2pts.pl’ program, this program does the same for a XYZR file.

```

#!/usr/bin/perl
use strict;
require '/home/vossman/lib/fixnum.pl';

my @nums = (0,50,100,150,200,300,400,500,600,700,800,900,1000,);

print STDERR "YOU MUST TYPE:\n$setenv DISPLAY :1\n";
print STDERR "OR -v\nto export DISPLAY=':1'\nBEFORE RUNNING\n\n";;

my @files = <> ccp4/vol_1ij2_*-[2-9]????.ccp4.bz2>;
foreach my $f (@files) {
    my $n=1100; $n=0; $n=50; {
        my $f = $filestem(rand($#files));
        print STDERR "#FILENUM\t$f\t$t\t$file2num($f), \"$n\";\n";
        my $g = $1;
        my $n = $2;
        my $nb = fixnum50b($n);
        if($n < 300) { die; }
        print STDERR "grid = \"$g,\";\n";
        print STDERR "probe = \"$n,\";\n";
        print STDERR "image = \"$image.png\";\n";
        my $image = "ping/vol$nb.png";
        if(stat($image)) {
            &setup($f);
            my $tf = $f;
            system("rm -f volume ccp4");
            if(stat($tf)) {
                system("cp $tf volume ccp4.bz2");
                system("bunzip2 volume ccp4.bz2");
            }
            &editVolPy($n);
            if(stat($tf)) {
                system("rm -f volume ccp4");
                if(stat($tf)) {
                    system("cp $tf volume ccp4.bz2");
                    system("bunzip2 volume ccp4.bz2");
                }
            }
        }
    }
}

```

```

my $str = "([\"fixnum32($r)\", \"fixnum32($e)\", \"fixnum32($b)\", \"])";
my $rot = 0; #&2rot($n);
print STDERR "==== ROTATION= $rot t RGB= \\".fixnum30($r).\", \"."fixnum30($g);
system("cat volume.py");
my $s = `sed 's/\\([200,200,200]\\)/\\1/g';
my $t = `sed 's/\\([0,100,100]\\)/\\1/g';
my $u = `sed 'vol_1ij2_py'`;
my $n = $1;

sub n2rgb {
    my ($n) = @_;
    my ($r,$g,$b);
    my ($r,$g,$b);
    # 0% = (-128, 0) +1r +1/2g
    # 33% = (255,255, 0) -1g
    # 67% = (255, 0, 0) +1b -1/2r
    #100% = (128, 0,255)
    ## 10% => 2.5
    ## 100% => 6.3
    my $per;
    my $per;
    if($n <= 15000) {
        $per = 100.0*( $n-0)/ (15000-00);
    } else {
        $per = 100.0 - 100.0*($n-15000)/(100000-15000);
    }
    if($per <= 33.333333) {
        my $fact = 255.0*$per/33.34;
        $g=$fact/2+128;
        $b0=0;
        $g=255-$fact;
        $b=0;
    } elsif($per <= 66.666667) {
        my $fact = 255.0*($per-33.33)/33.34;
        $g=255;
        $b=255-$fact;
        $b=0;
    } else {
        #100% = (128, 0,255)
        my $fact = 255.0*($per-66.66)/33.34;
        $r = 255 - $fact/2;
        $g = 0;
        $b = $fact;
    }
    return ($r,$g,$b);
}

sub n2rot {
    my ($n) = @_;
    my ($rot);
    if($n < 15000) {
        my $frame = $n-00;
        $rot = ($frame*0.06);
    } else {
        my $frame = $n-15000;
        $rot = ($frame*0.06) + 15000*0.06;
    }
    while($rot >= 360) { $rot -= 360.0; }
    return ($rot);
}

sub file2num {
    my ($f) = @_;
    my $n = $volume$nb.$f;
    my $n = $1;
    return $n;
}

sub editVolPy {
    my ($n) = @_;
    my $nb = fixnum50b($n);
    my ($r,$g,$b) = (50,187,50); #&n2rgb($n);

```

```

#!/usr/bin/perl
use strict;
use Getopt::Std;
use Term::ANSIColor;
require '/home/vossman/lib/fixnum.pl';
use constant PI => 3.14159265359;
my %args;
getopt("it",\%args);

my $version = "0.9";
my $pdbin = $args{i} || "PDB/1mlk-small.pdb";
my $tunnelin = $args{t} || "small-tunnel1-3b.pdb";
my @chks = split('/', $pdbin);
my $pdbid = lc(substr($chks[-2],0,4));
my $output_files
my $pdabout = "close2-".$pdbid.".pdb";
my $range = 6;
my $rangesq = $range**2;

main();
sub main {
    print STDERR color("magenta"), "\n\tgetClose v$version\n\n";
    my @vecs = readDB($tunnelin);
    print STDERR color("yellow");
    color('reset');
    #preloading
    print STDERR "WRITING TO $pdabout\n\n";
    my $vecs = readDB($tunnelin);
    print STDERR color("yellow");
    getClose($pdbin,\@vecs);
    print STDERR color("reset");
    print STDERR "\n\n";
}

sub getClose {
    my ($pdbin,$vecs) = @_;
    print STDERR "Reading in $pdbin...\n";
    my $numlines = getNumLines($pdbin);
    print STDERR "$numlines\n";
    my $cut = primBar($numlines);
    open(PDB, "<$pdbin") or die "$pdbin does not exist\n";
    open(OUT, ">$pdabout") or die "$pdabout does not exist\n";
    my $count; my $carrot = $cut; my $lines;
    while(defined(my $line = <PDB>)) {
        if($line =~ /ATOM/ || $line =~ /\^HETATM/) {
            my $num = clean(substr($line,6,5));
            my $x = clean(substr($line,30,8));
            my $y = clean(substr($line,38,8));
            my $z = clean(substr($line,46,8));
            #get x, y, z, vdw radius (req resi/atom-defs)
            my $good=0;
            my $vec = $vecs[$i];
            my $vecsq = $vecs[$i];
            if($distsq < $rangesq) { $good = 1; }
            if($good) {
                print OUT $line;
            }
            $count++;
        }
        $lines++;
    }
    if($lines > $carrot) { print STDERR "\n"; $carrot += $cut; }
    print STDERR "\n$count atoms\n";
    close(PDB);
}

```

```

print STDERR "<n$ccount atoms\n";
close(PDB);
return @vecs;
};

sub getNumLines {
    my ($file) = @_;
    my $num=0;
    system("wc -l $file > numlines.txt");
    open(NML, "< numlines.txt") or die;
    if(defined(my $line = <NML>)) {
        $line =~ m/\n*(10-9]+)/;
        $num = $1;
    }
    close(NML);
    system("rm -f numlines.txt");
    return $num;
};

sub list2vec {
    my (@list) = @_;
    my $vec;
    foreach my $item (@list) {
        $vec .= $item." ";
    }
    return $vec;
};

sub vec2list {
    my ($vec) = @_;
    return split(' ', $vec);
};

sub noclash {
    my ($vec1,$vec2) = @_;
    my ($x1,$y1,$z1,$r1,$anum) = vec2list($vec1); #protein/rna
    my ($x2,$y2,$z2,$r2) = vec2list($vec2); #water
    my $distsq = ($x1-$x2)**2 + ($y1-$y2)**2 + ($z1-$z2)**2;
    if($distsq < ($r1 + $r2)**2) { return 0; }
    return ($distsq,$anum);
};

sub printBar {
    my ($num) = @_;
    my $mult=1;
    while($num/$mult > 76) {
        $mult++;
    }
    $num = int($num/$mult+0.5);
    print STDERR "$";
    for my $i=1;$i<=$num-2;$i++) {
        if($i==int((($num-2)*2+0.501)) { print STDERR "\n"; }
        elsif($i==int(($num-2)/4+0.501)) { print STDERR "#"; }
        elsif($i==int(3*($num-2)/4+0.501)) { print STDERR "#"; }
        elsif($i%5==0) { print STDERR "+"; }
        else { print STDERR "."; }
        if($i%100==0) { print STDERR "\n"; }
    }
    print STDERR "\n";
    return $mult;
};

sub subVec {
    my ($v1,$v2) = @_;
    my ($x1,$y1,$z1) = vec2list($v1);
    my ($x2,$y2,$z2) = vec2list($v2);
    $x1 -= $x2;
}

```

```

#!/usr/bin/perl
use strict;
use Getopt::Std;
use Term::ANSIColor;
require '/home/vossman/lib/fixnum.pl';
use constant PI => 3.14159265359;
my %args;
getopt("lor", \%args);

#variables
my $version = "1.0";

#START POINT(S)
#perer active site
my $point = list2vec(71.529,120.46,97.335);
#DIRECTION VECTOR
#REFINE MORE
my $vector = normVec(list2vec(-35.88,37.27,-33.74));

#input pdb
my $pdbname = $args{'i'} || "PDB/lj2-fin.pdb";
my @chks = split('/', $pdbname);
my $pdbname = lc(substr($chks[-2],0,4));
#max radius to "give up"
my $endrad = $args{'r'} || 55;
#output files
my $dbout = $args{'o'} || $pdbname "-small.pdb";
#global system variables (do not modify)
my $maxDist = 80; #length of tunnel (to be removed)

main();

sub main {
    print STDERR color("magenta"), "\n\n\mMake Smaller v$version\n\n";
    color("reset");
    #preloading
    my @vecs = readPDB($pdbname);
    print STDERR "\n\n";
}

main();

```

```

sub main {
    print STDERR color("magenta"), "\n\n\mMake Smaller v$version\n\n";
    color("reset");
    #preloading
    my @vecs = readPDB($pdbname);
    print STDERR "\n\n";
}

main();

```

```

#!/usr/bin/perl
use strict;
use Getopt::Std;
use Term::ANSIColor;
require '/home/vossman/lib/fixnum.pl';
use constant PI => 3.14159265359;
my %args;
getopt("lor", \%args);

#variables
my $version = "1.0";

#START POINT(S)
#perer active site
my $point = list2vec(71.529,120.46,97.335);
#DIRECTION VECTOR
#REFINE MORE
my $vector = normVec(list2vec(-35.88,37.27,-33.74));

#input pdb
my $pdbname = $args{'i'} || "PDB/lj2-fin.pdb";
my @chks = split('/', $pdbname);
my $pdbname = lc(substr($chks[-2],0,4));
#max radius to "give up"
my $endrad = $args{'r'} || 55;
#output files
my $dbout = $args{'o'} || $pdbname "-small.pdb";
#global system variables (do not modify)
my $maxDist = 80; #length of tunnel (to be removed)

main();

sub main {
    print STDERR color("magenta"), "\n\n\mMake Smaller v$version\n\n";
    color("reset");
    #preloading
    my @vecs = readPDB($pdbname);
    print STDERR "\n\n";
}

main();

```

```

sub main {
    print STDERR color("magenta"), "\n\n\mMake Smaller v$version\n\n";
    color("reset");
    #preloading
    my @vecs = readPDB($pdbname);
    print STDERR "\n\n";
}

main();

```

```

#!/usr/bin/perl
use strict;
use Getopt::Std;
use Term::ANSIColor;
require '/home/vossman/lib/fixnum.pl';
use constant PI => 3.14159265359;
my %args;
getopt("lor", \%args);

#variables
my $version = "1.0";

#START POINT(S)
#perer active site
my $point = list2vec(71.529,120.46,97.335);
#DIRECTION VECTOR
#REFINE MORE
my $vector = normVec(list2vec(-35.88,37.27,-33.74));

#input pdb
my $pdbname = $args{'i'} || "PDB/lj2-fin.pdb";
my @chks = split('/', $pdbname);
my $pdbname = lc(substr($chks[-2],0,4));
#max radius to "give up"
my $endrad = $args{'r'} || 55;
#output files
my $dbout = $args{'o'} || $pdbname "-small.pdb";
#global system variables (do not modify)
my $maxDist = 80; #length of tunnel (to be removed)

main();

sub main {
    print STDERR color("magenta"), "\n\n\mMake Smaller v$version\n\n";
    color("reset");
    #preloading
    my @vecs = readPDB($pdbname);
    print STDERR "\n\n";
}

main();

```

```

#!/usr/bin/perl
use strict;
use Getopt::Std;
use Term::ANSIColor;
require '/home/vossman/lib/fixnum.pl';
use constant PI => 3.14159265359;
my %args;
getopt("lor", \%args);

#variables
my $version = "1.0";

#START POINT(S)
#perer active site
my $point = list2vec(71.529,120.46,97.335);
#DIRECTION VECTOR
#REFINE MORE
my $vector = normVec(list2vec(-35.88,37.27,-33.74));

#input pdb
my $pdbname = $args{'i'} || "PDB/lj2-fin.pdb";
my @chks = split('/', $pdbname);
my $pdbname = lc(substr($chks[-2],0,4));
#max radius to "give up"
my $endrad = $args{'r'} || 55;
#output files
my $dbout = $args{'o'} || $pdbname "-small.pdb";
#global system variables (do not modify)
my $maxDist = 80; #length of tunnel (to be removed)

main();

sub main {
    print STDERR color("magenta"), "\n\n\mMake Smaller v$version\n\n";
    color("reset");
    #preloading
    my @vecs = readPDB($pdbname);
    print STDERR "\n\n";
}

main();

```

```

sub printBar {
    my ($num) = @_;
    my $mult=1;
    while($num/$mult > 76) {
        $mult++;
    }
    $num = int($num/$mult+0.5);
    print STDERR "$_\n";
    for(my $i=1;$i<=$num/2;$i++) {
        if($i==int((($num-2)/2)+0.501)) { print STDERR "\n"; }
        elsif($i==int((($num-2)/4)+0.501)) { print STDERR "#\n"; }
        elsif($i==int(3*(($num-2)/4)+0.501)) { print STDERR "##\n"; }
        elsif($i==0) { print STDERR "+\n"; }
        else { print STDERR "-"; }
        if($i%100==0) { print STDERR "\n"; }
        print STDERR " ";
    }
    print STDERR "\n";
    return $mult;
}

sub getPerpVecs {
    my ($vec0) = @_;
    if($vec0) { $vec0 = $vector; }
    my ($xx,$yy,$zz) = vec2list($vec0);
    my ($x0,$y0,$z0); my ($x1,$y1,$z1);
    if($xx != 0) {
        $x0 = -1 * $xv / $xv;
        $x0 = -1 / sqrt($x0**2 + 1); #unit vector
        $x0 /= sqrt($x0**2 + 1);
        $y0 = 0;
    } else { die "xv = 0 not functional vet"; }
    my $len = sqrt($x0**2 + $y0**2 + $z0**2);
    #print STDERR "($x0,$y0,$z0)\n";
    $len = sqrt($x0**2 + $y0**2 + $z0**2);
    $x1 = ($y0*$zv - $z0*$yv)/$len;
    $y1 = ($x0*$zv - $z0*$xv)/$len;
    $z1 = ($x0*$yv - $y0*$xv)/$len;
    $len = sqrt($x1**2 + $y1**2 + $z1**2);
    #print STDERR "($x1,$y1,$z1)\n";
    $len = fixnum25($x0*$x1 + $y0*$y1 + $z0*$z1);
    my $vec1 = list2vec($x0,$y0,$z0);
    my $vec2 = list2vec($x1,$y1,$z1);
    return ($vec1,$vec2);
}

sub distAlongVec {
    my ($v1,$v2) = @_;
    my ($x1,$y1,$z1) = vec2list($v1);
    my ($x2,$y2,$z2) = vec2list($v2);
    my $ret = sqrt((($x1-$x2)**2+($y1-$y2)**2+($z1-$z2)**2));
    return $ret;
}

sub acos {
    my ($rad) = @_;
    if($rad > 1) { print STDERR "cannot take acos > 1\n"; return 0; }
    my $ret = atan2(sqrt(1 - $rad**2), $rad);
    return $ret;
}

```

```

    return $dist;
};

sub distAwayFromVec {
    my ($pt) = @_;
    my $len = distVec($pt,$point);
    my $t2 = vec2list($pt);
    my @v = vec2list($vector); #static
    my @t1 = vec2list($point); #static
    my $dist = ($t2[0]-@t1[0])*$v[0] +
    ($t2[1]-@t1[1])*$v[1] +
    ($t2[2]-@t1[2])*$v[2];
    #print STDERR "$pt,$dist,$len\n";
    return sqrt($len**2-$dist**2);
};

```

```

#!/usr/bin/perl
use strict;
use Getopt::Std;
require '/home/vossman/lib/fixnum.pl';

my %args;
getopt('r:',['%args']);
my @av = (71, 529, 120, 46, 97, 335);
my @pv = vec2list(normvec(list2vec(-35.88, 37.27, -33.74)));
my @vecs = readPDB($args{r});
my ($crmin, $crmax, $cragv) = (1000, 0, 0);
my ($dotmin, $dotmax, $dotavg) = (1000, 0, 0);
my ($dotmin, $dotmax, $dotavg) = (1000, 0, 0);
my $count=0;
my $vec;
foreach my $vec (@vecs) {
    my @v = vec2list($vec);
    my $dot=$dotlist($vec);
    my $cross=sqrt($dot**2-$dot**2);
    if($cross < $maxcross) {
        $count++;
        if($dot > $dotmax) { $dotmax = $dot; }
        if($dot < $dotmin) { $dotmin = $dot; }
        $dotavg += $dot;
        $dot+=($v[0]-$av[0])*$pv[0];
        $dot+=($v[1]-$av[1])*$pv[1];
        $dot+=($v[2]-$av[2])*$pv[2];
        $dist = sqrt($dist);
        $cross = sqrt($dist**2-$dot**2);
        if($cross < $maxcross) {
            $count++;
            if($dot > $dotmax) { $dotmax = $dot; }
            if($dot < $dotmin) { $dotmin = $dot; }
            $dotavg += $dot;
            $dot+=($v[0]-$av[0])*$pv[0];
            $dot+=($v[1]-$av[1])*$pv[1];
            $dot+=($v[2]-$av[2])*$pv[2];
            $dist = sqrt($dist);
            if($dist > $distmax) { $distmax = $dist; }
            if($dist < $distmin) { $distmin = $dist; }
            $distavg += $dist;
        }
    }
}
print STDERR "\nSummary:\n";
print STDERR "Dot Prod: \", fixnum32($dotmin), "\\", t", fixnum32($dotavg/$count), "\\", t",\n";
print STDERR "Cross Prod: \", fixnum32($dotmax), "\\", t", fixnum32($cragv/$count), "\\", t",\n";
print STDERR "Distance: \", fixnum32($distmin), "\\", t", fixnum32($distavg/$count), "\\", t",\n";
print STDERR "Dist max: \", fixnum32($distmax), "\\", t",\n";
#####
sub readPDB {
    my ($pdbname) = @_;
    print STDERR "Reading in $pdbname\n";
    my @vecs;
    my $numlines = getNumLines($pdbname);
    open(PDB, "< $pdbname") or die "$pdbname does not exist\n";
    my $count; my $spow=1; my $carrot = $cut; my $lines;
    while(defined(my $line = <PDB>)) {
        if($line =~ '/ATOM/ || $line =~ '^HETATM/') {
            my $num = clean(substr($line,6,5));
            my $x = clean(substr($line,30,8));
            my $y = clean(substr($line,38,8));
            my $z = clean(substr($line,46,8));
            push(@vecs,list2vec($x,$y,$z,$num));
            $count++;
        }
        $lines++;
        if($lines > $carrot) { print STDERR "\n"; $carrot += $cut; }
    }
}

```

```
#!/usr/bin/perl
use strict;
use Getopt::Std;

my %args;
getopt("i", \%args);

my $in = $args{i} or die;
open(IN, "< $in") or die;
my $out = "$in.pts";
print STDERR "\nWriting to $out\n";
open(OUT, "> $out\temp") or die;
my $count;
while(defined(my $line = <IN>)) {
    if($line =~ /^ATOM/) {
        $count++;
        my $x = clean(substr($line,30,8));
        my $y = clean(substr($line,38,8));
        my $z = clean(substr($line,46,8));
        print OUT "$x\t$y\t$z\n";
    }
}
close(OUT);
close(IN);
open(TEM, "> temp.txt") or die;
print TEM "3 /home/vossmann/reduce/$out $count D3\n";
print TEM "$count\n";
close(TEM);
system("cat temp.txt $out\temp > $out");
system("rm temp.txt $out\temp");
print STDERR "$count entries\n\n";

sub clean {
    my ($str) = @_;
    $str =~ s/\s+$/ /; $str =~ s/^ \s+ / /;
    return $str;
};
```

```

#!/usr/bin/perl -w
require '/home/vossman/lib/fixnum.pl';
use Getopt::Std;
use strict;

my %args;
getopt('ir', \%args);
my $in = $args{'i'};
my $rot = $args{'r'};

if ($in || $rot || !stat($in) || !stat($rot)) { die "no input files\n"; }

#! Created by ISOMAN V 060801/9.7.2 at Sat Oct 7 15:01:20 2006 for A. Nonymous
#.1sqLrt.m2.1o.m1.r 12 (3FL1.7)
# -0.2177295 -0.9388012 0.2669200
# -0.4691087 -0.1391606 -0.8721075
# 0.8558803 0.3150980 -0.4101004
# 285.8689270 124.3993530 187.0525665

open(ROT, "< $rot") or die;
my @rows;
while(defined(my $line = <ROT>)) {
    $line = clean($line);
    if($line =~ s/^!/ && $line =~ /^[0-9\-\-]/) {
        print STDERR $line, "\n";
        push(@rows,$line);
    }
}

my @a1 = split( ' ', $rows[0]);
my @a2 = split( ' ', $rows[1]);
my @a3 = split( ' ', $rows[2]);
my @t = split( ' ', $rows[3]);

foreach my $item (@a1, @a2, @a3, @t) {
    $item =~ s/\^s+//;
    $item =~ s/\^s+\$/;
    #print STDERR "\n", $item, "\n\t";
}

my $out = $in;
$out =~ s/\^s+//g;
$out = substr($out,0,4) . "-rota.pdb";
open(IN, "< $in") or die;
open(OUT, "> $out") or die;
my $count = 0;
while(defined(my $line = <IN>)) {
    if($line =~ /#ATOM/ || $line =~ /#HETATM/) {
        $count++;
        if($count%1000 == 0) { print STDERR "\n"; }
        my $x = clean(substr($line,30,8));
        my $y = clean(substr($line,38,8));
        my $z = clean(substr($line,46,8));
        $x = $x - $t[0];
        $y = $y - $t[1];
        $z = $z - $t[2];
        my $xp = $a1[0]*$x + $a1[1]*$y + $a1[2]*$z - $t[0];
        my $yp = $a2[0]*$x + $a2[1]*$y + $a2[2]*$z - $t[1];
        my $zp = $a3[0]*$x + $a3[1]*$y + $a3[2]*$z - $t[2];
        my $xp = $a1[0]*$x + $a1[1]*$y + $a1[2]*$z;
        my $yp = $a2[0]*$x + $a2[1]*$y + $a2[2]*$z;
        my $zp = $a3[0]*$x + $a3[1]*$y + $a3[2]*$z;
        if(abs($xp) < 0.001) { $xp = 0.0; }
        if(abs($yp) < 0.001) { $yp = 0.0; }
        if(abs($zp) < 0.001) { $zp = 0.0; }
        substr($line,30,8) = fixnum43($xp);
        substr($line,38,8) = fixnum43($yp);
        substr($line,46,8) = fixnum43($zp);
        substr($line,21,1) = "A";
    }
}
#012345678901234567890123456789
#ATOM 59382 O1P C 9 5

```

```
#!/usr/bin/perl
use strict;
use constant PI => 3.14159265359;
```

```
require '/home/vossman/lib/fxnum.pl';
my %types = ("C4","1","04","1","C2","1","03","1,"C1","1,");
my @list = ("C4","04","C2","03","C1");
```

```
my $infile = "ijj2-rnaonly.pdf";
open(IN, "< $infile") or die;
```

```
my $count = 0;
```

```
my %info;
```

```
my $current = 0;
```

```
while(defined(my $line = <IN>)) {

```

```
if($line =~ /\ATOM/) {

```

```
$count++;

```

```
my $x = clean(substr($line,30,80));

```

```
my $y = clean(substr($line,38,80));

```

```
my $z = clean(substr($line,46,80));

```

```
my $atom = clean(substr($line,32,4));

```

```
if($types{$atom}) {

```

```
#print STDERR "$atom, \"\$z\";
```

```
$info{$atom} = list2vec($x,$y,$z);

```

```
if($rnum != $current) {

```

```
my $check = 0;

```

```
foreach my $a (@list) {

```

```
if($info{$a}) { $check++;

```

```
} if($check == 5) {

```

```
my $v0 = DIHED($info{"C4"},$info{"04"},$info{"C1"},$info{"C2"});

```

```
my $v1 = DIHED($info{"04"},$info{"C1"},$info{"C2"},$info{"C3"});

```

```
my $v2 = DIHED($info{"C1"},$info{"C2"},$info{"C3"},$info{"C4"});

```

```
my $v3 = DIHED($info{"C2"},$info{"C3"},$info{"C4"},$info{"04"});

```

```
my $v4 = DIHED($info{"C3"},$info{"C4"},$info{"04"},$info{"C1"});

```

```
print STDOUT $current."\"t\",fixnum33($v3),\"t\",fixnum33($v1),\"t\",fixnum33($v2),

```

```
\"t\",fixnum33($v3),\"t\",fixnum33($v4),\"t\",fixnum33($v1+$v2+$v3+$v4),\"t\",

```

```
my $p = getP($v0,$v1,$v2,$v3,$v4,$v5);

```

```
#print STDOUT '$current result',fixnum33($p),"\n";

```

```
print STDOUT fixnum33($p),"\n";

```

```
$current = $rnum;

```

```
}

```

```
close(IN);

```

```
print STDERR "COUNT: $count\n\n";

```

```
sub clean {

```

```
my ($str) = @_;

```

```
my $stop = ($v4+$v1) - ($v3+$v0);

```

```
my $bot = atan2($stop*PI/180,$bot*PI/180) + sin(72*PI/180);

```

```
if($p < -50) {

```

```
$p += 360;

```

```
if($p > 310) {

```

```
$p -= 360;

```

```
return $p;

```

```
};

```

```
sub getP {

```

```
my ($v0,$v1,$v2,$v3,$v4) = @_;

```

```
my $stop = ($v4+$v1) - ($v3+$v0);

```

```
my $bot = atan2($stop*PI/180,$bot*PI/180) + sin(72*PI/180);

```

```
return $str;

```

```
#!/usr/bin/perl
use strict;
use constant PI => 3.14159265359;
require '/home/vossman/lib/fxnum.pl';
my (@DO) = vec2list($vec);
my $TOR;
foreach my $foo (@DO) {
    if(($foo && !/\d/) || $foo == 0) {
        $TOR = 400;
    }
}
my $SX = ($DO[0] - $DO[5]);
my $SPY = ($DO[1] - $DO[4]);
my $SPZ = ($DO[2] - $DO[5]);
my $SOX = ($DO[6] - $DO[3]);
my $SRY = ($DO[7] - $DO[4]);
my $SOZ = ($DO[8] - $DO[5]);
my $SRX = ($DO[9] - $DO[6]);
my $SRV = ($DO[10] - $DO[5]);
my $SRZ = ($DO[11] - $DO[15]);
my $SLNX = ($SOY * $SRX) - ($SOZ * $SRZ);
my $SLNZ = ($SOX * $SRY) - ($SOY * $SRX);
my $SANK = ($SOY * $SPZ) - ($SOZ * $SPY);
my $SNY = ($SOZ * $SPX) - ($SOX * $SPZ);
my $SNWZ = ($SOX * $SPY) - ($SOY * $SPX);
my $SCOS = ((SLNX*$MNXY)+($LNZ*$MNZZ))/(((SLNX*$2+$LNZ*$2)*($MNXY*$2+$MNZZ*$2)+($MNXY*$2+$LNZ*$2)*($MNZZ*$2)+($MNXY*$2+$LNZ*$2)*($MNYY*$2+$MNZZ*$2))/(((SLNX*$2+$LNZ*$2)*($MNYY*$2+$MNZZ*$2))/(((SLNUCOS*($RZ))+(($MNXY*$RZ))+($MNYY*$RZ))+($MNZ*$RZ)));
my $SSIN = (1 - ($SCOS*$2))*((1/2));
my $SNICOS = ((SMNXY*$SPX)+(SMNYY*$SPY)+(SMNYY*$RZ));
my $SRZ**2)*((1/2));
$TOR = (atan2($STIN, $COS)*(180/PI))*($NUCOS / ($NUCOS / ($NUCOS * $2)*(1/2)));
# if($TOR < -100) {
#     $TOR = 360 +
# };
# if($TOR > 260 && $TOR != 400) {
#     $TOR = $TOR - 360;
# };
# return $TOR;
};

sub vec2list {
    my ($vec) = @_;
    my (@list) = @_;
    return split(' ', $vec);
};

sub list2vec {
    my ($vec) = @_;
    my (@list) = @_;
    foreach my $item (@list) {
        $vec .= $item." ";
    }
    return $vec;
};

sub clean {
    my ($str) = @_;
    $str =~ s/^\$/$/;
    $str =~ s/^\$/$/;
    return $str;
};
```

```

#!/usr/bin/perl
use strict;
use Getopt::Std;
use Term::ANSIColor;
#use Math::Trig;
require '/home/vossman/1ib/fixnum.p1';
my %args;
getopt("1",\%args);

#variables
my $version = "2.9.24";
my $probeSize
my $probekad = 2.4; #dynamic
#START POINT(S)
my $point = List2vec(71.529,120.46,97.335,$probeRad);
my $vector = normVec(List2vec(-35.88,37.27,-33.74));
#DIRECTION VECTOR
my $planeBtwPlanes
$distance btw planes
my $planeBens = 0.2;
#radial resolution for drawing arcs
my $radiAlkes = 0.15*$probekad; #rough
my $wobMaxTilt = 15; #max wobble tilt in degrees
my $woBnTilts = 3; #number of tilts to do
my $woBnRots = 5; #spins about each title
#output files
my $pdfout = "tunneL-$Spdbid.".pdbs";
my $pdbsurf = "surface-$Spdbid.".pdbs";
my $pdcenter = "central-$Spdbid.".pdbs";
my $datafile = "output-$Spdbid.dat";
my $woBcenter = "wobcentral-$Spdbid.".pdbs";
#glob variables (do not modify)
my $activePoint = List2vec(71.529,120.46,97.335,$probeRad); #active site
my $atom_VDWr;
my $default_VDWr = 1.6; #for undefined atoms
my $strongGrid = 6.0; #speedy atom lookup grid size >=4
my $gridXYZ;
my $gridXYZZ;
# gimme xyz get atoms
my $maxCrawl = 2.5; #initial jump out of gate, avg min size 3.6
my $maxDist = 80; #length of tunnel (to be removed)
my $guesses = 8; #plane resolution $initCrawl*(2**-$guess)
my ($xmin,$ymax,$zmin) = (100,100,100); #setup grid
my ($xmax,$ymax,$zmax) = (0,0,0); #setup grid
my $printDataFiles = 0;
main();
#preloadings
loadDB();
my @vewc = readPDB($Spdbin);
print STDERR color("yellow");
fillRoughGrid(@vewcs);
print STDERR color("reset");
#the 'beef'

```

```

sub readPath {
    my ($hash,$incr) = @_;
    my $pathfile = "PDB/with_path.pdb";
    my $distfrom; my @pts;
    my $maxd;
    open(PATH, "< $pathfile") or
        die "cannot open $pathfile for reading path info\n";
    while(defined(my $line = <PATH>)) {
        if($line =~ /ATOM/ || $line =~ /^HETATM/) {
            my $x = clean(substr($line,30,8));
            my $y = clean(substr($line,38,8));
            my $z = clean(substr($line,46,8));
            my $pt = listvec($x,$y,$z);
            push(@pts,$pt);
            my $dist = distAlongVec($pt);
            if($dist > $maxd) { $maxd = $dist; }
            $distfrom{$pt} = $dist;
        }
    }
    close(PATH);
    @pts = sort { $distfrom[$a] <> $distfrom[$b] } @pts;
    my $i=0;
    my $d=0; $d+=0;
    if($TESTPDB, "> testpath.pdb") or die;
    #open(CTE,STPDB, "> testpath.pdb");
    for(my $j=0; $d<=$maxd; $d+=$incr) {
        while($d > $distfrom{ $pts[$i-1] }) { $i++; }
        $pts[$i] => $distfrom; $pts[$i-1] <- $distfrom
        #distance for each increment
        my $dvec = normVec(subVec($pts[$i-1],$pts[$i]));
        my ($dx,$dy,$dz) = vec2list($dvec);
        #start point
        my ($x0,$y0,$z0) = vec2list($pts[$i-1]);
        #distance for start point
        my $d0 = distAlongVec($pts[$i-1]);
        #distance for each increment
        my $dd = distAlongVec(list2vec($x0+$dx, $y0+$dy, $z0+$dz)) - $d0;
        #number of increments
        my $dm = ($dd-$d0)/$dd;
        my $x = $x0 + $dx*$dm;
        my $y = $y0 + $dy*$dm;
        my $z = $z0 + $dz*$dm;
        my $vec = list2vec($x,$y,$z);
        $hash{fixnum3($d)} = $vec;
        #print TESTPDB vec2pdb($vec,$d,$d,$d);
    }
    #close(TESTPDB);
    return;
}

#changeRadii {
    my ($min) = @_;
    my $probeRad=2;
    if($probeRad < $min/1.1-0.2) {
        my $oldVDW = $probeRad;
        $probeRad += 0.2;
        print STDERR color("red"), "mindia > probeRad, increasing size ";
        "VDW: $oldVDW -> $probeRad and moving on\",color('reset')");
    } elsif($min > 0.1 && ($probeRad > 2.5*$min || $min < 0.9)) {
        my $oldVDW = $probeRad;
        $probeRad -= 0.2;
        print STDERR color("red"), "probeRad > mindia, decreasing size ";
        "VDW: $oldVDW -> $probeRad and moving on\",color('reset')");
    } else {
        my $oldVDW = $probeRad - $default_VDW;
        print STDERR color("red"), "probeRad > default_VDW ";
        $oldVDW -> $probeRad and moving on\",color('reset'));
    }
    if($probeRad > $roughGrid - $default_VDW) {
        print STDERR color("red"), "probeRad > roughGrid... reset to $roughGrid\n",color('reset'));
        $probeRad = $roughGrid - $default_VDW;
    }
    return;
}

sub normVec {
    my ($vec) = @_;
    my ($x,$y,$z) = vec2list($vec);
    my $len = sqrt($x**2 + $y**2 + $z**2);
    my $x/$len; $y/$len; $z/$len;
    return list2vec($x,$y,$z);
}

sub clkPoint {
    my ($vec) = @_;
    my ($x,$y,$z,$r) = vec2list($vec);
    if($r) {
        my $probeRad;
        $r = $probeRad;
        $vec = list2vec($x,$y,$z,$r);
    }
    #BIG assumption for drugs
    my $key = xyz2Key($x,$y,$z);
    my $good=1;
    #my $minDist = 1000;
    my $checks = 0;
    my @keys;
    my @keys = varyKey($key,$x,$y,$z);
    foreach my $key (@keys) {
        my $i=0;$i<=$#{$gridXYZ{$key}} && $good;$i++);
        my $pvec = ${$gridXYZ{$key}}{$i};
        $pvec++;
        #my ($dist,$sum) = no Clash($pvec,$vec);
        my ($dist,$sum) = no Clash($pvec,$vec);
        if($dist || !$sum) { $good=0; }
        if($dist) { $good=0; }
        #else if($dist<$minDist) { $minDist=$dist; }
    }
    if($good && $checks) {
        return 1;
    }
    #print OUT vec2pdb($pvec,$aout,500/$minDist,$minAtom/10);
}

```

```

return @keys;
};

sub loadVDWr {
    my %type2rad;
    print STDERR "Loading atomdefs";
    open(ATOM, "< $atomdefs") or die;
    my $count;
    while(defined(my $line = <ATOM>)) {
        my ($type,$rad) = split(' ',clean($line));
        $type2rad{$type} = $rad;
        print STDERR "$type $rad\n";
        $count++;
    }
    close(ATOM);
    print STDERR " $count types\\nloading residefs";
    my $curres;
    $count = 0;
    while(defined(my $line = <RESI>)) {
        my $line2 = clean($line);
        if($line2) {
            if($curres) {
                substr($curres, 0,1) ne "\n" ) {
                ($curres) = split(' ', $line2);
                print STDERR "$curres\n";
            } else {
                my ($atom,$type) = split(' ', $line2);
                $atom_VDWr{$curres}->{$atom} = $type;
                $count++;
            }
        } else {
            print STDERR "$count atoms\n";
        }
    }
    close(RESI);
    print STDERR " $count atoms\n";
}

sub clean {
    my ($str) = @_;
    $str =~ s/\s+//; $str =~ s/^\s+//;
    return $str;
}

sub readDB {
    my ($pdbin) = @_;
    print STDERR "Reading in $pdbin...\n";
    my $count;
    my $numlines = getNumLines($pdbin);
    print STDERR "[ $numlines ]\n";
    my @vecs;
    my $cut = printBar($numlines);
    open(PDB, "< $pdbin") or die "pdbin does not exist\n";
    my $line =~ '/ATOM/ || $line =~ '/HETATM/' {
        if($line == '/ATOM/ || $line =~ '/HETATM/' ) {
            my $num = clean(substr($line,6,5));
            my $resi = clean(substr($line,17,3));
            $resi = one2three($resi);
            my $atom = clean(substr($line,13,3));
            $atom = fixSugation($atom);
            my $rad;
            if($atom_VDWr{$resi}->{$atom}) {
                $rad = $atom_VDWr{$resi}->{$atom};
            } else {
                $rad = $default_VDWr;
            }
        }
    }
}

```

```

$PROTRID{$key} = 1;
push(@{@GridXYZ{$key}},$vec);
}
print STDERR "\n";
};

sub xyz2key {
my ($x,$y,$z) = @_;
my $key = fixnum2ob((($x-$xmin)/$roughGrid+1).fixnum2ob((($y-$ymin)/$roughGrid+1).fixnum2ob((($z-$zmin)/$roughGrid+1));
if ($x < $xmin || $y < $ymin || $z < $zmin) { $key = 999999; }
if (length($key) == 6) { return $key; }
die;
};

sub noclash {
my ($vec1,$vec2) = @_;
my ($x1,$y1,$z1,$r1,$anum) = vec2list($vec1); #protein/rna
my ($x2,$y2,$z2,$r2) = vec2list($vec2); #water
my $distsq = ($x1*$x2)**2 + ($y1*$y2)**2 + ($z1*$z2)**2;
if ($distsq < ($r1 + $r2)**2) { return 0; }
return ($distsq,$anum);
};

sub printBar {
my ($num) = @_;
my $mult=1;
my $per <= 100;
if ($num/$mult > 76) {
    $mult++;
}
$num = int($num/$mult+0.5);
print STDERR "#";
for(my $i=1;$i<=$num-2;$i++) {
    if($i==int((($num-2)/2+0.501)) { print STDERR "\n"; }
    elsif($i==int(3*(($num-2)/4+0.501)) { print STDERR "#"; }
    elsif($i==int(5*(($num-2)/4+0.501)) { print STDERR "#"; }
    elsif($i==0) { print STDERR "+"; }
    else { print STDERR "."; }
    if ($%100==0) { print STDERR "\n"; }
    print STDERR "#";
}
print STDERR "\n";
return $mult;
};

sub printVec {
my (@list) = @_;
my $line;
if (@list == 0) { #list
my $vec = @list[0];
@list = vec2list($vec);
}
$line = "(";
foreach my $num (@list) { $line .= " "; }
$line .= ")";
return $line;
};

sub vec2bdp {
#TYPE--ATOM# Atm ResChRes#
#ATOM 1 05* U 0 10 16.071 148.494 104.415 1.00 83.83
#HETATM91176 0 HOH 496
my ($vec,$atomnum,$temp,$resnum,$chain) = @_;
my ($x,$y,$z) = vec2list($vec);

```

```

my @Vi = vec2List(@vecs[$i]);
my @Vj = vec2List(@vecs[$j]);
my @Vk = vec2List(@vecs[$k]);
if($coord == 1) {
    $area += ($Vi[1] * ($Vj[2] - $Vk[2]));
}
elsif($coord == 2) {
    $area += ($Vi[0] * ($Vj[2] - $Vk[2]));
}
elsif($coord == 3) {
    $area += ($Vi[0] * ($Vj[1] - $Vk[1]));
}
my $an = sqrt($ax**2 + $ay**2 + $az**2);
if($coord == 1) {
    $area *= ($an / (2*$ax));
}
elsif($coord == 2) {
    $area *= ($an / (2*$ay));
}
elsif($coord == 3) {
    $area *= ($an / (2*$az));
}
return abs($area);

sub get2DArea {
    my (@vecs) = @_;
    my $area=0;
    #my ($i,$j,$k)=(1,2,0); $i<=$n; $i++,$j++,$k++;
    for(my ($i,$j,$k)=1,(2,0); $i<=$n; $i++,$j++,$k++) {
        my @Vi = vec2List(@vecs[$i]);
        my @Vj = vec2List(@vecs[$j]);
        my @Vk = vec2List(@vecs[$k]);
        $area += $Vi[0] * ($Vj[1] - $Vk[1]);
        $area += $Vi[0] * ($Vj[2] - $Vk[2]);
        $area += $Vi[1] * ($Vj[2] - $Vk[1]);
        return abs($area/2);
    }
}

sub getCenter { #battle weary
    my (@vecs) = @_;
    my ($xp,$yp,$zp) = vec2List($pt);
    my $xs = $xp# + $planeDens*$xy*$len/2;
    my $yp# + $planeDens*$yz*$len/2;
    my $zs = $zp# + $planeDens*$xz*$len/2;
    my $count = 1;
    foreach my $vec (@vecs) {
        my ($x2,$y2,$z2,$ind) = vec2List($vec);
        while($ind >= 1) { #while of if
            $xs+=$x2; $ys+=$y2; $zs+=$z2;
            $count++;
            #$ind--;
            #$ind=2;
            $ind=0;
        }
    }
    $xs-=$count; $ys-=$count; $zs-=$count;
    return list2vec($xs,$ys,$zs);
};

sub get2dCenter { #battle weary
    my (@vecs) = @_;
    my ($xp,$yp) = vec2List($pt);
    my $xs = 0; #$xp# + $planeDens*$xy*$len/2;
    my $ys = 0; #$yp# + $planeDens*$yz*$len/2;
    my $count = 0;
    foreach my $vec (@vecs) {
        my ($x2,$y2) = vec2List($vec);
        $xs+=$x2; $ys+=$y2;
        $count++;
    }
    $xs-=$count; $ys-=$count;
    return list2vec($xs,$ys);
};

sub getBoxLimits {
    my ($resi) = @_;
    $resi =~ $/\^+/;
    if(length($resi) > 1) { return $resi; }
    if($resi eq "A") { return "ADE"; }
    if($resi eq "C") { return "CYT"; }
    if($resi eq "U") { return "UR1"; }
    if($resi eq "G") { return "GUA"; }
};

sub fixSugAtom {
    my ($atom) = @_;
    if($atom =~ /\^*/;/) { return $atom; }
    $atom =~ $/\^$/\^/;
    return $atom;
};

sub fixBoxLimits {
    my ($pt,$vec) = @_;
    my ($sx,$sy,$sz,$zn) = (-1,1000,-1,1000);
    my $apln = $endRad-1/2; #don't look beyond this point
    my $dprp = 1.8*$probeRad; #don't look beyond this point
    my ($vec1,$vec2) = getPerpVecs($vec);
    my ($xp,$yp,$zp) = vec2List($vec);
    my ($xr,$yr,$zr) = vec2List($vec);
    my ($xrl,$yrl,$zrl) = vec2List($vec1);
    my ($xrv,$yrv,$zrv) = vec2List($vec2);
    foreach my $i (-1,0,1) {
        foreach my $j (-1,0,1) {
            foreach my $k (-1,0,1) {
                my $x = $xp + $i*$xrl*$dprp + $j*$xrv*$dprp + $k*$yrl*$dprp;
                my $y = $yp + $i*$yrl*$dprp + $j*$yrv*$dprp + $k*$zrl*$dprp;
            }
        }
    }
};

```

```

my $z = $xp + $y*$zv1*$dpPp + $j*$zv1*$dpIn + $k*$zv2*$dpIn + $l*$zv2*$dpJn;
my $key = xv2Key($x,$y,$z);
$key =~ m/([0-9][0-9-]([0-9][0-9-]([0-9][0-9-]))(0-9))/;
my $yk = $1; my $zk = $2; my $sk = $3;
if ($sk > $skc) { $skc = $sk; }
if ($yk > $ykc) { $ykc = $yk; }
if ($sk < $skc) { $skc = $sk; }
if ($yk < $ykc) { $ykc = $yk; }
if ($yk < $skn) { $skn = $yk; }
if ($sk < $skn) { $skn = $sk; }
if ($yk < $zkn) { $zkn = $yk; }
if ($sk < $zkn) { $zkn = $sk; }

#print STDERR "(skc,$skn,$ykc,$ykn,$zkx,$zkn)\n";
return ($skn,$ykn,$ykc,$ykn,$zkx,$zkn);
```

```

sub getPointsWithinPlane {
    my ($pt,$vec) = @_;
    my @xyzr;
    my $count = 0;
    my $count = ($skx-$skn)*($ykc-$ykn)*($zkx-$zkn)/$roughGrid**3;
    print STDERR "Count > ";
    print STDERR "$count\n";
    my $dpr = 2.5*$probeRad; #don't look beyond this point
    #for my $i=>$skn; $i<=$ykc; $i++ {
    #for my $i=>$ykc; $i<=$zkn; $i++ {
    #for my $i=>$ykc; $i<=$zkn; $i++ {
    #for my $i=>$ykc; $i<=$zkn; $i++ {
    #    my $key = fixnum20b($i).fixnum20b($j).fixnum20b($k);
    #    if ($gridXYZ($key) >-1) {
    push(@xyzr,@$gridXYZ($key));
    }
    }
    }
    #print STDERR "#XYZR > ";
    #print STDERR "Got me $xyzr entries\tex: \"$xyzr[$xyzr2]\",\"\\n\";
    my ($xp,$yp,$zp) = vec2list($pt);
    my ($xv,$yv,$zv) = vec2list($vec);
    foreach my $v (@xyzr2) {
        my $cutoff = $endRad1.2;
        foreach my $v (@AllXYZ) {
            my ($x,$y,$z,$r) = vec2list($v);
            my $dx = $x-$xp; my $dy = $y-$yp; my $dz = $z-$zp;
            if ($dx < $cutoff && $dy < $cutoff && $dz < $cutoff) {
                my $dot = fixnum35($xv*$dx + $yv*$dy + $zv*$dz);
                if($dot <= $r + $probeRad) {
                    push(@xyzr,$v.$dot." ");
                }
            }
        }
    }
    print STDERR "#XYZR > ";
    #print STDERR "Got me $xyzr entries\tex: \"$xyzr[$xyzr]\",\"\\n\";
    return @xyzr;
}

sub genCircles {
    my ($pt,$vec,$vx,$vy) = @_;
    my @xyzr; #circles to return
    my $xyzr = getPointsWithinPlane($pt,$vec);
    my ($vec1,$vec2) = getPerpVecs($vec);
    my ($xp,$yp,$zp) = vec2list($pt);
    my ($xv,$yv,$zv) = vec2list($vec);
    my ($xxy,$yyz,$zzx) = vec2list($xyzr);
    my ($yyx,$zyx,$xxz) = vec2list($xyzr);
    foreach my $v (@xyzr) {
        my ($xv,$yv,$zv,$rv) = vec2list($v);
        my $dist = fixnum35($qr(( $x0-$xp)**2 + ($y0-$yp)**2 + ($z0-$zp)**2));
        my $x = fixnum35(( $x0-$xp)*$xxz + ($y0-$yp)*$yyz + ($z0-$zp)*$zrz);
        my $y = fixnum35(( $x0-$xp)*$xxy + ($y0-$yp)*$yyx + ($z0-$zp)*$zyx);
        my $d = fixnum35($qr(( $x+$probRad)**2 - $q0**2));
    }
}
```

```

sub intArcs {
    my ($v1,$v2) = @_;
    #needs to return legit vectors
    my ($x1,$y1,$r1,$th1,$th12) = vec2list($v1);
    my ($x2,$y2,$r2,$th2,$th22) = vec2list($v2);
    my $d = $r1*$r2 * print STDERR "missing rad\n"; return 0;
    if($d < 0.0001) {
        #print STDERR color("red"),"same_center",color("reset");
        return ($v1,$v2);
    }
    my $qout = ($r1**2 + $r2**2 - $r1*$r2)/(2*$r1*$r2);
    if(abs($qout) >= 1) {
        #print STDERR color("red"),"inside",color("reset");
        return ($v1,$v2);
    }
    my $phi11 = fixnum35(acos($qout));
    my $phi1 = fixnum35(atan2($y2-$y1,$x2-$x1));
    foreach my $phi1 ( $phi1+$dphi1,$phi1-$dphi1) {
        while($phi1 > $th12) { $phi1 -= 2*pi; }
        if($phi1 < $th12 && $phi1 > $th11) {
            #print STDERR "$th11 -- $th12 > ";
            #print STDERR "phi ";
            #print STDERR color("green"),"good ",color("reset");
            $v1 = list2vec($x1,$y1,$r1,$phi,$th12);
            my $xp = $r1*cos($phi)-$xi;
            my $yp = $r1*sin($phi)+$yi;
            my $phi2 = atan2($yp-$y2,$xp-$x2);
            $phi2 = fixnum35($phi2);
            my $thh = $th22 - $th21;
            my $dthh = $phi2 - $th21;
            my $dthh35 = atan2($y2-$y1,$x2-$x1);
            #print STDERR " $dth > $dth2 ";
            if($dthh > 0 && $dthh < $thh) {
                $v2 = list2vec($x2,$y2,$r2,$th21,$phi2);
            } else {
                #print STDERR color("yellow"),"\n",$v2,"\'t > $phi2\n";
            }
            #print STDERR $v2,"\n",color("reset");
        } else {
            #print STDERR color("red"),"bad ",color("reset");
        }
    }
    #print STDERR color("yellow"),"\n",$v1,"\'t > $phi1\n";
    #print STDERR $v1,"\n",color("reset");
}
#my $xy1 = fixnum35($r1*cos($phi+$dphi)+$x1); my $fixnum35($r1*$sin($phi+$dphi)+$y1); my $xy2 = fixnum35($r1*cos($phi-$dphi)+$x1); my $fixnum35($r1*$sin($phi-$dphi)+$y1);
return ($v1,$v2);
}

sub nextCircle {
    my ($centid,$scrpct,$scrc) = @_;
    my ($a1,$a2) = intersect($centid,$scrc);
    foreach my $in (@ints) {
        my $start_circ, $start_pt, $lost_of_circs
        while($a1 < $currang) { $a1 += 2pi; }
        while($a2 < $currang) { $a2 += 2pi; }
        if($a1 < $bestang) { $bestang = $a1; $bestcirc = $in; }
        if($a2 < $bestang) { $bestang = $a2; $bestcirc = $in; }
    }
    my $scrrng = fixnum35(atan2($yc,$xp-$xc)-0.0001);
    my $bestang = 400; my $bestcirc;
    my @ints = get2ints($centid,$scrc);
    my $ints = get2ints($cent,$scrrng);
}

sub isSect {
    my ($scirc1,$isect1,$scirc2,$isect2) = @_;
    my ($x1,$y1) = vec2list($isect1);
    my ($x2,$y2) = vec2list($isect2);
    my ($xcl,$ycl,$scrl) = vec2list($scirc1);
    my ($xcl2,$ycl2,$scrl2) = vec2list($scirc2);
    my $th1 = fixnum35(atan2($y1-$ycl,$x1-$xcl));
    my $th2 = atan2($y2-$ycl,$x2-$xcl);
    while($th2 < $th1) { $th2 += 2pi; }
    my $rad = fixnum35($th2);
    my $thh = fixnum35($rcl-$probeRad);
    # print STDERR color("blue"),"IN: ($sccl,$ycl,$rad,$th1,$th2)\n";
    # color("red");
    return ($sccl,$ycl,$rad,$th1,$th2,$rad);
};

sub isSectFo {
    my ($scirc1,$isect1,$scirc2,$isect2) = @_;
    my ($x1,$y1) = vec2list($isect1);
    my ($x2,$y2) = vec2list($isect2);
    my ($xcl,$ycl,$scrl) = vec2list($scirc1);
    my ($xcl2,$ycl2,$scrl2) = vec2list($scirc2);
    my $th1 = fixnum35($rcl-$scrl);
    my $th2 = atan2($y1-$ycl,$x1-$xcl);
    while($th2 < $th1) { $th2 += 2pi; }
    my $rad = fixnum35($th2);
    my $thh = atan2($y2-$ycl,$x2-$xcl);
    # print STDERR color("blue"),"IN: ($sccl,$ycl,$rad,$th1,$th2)\n";
    # color("red");
    return ($sccl,$ycl,$rad,$th1,$th2,$rad);
};

sub writeSurf2PDB {
    my ($isiperp,$pt,$sx,$sy,$surf) = @_;
    if($isiperp) { open(PERP,>> $pdbsurf) or die; }
    open(SURF,>> $pdbsurf) or die;
    my $count;
    foreach my $xy (@$surf) {
        $count++;
        my ($x,$y,$n) = vec2list($xy);
        my $xyz = d2intm3($x,$y,$z,$n);
        #xyz,atomnum,temp,resnum,chain
        print SURF vec2pdh($xyz,$count,$x*$y*2*$y*$z/3,$n,"HH");
        if($isiperp) { print PERP vec2pdh($xyz,$count,$n,$n,"HH"); }
    }
    close(SURE);
    if($isiperp) { close(PERP); }
}

sub printPath {
    my ($paths) = @_;
    if($printDATfiles) { open(PATH, "> path.dat") or die; }
    my $path = $radialRes;
    my $pts;
    my $count=0;
    foreach my $path (@$paths) {
        $count++;
        my ($x,$y,$r,$th1,$th2) = split(",",$path);
        while($th2 < $th1) { $th2 += 2pi; }
        for(my $th=$th1; $th<$th2; $th+=$dth/$r) {
            my $xp = fixnum35($r*cos($th+$x));
            my $yp = fixnum35($r*sin($th+$y));
            if($printDATfiles) { print PATH "$xp\t$yp\n"; }
            push(@pts, $xp,$yp,$count,"");
        }
        print PATH "\n";
    }
    if($printDATfiles) { close(PATH); }
    return @pts;
}

sub circFo {
    my ($isect1,$scirc1,$isect2) = @_;
    my ($x1,$y1) = vec2list($isect1);
    my ($x2,$y2) = vec2list($isect2);
    my ($xcl,$ycl,$scrl) = vec2list($scirc1);
    my ($xcl2,$ycl2,$scrl2) = vec2list($scirc2);
    my $th1 = fixnum35(atan2($y1-$ycl,$x1-$xcl));
    my $th2 = atan2($y2-$ycl,$x2-$xcl);
    while($th2 < $th1) { $th2 += 2pi; }
    my $rad = fixnum35($th2);
    my $thh = fixnum35($rcl-$probeRad);
    # print STDERR color("blue"),"IN: ($sccl,$ycl,$rad,$th1,$th2)\n";
    # color("red");
    return ($sccl,$ycl,$rad,$th1,$th2,$rad);
};

sub isSect {
    my ($scirc1,$isect1,$scirc2,$isect2) = @_;
    my ($x1,$y1) = vec2list($isect1);
    my ($x2,$y2) = vec2list($isect2);
    my ($xcl,$ycl,$scrl) = vec2list($scirc1);
    my ($xcl2,$ycl2,$scrl2) = vec2list($scirc2);
    my $th1 = fixnum35($rcl-$scrl);
    my $th2 = atan2($y1-$ycl,$x1-$xcl);
    while($th2 < $th1) { $th2 += 2pi; }
    my $rad = fixnum35($th2);
    my $thh = atan2($y2-$ycl,$x2-$xcl);
    # print STDERR color("blue"),"IN: ($sccl,$ycl,$rad,$th1,$th2)\n";
    # color("red");
    return ($sccl,$ycl,$rad,$th1,$th2,$rad);
};

sub writeSurf {
    my ($isiperp,$pt,$sx,$sy,$surf) = @_;
    if($isiperp) { open(PERP,>> $pdbsurf) or die; }
    open(SURF,>> $pdbsurf) or die;
    my $count;
    foreach my $xy (@$surf) {
        $count++;
        my ($x,$y,$n) = vec2list($xy);
        my $xyz = d2intm3($x,$y,$z,$n);
        #xyz,atomnum,temp,resnum,chain
        print SURF vec2pdh($xyz,$count,$x*$y*2*$y*$z/3,$n,"HH");
        if($isiperp) { print PERP vec2pdh($xyz,$count,$n,$n,"HH"); }
    }
    close(SURE);
    if($isiperp) { close(PERP); }
}

```

```

my $rad = fixnum35($probeRad);
#printf STDERR color('green'), "OUT: ($xi1,$yi1,$rad,$th1,$th2)\n",
#    color("reset");
return "$xi1,$yi1,$rad,$th1,$th2";
};

#we have a negative intersect
# my ($xi1,$yi1) = vec2list($lastisect);
# my ($xi2,$yi2) = vec2list($isect);
# my $circ1 = list2vec($xi1,$yi1,$probeRad);
# my $circ2 = list2vec($xi2,$yi2,$probeRad);
# my ($p1,$p2) = intersectXY($circ1,$circ2);
# foreach my $p (Spl($p1,$p2)) {
#     my ($xp,$yp) = vec2list($p);
#     my ($xpl,$ypl,$rip,$rlip,$th1lp) = vec2list($paths[$#paths-1]);
#     my $th1 = fixnum35(atan2($yp-$ypl,$xp-$xpl));
#     printf STDERR "angle: $th1\n";
#     while($th1 < $th1lp) { $th1 += 2*pi; }
#     if($th1 > $th1lp && $th1 < $th2lp) {
#         $paths[$paths-1] = "$xpl $ypl $rip $rlip $th1lp ";
#         my $th2 = fixnum35(catan2($yp-$yi2,$xp-$xi2));
#         push(@paths,"$xi,$yi,$ri,$th1,$th2,");
#     }
#     #print STDERR color("reset");
#     #} else {
#     #    $Firstcirc = $circs[$#circs-1];
# }

sub genPath {
    #alias sub getpath
    my ($xxr,$startCirc) = @_;
    my @paths; #list of x,y,radius,theta,theta2 vectors
    my @circs; #list of circles
    my $newiSect = "0.0.0"; #keep off last intersect pt
    my $newcirc = $startCirc; #changes s
    my $loopcount = 0;
    my $circcount = 1000;
    my $death=0;
    print STDERR " ";
    #new is old
    my $oldcirc = $newcirc;
    my $oldiSect = $newiSect;
    push(@circs,$oldcirc);
    #give circle and point (w/ list)
    #provides new circle and point (move newcirc to oldcirc)
    ($newiSect,$newcirc) = nextCircle($oldcirc,$oldiSect,$xxr);
    $circcount++; #if($circcount>$newcirc) > 3) {
    print STDERR color('red'), "negative intercept\n",color("reset");
    $death=1;
}

my ($ciarc,$ciRad) = circFo($oldiSect,$xxr,$oldcirc,$newiSect);
my $iSarc = iSectFo($xxr[$oldcirc],$newiSect,$xxr[$newCirc]);
if($ciRad > 0) {
    push(@paths,$ciArc);
    push(@paths,$iArc);
    #print STDERR "$ciArc\n$iArc\n=====\\n";
} else {
    #print STDERR color("red"), "negative intercept\\n",color("reset");
    push(@paths,$iArc);
    #print STDERR "$iSarc\\n=====\\n";
}
#time to stop?
if($newcirc==$startCirc) { $loopcount = 2; }

$loopcount--;
} #end loop over path

```

```

#we have a negative intersect
# my ($xi1,$yi1) = vec2list($lastisect);
# my ($xi2,$yi2) = vec2list($isect);
# my $circ1 = list2vec($xi1,$yi1,$probeRad);
# my $circ2 = list2vec($xi2,$yi2,$probeRad);
# my ($p1,$p2) = intersectXY($circ1,$circ2);
# foreach my $p (Spl($p1,$p2)) {
#     my ($xp,$yp) = vec2list($p);
#     my ($xpl,$ypl,$rip,$rlip,$th1lp) = vec2list($paths[$#paths-1]);
#     my $th1 = fixnum35(atan2($yp-$ypl,$xp-$xpl));
#     my $th2 = fixnum35(catan2($yp-$yi2,$xp-$xi2));
#     push(@paths,"$xi,$yi,$ri,$th1,$th2,");
# }

sub trimPaths {
    my (@paths) = @_;
    #print STDERR "trimming things...\\r";
    for(my $i=0;$i<@#paths;$i++) {
        my ($xi1,$yi1,$th1,$th2) = vec2list($paths[$i]);
        for(my $j=$i+1;$j<@#paths;$j++) {
            my ($xi2,$yi2,$th2,$th1) = vec2list($paths[$j]);
            my $distsq = ($xi1-$xi2)**2 + ($yi1-$yi2)**2;
            my $radsq = ($ri1-$ri2)**2 + ($y1-$y2)**2;
            if($distsq < $radsq) {
                #print STDERR "\\ns1->j ";
                ($paths[$i],$paths[$j]) = intArcs($paths[$i],$paths[$j]);
            }
        }
    }
    #print STDERR "done\\n";
}

sub trimPaths {
    my (@paths) = @_;
    #print STDERR "trimming things...\\r";
    for(my $i=0;$i<@#paths;$i++) {
        my ($xi1,$yi1,$th1,$th2) = vec2list($paths[$i]);
        for(my $j=$i+1;$j<@#paths;$j++) {
            my ($xi2,$yi2,$th2,$th1) = vec2list($paths[$j]);
            my $distsq = ($xi1-$xi2)**2 + ($yi1-$yi2)**2;
            my $radsq = ($ri1-$ri2)**2 + ($y1-$y2)**2;
            if($distsq < $radsq) {
                #print STDERR "\\ns1->j ";
                ($paths[$i],$paths[$j]) = intArcs($paths[$i],$paths[$j]);
            }
        }
    }
    #print STDERR "done\\n";
}

sub printXYR {
    my ($xxr,$yyr,$file) = @_;
    open(XXR, "> $file") or die;
    print XXR "(%t0.01\\n";
    foreach my $v (@$xxr) {
        my ($x,$y,$r) = vec2list($v);
        print XXR "$x $y $r = vec2list($v) ";
        print XXR "$x\\t$y\\t$r\\n";
    }
    close(XXR);
    return;
}

sub d2intod3 {
    my ($xx,$yy,$yyr,$yyv) = @_;
    my ($xp,$yp,$zp) = vec2list($pt);
    my ($xxr,$yyr,$yyv) = vec2list($xx);
    my ($xv,$yy,$yyv) = vec2list($yy);
    my ($xy,$yo) = vec2list($yy);
    my $x = fixnum35((($x-$xp)*$xxv + ($yy-$yyv)*$yyv + ($yy-$yyv)*$yyv));
    my $y = fixnum35((($y-$yp)*$yyv + ($yy-$yyv)*$yyv + ($yy-$yyv)*$yyv));
    my $z = $xxr*$xx+$yyr*$yy+$yyv;
    my $v = $yyr*$yy+$yyv*$yy+$yyv;
    my $s = $yyr*$yy+$yyv*$yy+$yyv;
    return list2vec($x,$y,$z);
}

sub calcSESLice {
    my ($pt,$vec,$isperp) = @_;
    #print STDERR "genPerVecs...";
    my ($xx,$yy) = genPerVecs($vec);
    my $xxr = genCircles($pt,$vec,$yy);
    #print STDERR "genCircles...";
    my $yyr = genCircles($pt,$vec,$xx,$yy); #xx, yy
    #print STDERR "printXYR...";
    if($printXYRFiles) { printXYR(\@yyr, "zigzag.dat"); }
    #only gets closest path
    #print STDERR "genPath...";
    my @paths = genPath(\@yyr,0);
    if($printPATHFiles) { printPath(\@paths, "zigzag.dat"); }
    #print STDERR "printPath...";
    my $surf = printPath(\@paths);
    if($printSurfFiles) { printSurf("done\\n"); }
}

#time to stop?
if($newcirc==$startCirc) { $loopcount = 2; }

$loopcount--;
} #end loop over path

```

```

#print STDERR "getArea...";
my $area = get2dArea(@surf);
print STDERR "AREA = ",fixnum2($area),"\n";
#print STDERR "getPerimeter...";
my $perimeter = get2dPerimeter(@surf);
print STDERR "PERM = ",fixnum2($perimeter),"\n";
#print STDERR "getCenter...";
my $avpt = getTrap2dCenter(@surf);
my $cpn, $cpv) = vec2list($avpt);
my $d3avg = d2intod3($avpt,$pt,$x2,$y2);
my $cp.shift = fixnum3($pt,(cpx*2+cpv*2));
#print STDERR "center pt: $avpt ---- ";
print STDERR "CP = $cp.shift ---- ";
#print STDERR "isInside...";
my $isInside = p1InsidePoly($o,o,@surf);
print STDERR "$INS = $isInside ---- ";
if($isInside && $perimeter < 800) {
    print STDERR color("green"),"good!\n",color("reset");
    print STDERR color("red"),"wrap!\n",color("reset");
    print STDERR color("red"),"island!\n",color("reset");
} else {
    print STDERR color("red"),"leak!\n",color("reset");
}
return -1;
#OLD STUFF OLD STUFF OLD STUFF
#die;
my ($mindia,$avgdia,$maxdia) = get2dMinMax(@surf);
my ($area,$perimeter,$min,$max); #stuff to return
#return ($area,$perimeter,$mindia,$avgdia,$avgpt);
#die;
my (@list,$polys);
push(@list,$list[0]); #close polygon
my ($x,$y) = vec2list($pt);
my $theta = 0;
for my $i(0..$#list;$i++) {
    my ($x2,$y2) = vec2list($list[$i]);
    my ($x1,$y1) = vec2list($list[$i+1]);
    $x1 -= $x; $x2 -= $x;
    $y1 -= $y; $y2 -= $y;
    #my $dp = $x1*$x2 + $y1*$y2;
    #my $cp = $x1*$y2 + $y1*$x2;
    #$theta += atan2($dp,$dd);
    $theta += angle2d($x1,$y1,$x2,$y2);
}
#print STDERR fixnum35($theta);
#If this sum is 2pi then the point is an interior point, if 0 then the
#point is an exterior point.
if(abs($theta) > PI) { return 1; }
return 0;
}

sub angle2d {
    my ($x1,$y1,$x2,$y2) = @_;
    my $theta1 = atan2($y1,$x1);
    my $theta2 = atan2($y2,$x2);
    my $dtheta = $theta2 - $theta1;
    while ($dtheta > PI) { $dtheta -= 2*PI; }
    return $dtheta;
};

sub findDist {
    my ($pt,$vec) = @_;
    my ($x0,$y0,$z0) = vec2list($pt);
    my ($x1,$y1,$z1) = vec2list($vec);
    my $good = 1; #stop factor
    my $ind = 0.0001; #last sucessful jump
    my $inrcr = $initCrawl; #beginning distance to travel
    my $guess = $guesses; #number of times to reduce incr
    for my $i($initCrawl,$good && $i < $endRad,$i+$inrcr) {
        my $x = $x0*$i1 + $x0;
        my $y = $y0*$i1 + $y0;
        my $z = $z0*$i1 + $z0;
        if(chkPoint(list2vec($x,$y,$z))) { #print STDERR "it is!\n"; }
        $ind=$i;
    }
    if($ind > $endRad - $initCrawl) {
        print STDERR color("red"),"\n";
        color("reset");
        #die;
        $ind = $endRad*$i;
        return $ind;
    }
    if($ind > $endRad - $initCrawl - 1) {
        $i-=$inrcr; $inrcr/=2; $guess--;
        #step back and reduce path
    } else {
        $good = 0; #stop the party
    }
}

sub newDirVect {
    my ($vec1,$vec2,$phi) = @_;
    my ($x1,$y1,$z1) = vec2list($vec1);
    my ($x2,$y2,$z2) = vec2list($vec2);
    my ($x3,$y3,$z3) = ($x1*$sin($phi) + $y1*$cos($phi),
                         $y1*$sin($phi) + $x1*$cos($phi),
                         $z1*$sin($phi) + $x2*$cos($phi));
    my $len = sqrt($x3**2 + $y3**2 + $z3**2);
    if($abs($len - 1) > 0.001) { print STDERR "search vector ($phi = ",fixnum23($len)," length ",fixnum23($len)," != 1)\n"; }
    return list2vec($x3,$y3,$z3);
};

sub getPerpVecs {
    my ($vec0) = @_;
    if($vec0) { $vec0 = $vector; }
    my ($xv,$yv,$zv) = vec2list($vec0);
    my ($x0,$y0,$z0) = my ($x1,$y1,$z1);
    if($xv != 0) {
        $x0 = -1 * $xv / $xv;
        $z0 = 1 / sqrt($xv**2 + 1); #unit vector
        $y0 = sqrt($xv**2 + 1);
    } else {
        if($die == 0, not functional yet); }
    my $len = sqrt($x0**2 + $y0**2 + $z0**2);
    #print STDERR "($x0,\$y0,\$z0) = ",printVec($x0,$y0,$z0),", -- $len\n";
}


```

```

$len = sqrt($xx**2 + $yy**2 + $zz**2);
$x1 = ($x0*$xx - $z0*$yy) / $len;
$y1 = ($xx*$z0 - $zz*$xx) / $len;
$z1 = ($yy*$z0 - $yy*$xx) / $len;
$len = sqrt($xx**2 + $yy**2 + $zz**2);
#print STDERR "<($x1,\$y1,\$z1) = \"printVec($x1,$y1,$z1),\" -- $len\n";
#print STDERR "<$1 . 'v1 . v2 = fixnum25($x0*$x1 + $y0*$y1 + $z0*$z1),\" --\n";
my $vec1 = list2vec($x0,$y0,$z0);
my $vec2 = list2vec($x1,$y1,$z1);
return ($vec1,$vec2);

sub getMinMax {
    my (@dists) = @_;
    my $min $avg $max = (1000,0,-1);
    my $r = (#dists+1)/2;
    my $count=0;
    my $d;
    if(int($r) != $r) { die "ERROR: MinMax, # of vecs not even \[$r]\n"; }
    for(my $i=0;$i<$r;$i++) {
        if(@dists[$i] < $endRad && $dists[$i+$r] < $endRad) {
            my $d = $dists[$i] + $dists[$i+$r];
            if($d > $max) { $max = $d; }
            if($d < $min) { $min = $d; }
            $count++;
            $avg += $d;
        }
    }
    if($count > 0) { $avg /= $count; }
    return ($min,$avg,$max);
}

sub getNextPoint {
    my ($step,$pt) = @_;
    my ($x0,$y0,$z0) = vec2list($pt); #last center
    my @t = vec2list($point);
    my @v = vec2list($vector);
    #print STDERR color("magenta"),"\n";
    printVec($point,"--> \"$pt\",";
    my $dists = ($x0-$t[0])*$v[0] + ($y0-$t[1])*$v[1] + ($z0-$t[2])*$v[2];
    my $add = $step - $dist;
    print STDERR color("blue"),"Dist travelled [dist] ";
    print STDERR color("blue"),"Increase to [step] ",fixnum23($step),
    color("reset");
    if($add < 0) {
        print STDERR color("red"),"NEGATIVE travel distance ***\n",
        color("reset");
        $add=0;
    }
    if(abs($add) > $planeDens*3) {
        print STDERR color("red"),"*** LARGE travel distance ***\n",
        color("reset");
    }
}

my $x = $x0 + $add*$v[0];
my $y = $y0 + $add*$v[1];
my $z = $z0 + $add*$v[2];
my $pt2 = list2vec($x,$y,$z);
#print STDERR color("blue"),"old--> new ",printVec($pt)," --> ",
# printVec($pt2)," --> new ";
return list2vec($x,$y,$z);

sub wobble {
    #$wobbleTilt = max wobble tilt in degrees
    #$wobNumTilts = must be odd to get zero angle
    #$wobNumRot = spins about each title
    my ($pt,$step) = @_;
}

```

```

$p2 = averVec(@p);
print STDERR " cen pt shift " fixnum23(distVec($pt,$pt2)), "\t";
print STDERR "average area " fixnum32(averHash(%$ar,@p)), "\n";
$avper = averList(@perimeter);
#PRINT STDERR "average perimeter " fixnum32($avper), "\n", color("reset");
@p = sort { $ar{$a} <=> $ar{$b} } @p;
my $avarea = averHash(%$ar,@p);
if($avarea > 600) { die; }

my $maxarea = $ar{$p[0]}; 
my $minarea = $ar{$p[1]};
my $avdia = averList(@dia);
print WOB vec2pob($p[2],$step*100,$avarea,$step*100,"CC");
close(WOB);

return ($mindia,$avdia,$minarea,$avarea,$maxarea,$pt2,$avper);
};

sub remLargeAreas {
    my ($cut,$ar,@p) = @_;
    my @p2;
    foreach my $pt (@p) {
        if($$ar{$pt} < $cut) {
            push(@p2,$pt);
        }
    }
    #may need check for removing too much
    while(( $$p2[-1] < 0.2*( $$p[-1] ) ) {
        $cut*=1.2;
        @p2 = remLargeAreas($cut,$ar,@p);
        print STDERR "^.";
        print STDERR color("blue"), " ", ("$p-$#p2"), " point(s) removed\t",color("reset"));
        return @p2;
    };
}

sub remAreaOutlier {
    my ($ar,@p) = @_;
    my $cutoff = 0.6;
    @p = sort { $ar{$a} <=> $ar{$b} } @p;
    my $area = averHash($ar,@p);
    #PRINT STDERR "average area before remove: " ,fixnum33($area),"\n";
    my $av;
    #foreach my $pt3 (@p) { $av += $$ar{$pt3}; }
    #PRINT STDERR "average area after remove: " ,fixnum33($area),"\n";
    my $max = int($area*(1+$cutoff)*5);
    my $min = int($area*(1-$cutoff)*1);
    foreach my $pt (@p) {
        if($$ar{$pt} < $max && $$ar{$pt} > $min) {
            push(@p2,$pt);
        }
    }
    $area = averHash($ar,@p);
    #PRINT STDERR "average area after remove: " ,fixnum33($area),"\n";
    #$av = "";
    #foreach my $pt3 (@p) { $av .= $$ar{$pt3}; }
    #PRINT STDERR printVec($av),"\n";
    print STDERR color("blue"), " ", ("$p-$#p2+1"), " point(s) removed ($min/$max)\n",color("reset");
    return @p2;
};

sub remCentPOutlier {
    my ($pt,@p) = @_;
    my $cutoff = 2.0;
    my $av;
    foreach my $pt3 (@p) {
        $av .= $$ar{$pt3};
    }
    print STDERR color("blue"), " ", ("$p-$#p2+1"), " point(s) removed ($min/$max)\n",color("reset");
    return @p2;
};

sub remCentPOutlier {
    my ($pt,@p) = @_;
    my $cutoff = 2.0;
    my $av;
    foreach my $pt3 (@p) {
        $av .= $$ar{$pt3};
    }
    print STDERR color("blue"), " ", ("$p-$#p2+1"), " point(s) removed ($min/$max)\n",color("reset");
    return @p2;
};

sub remCentPOutlier {
    my ($pt,@p) = @_;
    my $cutoff = 2.0;
    my $av;
    foreach my $pt3 (@p) {
        $av .= $$ar{$pt3};
    }
    print STDERR color("blue"), " ", ("$p-$#p2+1"), " point(s) removed ($min/$max)\n",color("reset");
    return @p2;
};

sub distVec {
    my ($x1,$y1,$z1) = @_;
    my ($x2,$y2,$z2) = vec2list($v1);
    my ($x3,$y3,$z3) = vec2list($v2);
    $x1 -= $x2;
    $y1 -= $y2;
    $z1 -= $z2;
    $v1 = list2vec($x1,$y1,$z1);
    return $v1;
}

```

```

    };

    sub averVec {
        my (@v) = @_;
        if ($#v > 0) {
            my ($x,$y,$z);
            foreach my $vec (@v) {
                my ($x1,$y1,$z1) = vec2List($vec);
                $x+= $x1;
                $y+= $y1;
                $z+= $z1;
            }
            $x=($#/v+1);
            $y=($#/v+1);
            $z=($#/v+1);
            return list2vec($x,$y,$z);
        } elsif($#v == 0) {
            return $v[0];
        }
        return 0;
    };

    sub rotVecAboutVec {
        my ($vec,$rotvec,$angle) = @_;
        $angle = $angle*PI/180;
        my ($rx,$ry,$rz) = vec2List($rotvec);
        my ($x0,$y0,$z0) = vec2List($vec);
        my ($a,$b,$c);

        #error checking
        my $len = $x0**2 + $y0**2 + $z0**2;
        if($abs($len-1) > 0.0001) { print STDERR "rotVec: vec not a unit vector\n"; }

        my $len = $rx**2 + $ry**2 + $rz**2;
        if($abs($len-1) > 0.0001) { print STDERR "rotVec: rotvec not a unit vector\n"; }

        #first direction
        $a = 1 + (1-$cos($angle))*( $rx*$rx-1 );
        $b = -$rx*$sin($angle)+(1-$cos($angle))*$rx*$ry;
        $c = $ry*$sin($angle)+(1-$cos($angle))*$rx*$rz;
        my $x = $a*$x0+$b*$y0+$c*$z0;

        $a = $rz*$sin($angle)+(1-$cos($angle))*$rx*$rz;
        $b = 1 + (1-$cos($angle))*( $ry*$ry-1 );
        $c = -$rx*$sin($angle)+(1-$cos($angle))*$ry*$rz;
        my $y = $a*$x0+$b*$y0+$c*$z0;

        #third direction
        $a = -$ry*$sin($angle)+(1-$cos($angle))*$rx*$rz;
        $b = $rx*$sin($angle)+(1-$cos($angle))*$ry*$rz;
        $c = 1 + (1-$cos($angle))*( $rz*$rz-1 );
        my $z = $a*$x0+$b*$y0+$c*$z0;

        #error checking -- only true if vec perp rotvec
        #my $dot = ($x*$x0+$y*$y0+$z*$z0);
        #if($abs($dot-cos($angle)) > 0.0001) {
        #    print STDERR "Rotvec: ",fixnum13(cos($angle)), " != ",$dot,"\\n";
        #    print STDERR "Rotvec: ",fixnum13($angle), " != ",fixnum13acos($dot), "\\n";
        #}
        return list2vec($x,$y,$z);
    };

    sub acos {
        my ($rad) = @_;
        if($#rad >= 1) {
            print STDERR color("red"), "cannot take acos > 1\\n", color("reset");
            return 0;
        }
        my $ret = atan2(sqrt(1 - $rad**2), $rad);
        return $ret;
    };

    sub removeFiles {

```

```

#!/usr/bin/perl -w
use strict;
use Getopt::Std;
use constant PI => 3.14159265359;

my %args;
getopt("r:", \%args);
my $numps = 36;

my $in = $args{i} or die;
my $out = "$in-meth2.prs";
$out =~ s/\./\.\#/g;

my @angs = getAngles($numps);
my $atoms = getNumLines($in);
my $expect_count = ($#angs+2)*$expect_atoms;

print STDERR "\nwriting to $out...\n";
for(my $i=1; $i<=$expect_atoms; $i++) {
    if($i%300 == 0) { print STDERR "\#\n"; }
    print STDERR "\n";
}

open(IN, "< $in") or die;
open(OUT, "> $out") or die;
print OUT "/home/vosman/reduce/$out $expect_count D3\n";
my $count=0;
my $atoms=0;
while((my $line = <IN>) < 0) {
    $atoms++;
    my ($x,$y,$z,$r) = split(" ",$line);
    foreach my $p ($x,$y,$z,$r) {
        $p -= $atoms/4;
    }
    if(!$p) { die; }
}
$count++;
print OUT "$x\n$y\n$z\n";
foreach my $ang (@angs) {
    my ($phi,$theta) = split(" ",$ang);
    #for(my $i=1; $i<=$numps; $i++) {
    my $theta = rand(2*PI);
    my $phi = $x + $r*cos($theta)*sin($phi);
    my $yd = $y + $r*sin($theta)*sin($phi);
    my $zd = $z + $r*cos($phi);
    $count++;
    print OUT "$xd\n$yd\n$zd\n";
}
if($atoms%3000 == 0) { print STDERR "\#\n"; }

print STDERR "\n";
close(OUT);
close(IN);
print STDERR "writing header\n";
print STDERR "$count of $expect_count entries for",
" $atoms of $expect_atoms atoms\n";
print STDERR "now type cat $out | nice -e19 qhull-2003.1/src/qconvex",
" > results.txt\n";
}

sub clean {
    my ($str) = @_;
    $str =~ s/^\$/s//; $str =~ s/^\$s//;
    return $str;
}

```

Appendix F

The Voss Volume Voxelator Program

F.1 Library

The main library which is called by all the programs in the 3v package is broken into four files:

- Makefile: commands to compile all the programs. Page 290
- utils.h: the header file, a list of all the functions. Page 291
- utils-main.cpp: the main functions for rolling a probe, importing a file, *etc.* Page 292
- utils-output.cpp: this file contains functions dealing with outputting the grid, either to PDB (x, y, z coordinates) or EZD (electron density grid) file formats. EZD can then be converted into the universal CCP4 file format using MAPMAN. Page 303
- pdb_to_xyzr.sh: this file is the script that converts a PDB file into an XYZR file using “atmtypenumbers.dat” (below). Page 309
- atmtypenumbers.dat: the files contains the VDW radius information for converting a PDB file into an XYZR file. Page 310

```
CC = g++
#EXT= `perl ccpuflags.pl`
#FLAGS = -O3 -Wall $(EXT)
FLAGS = -O7 -Wall -mfpmath=sse -march=pentium4 -msse -ffast-math -mmmx -m3dnow -msse2
#FLAGS = -O7 -Wall -mfpmath=sse,387 -march=k8 -msse -mmmx -m3dnow -ffast-math -m64 -msse2 -mcpu=k8
#FLAGS = -mcpu=pentium3 -O7 -Wall -march=pentium3 -mfpmath=sse -msse -mmmx -mtune=pentium3

all: cav chan fsv sol tun vdw vol

cav: utils-main.o utils-output.o cavities.cpp
    $(CC) $(FLAGS) -o Cav.exe utils-main.o utils-output.o cavities.cpp
    chmod 777 Cav.exe
    mv Cav.exe ..../bin

chan: utils-main.o utils-output.o channel.cpp
    $(CC) $(FLAGS) -o Channel.exe utils-main.o utils-output.o channel.cpp
    chmod 777 Channel.exe
    mv Channel.exe ..../bin

fsv: utils-main.o utils-output.o calc_fsv2.cpp
    $(CC) $(FLAGS) -o FsvCalc.exe utils-main.o utils-output.o calc_fsv2.cpp
    chmod 777 FsvCalc.exe
    mv FsvCalc.exe ..../bin

tun: utils-main.o utils-output.o tunnel.cpp
    $(CC) $(FLAGS) -o Tunnel.exe utils-main.o utils-output.o tunnel.cpp
    chmod 777 Tunnel.exe
    mv Tunnel.exe ..../bin

vdw: utils-main.o utils-output.o vdw.cpp
    $(CC) $(FLAGS) -o VDW.exe utils-main.o utils-output.o vdw.cpp
    chmod 777 VDW.exe
    mv VDW.exe ..../bin

vol: utils-main.o utils-output.o volume.cpp
    $(CC) $(FLAGS) -o Volume.exe utils-main.o utils-output.o volume.cpp
    chmod 777 Volume.exe
    mv Volume.exe ..../bin

utils-main.o: utils-main.cpp
    $(CC) $(FLAGS) -c -o utils-main.o utils-main.cpp

utils-output.o: utils-main.o utils-output.cpp
    $(CC) $(FLAGS) -c -o utils-output.o utils-main.o utils-output.cpp

none:
    echo "Please type make xxx, where xxx = cav, chan, fsv, sol, tun, vdw, vol"

clean:
    rm -f *.o
    rm -f *.*~
    rm -f ..../binbin/*.exe
```

```

#ifndef CITATION
#define CITATION \
    fprintf(stderr, \
    "Citation: %s\nDOI: %s<%s> or MR Voss <%s>. \n\n" \
    "'Neil R Voss, et al. J Mol Biol. v360 (4): 2006, pp. 893-906', \n \
    "http://dx.doi.org/10.1016/j.jmb.2006.05.023", "mark.gerstein@yale.edu", "vossman77@yahoo.com") \
#endif

#define MAXWDW 2.0
#define MAXBINS 1048576 //2^31
#define MAXLIST 1048576 //2^20

#define COMPILE_INFO sprintf(stderr, \
"Program %s at line %d in source %s, which was compiled on %s.\n\n", \
argv[0], __LINE__, __FILE__, __DATE__, __TIME__)

//MAXWDW DEPENDS ON AMOUNT OF RAM ON COMPUTER
#define MAXXIN 2147483647 //2^31
#define MAXBINS 1048576 //2^20

extern float XMIN, YMIN, ZMIN;
extern float XMAX, YMAX, ZMAX;
extern int DX, DY, DZ;
extern int DXY, DXZY;
extern unsigned int NUMBINS;
extern float MAXPROBE;
extern float GRIDVOL;
extern float WATERRES;
extern float CUTOFF;
extern char XYZFILE[256];

//using namespace std;
using std::cerr;
using std::endl;
using std::flush;
using std::ifstream;
using std::ofstream;
using std::cout;
using std::cout;

//typedef unsigned short int gridpt;
#ifndef _GRID_PT
typedef bool gridpt;
#define _GRID_PT
#endif

#ifndef _GRID_STRUCT
# define _GRID_STRUCT
struct ind { int i; int j; int k; float b; };
struct real { float x; float y; float z; };
struct vector { float x; float y; float z; };
#endif

//init functions
void finalGridBins (float maxprobe);
float getIdealGrid (O);
void assignLimits (O);
void restLimits (gridpt grid[]);

//grid util functions
int countGrid (gridpt grid[]);
int zeroGrid (gridpt grid[]);
int copyGridFromTo (gridpt oldgrid[], gridpt newgrid[]);

int copyGrid (gridpt oldgrid[], gridpt newgrid[]);
void inverseGrid (gridpt grid[]);

//file based functions
int readNumAtoms (char file[]);
int fillAccessGrid_fromFile (int numatoms, const float probe, char file[], gridpt grid[]);
int get_ExcludeGrid_fromFile (int numatoms, const float probe, char file[], gridpt EXGrid[]);

//generate grids / grid changers;
void trun_ExcludeGrid (const float probe, gridpt ACCgrid[], gridpt EXgrid[]); //contract
void grow_ExcludeGrid (const float probe, gridpt ACCgrid[], gridpt EXgrid[]); //expands
int get_Connected (gridt grid[], gridt connect[], const float x, const float y, const
float z);
int get_ConnectedRange (gridt grid[], gridt connect[], const float x, const float y, const
float z);
int get_ConnectedPoint (gridt grid[], gridt connect[], const int gp);
int sub_Grids (gridt biggrid[], gridt smgrid[]); //Modifies biggrid; returns difference
int intersect_Grids (gridt grid[], gridt grid2[]); //Modifies grid2[]; returns final vox num

//point based function
int fillAccessGrid (const float x, const float z, const float r, gridpt grid
[]); //Modifies grid[]; returns difference
void empty_ExcludeGrid (const int i, const int k, const float probe, gridpt grid
[]); //Modifies grid[]; returns difference
void fill_ExcludeGrid (const int i, const int j, const int k, gridpt grid[]); //Modifies grid[]; returns difference
bool isEdgePoint (const int i, const int j, const int k, gridpt grid[]); //Modifies grid[]; returns difference
int ij2pc (int i, int j, int k);
bool isEdgePoint_Fill (const int pt, gridpt grid[]); //Modifies grid[]; returns difference
bool isEdgePoint_Star (const int pt, gridpt grid[]); //Modifies grid[]; returns difference
void ij2pcdb (char line[], int i, int j, int k, int n);

//special
bool isCloseToVector (const float radius, const int pt);
void limitTunnelArea (const float radius, gridpt grid[]);
float distFront (const float x, const float y, const float z);
float crossSection (const real p, const vector v, const gridpt grid[]); //Modifies grid[]; returns difference
void real2index (const real p, int i);

//string functions
void padLeft(char a[], int n);
void padRight(char a[], int n);
float crossSection (const gridpt grid[]); //Modifies grid[]; returns difference
void printBar ();
void printVol (int vox);
void printVolCut (int vox);
void basename(char str[], char base[]);

//surface area
float surfaceArea (gridpt grid[]);
int classifyEdgePoint (const int pt, gridpt grid[]);

//other ideas
int bounding_box (gridpt grid[], gridpt bbox[]);
int fill_cavities (gridpt grid[]);

//output functions (in utils-output.cpp)
void write_PDB (gridpt grid[], char outfile[]);
//int convex_hull(gridpt grid[], gridpt hull[]);
int write_SurfPDB (gridpt grid[], char outfile[]);
void write_EZD (gridpt grid[], char outfile[]);
void write_BluEZD (gridpt grid[], char outfile[]);
void write_HalfEZD (gridpt grid[], char outfile[]);
void write_ThirdEZD (gridpt grid[], char outfile[]);
void write_FifthEZD (gridpt grid[], char outfile[]);

```

```

/*
 ** utils.cpp
 * /
 #include "utils.h"

float XMIN, YMIN, ZMIN;
float XMAX, YMAX, ZMAX;
int DX, DY, DZ;
int DXY, DXYZ;
unsigned int NUMBINS;
float MAXPROBE=15;
float GRID=0.5;
float GRIDVOL=GRID*GRID*GRID;
float WATER_RES=14.1372*GRIDVOL;
float CUTOFF=10000;
char XYZFILE[256]; //XYZRFILE[0]='\0';

//INITIALIZE FUNCTIONS
******/
```

```

void finalGridBins (float maxprobe) {
    GRIDVOL=GRID*GRID*GRID;
    WATER_RES=14137.2*GRIDVOL;
    MAXPROBE = maxprobe;
    XYZFILE[0]= '\0';
    XMIN=1000;
    YMIN=1000;
    ZMIN=1000;
    XMAX=-1000;
    YMAX=-1000;
    ZMAX=-1000;
}

//init functions
void finalGridBins (float maxprobe) {
    GRIDVOL=GRID*GRID*GRID;
    WATER_RES=14137.2*GRIDVOL;
    MAXPROBE = maxprobe;
    XYZFILE[0]= '\0';
    XMIN=1000;
    YMIN=1000;
    ZMIN=1000;
    XMAX=-1000;
    YMAX=-1000;
    ZMAX=-1000;
}

float getIdealGrid () {
    bool cont = 1;
    unsigned int numbins;
    int diff;
    unsigned int dx,dy,dz,dxy,dxyz;
    float bng=-1.0, bng=-1.0;
    int bnd=1, bnd=1;
    int counts=0;
    int (cont) {
        counts++;
        if(grid < 0.0001) { grid += 0.01; }
        dx=int((XMAX-XMIN)/grid+1);
        dy=int((YMAX-YMIN)/grid+1);
        dz=int((ZMAX-ZMIN)/grid+1);
        dxy=(dx*dy);
        dxyz=(dz*dxy);
        numbins = dxyz + dxy + dx + 1;
        diff = MAXBINS - numbins;
        //cerr << "grid = " << grid << "\tdiff = " << diff <<
        //"\tnumbins = " << numbins << endl;
        if(diff < 0) { diff > bnd; }
        bng = grid;
        bnd = diff;
        grid += 0.0001;
    }
}
```

```

if(diff > 0) {
    if(bpg < 0 || diff < bpd) {
        bpg = grid;
        bpd = diff;
    }
    grid -= 0.0001;
}
// cerr << "\tbpg = " << bpg << "\tbng = " << bng << endl;
// cerr << "\tbndiff = " << bpd << "\tbndiff = " << bnd << endl;
if(fabs(bpg - bng) < 0.0002 || counts > 10000) {
    cont = 0;
}
return bpg;
};

void assignLimits () {
    DX=int((XMAX-XMIN)/(GRID+1));
    DY=int((YMAX-YMIN)/(GRID+1));
    DZ=int((ZMAX-ZMIN)/(GRID+1));
    DXY=(DX*DY);
    DXYZ=(DZ*DXY);
    //NMBINS = 3*DXYZ;
    //NMBINS = DXYZ + DXY + DYZ + DX + DY + DZ + 1;
    NMBINS = DXYZ + DXY + DZ + 1;
}

void precent_filled_NUMBINS() {
    int NUMBINS=1000.0/MAXBINS/10.0 << "%";
    float idealGrid = getIdealGrid();
    cerr << "ideal Grid: " << idealGrid << endl;
    if(NUMBINS > MAXBINS) {
        cout << MAXPROBE << " grid " << GRID << " is too large; use " <<
        idealGrid << " for " << XYZFILE << endl;
        cerr << "#### grid is too large #####" << endl << endl;
        exit (1);
    }
}

void testLimits (gridpt grid[])
{
    cerr << "int(1.2) is " << int(1.2) << endl;
    cerr << "int(-1.2) is " << int(-1.2) << endl;
    cerr << "XMIN: " << XMIN << endl;
    cerr << "YMIN: " << YMIN << endl;
    cerr << "ZMIN: " << ZMIN << endl;
    cerr << "DX: " << DX << endl;
    cerr << "DY: " << DY << endl;
    cerr << "DZ: " << DZ << endl;
    cerr << "DXY: " << DXY << endl;
    cerr << "DXYZ: " << DXYZ << endl;
    cerr << "NUMBINS: " << NUMBINS << endl;
}

unsigned int i;
//first filled spot;
for(i=0; i<NUMBINS && !grid[i]; i++) { }
//i << endl << "Last filled spot: ";
for(i=NUMBINS-1; i>0 && !grid[i]; i--) { }
//i << endl;

//*****
//GRID UTILITY FUNCTIONS
*****
```

```

*****  

***** Inverting All Voxels in the Grid... *****  

*****  

int countGrid (gridpt grid[]) {
    int voxels=0;
    for(unsigned int pt=0; pt<NUMBINS; pt++) {
        if(grid[pt]) {
            grid[pt] = 0;
        } else {
            grid[pt] = 1;
        }
        cerr << "done " << endl << endl;
        return;
    }
}

cerr << "Counting up Voxels in Grid for Volume..." << flush;

for(unsigned int pt=0; pt<NUMBINS; pt++) {
    if(grid[pt]) {
        voxels++;
    }
}
cerr << "done [ " << voxels << " voxels ] " << endl << endl;
return voxels;
};

void zeroGrid (gridpt grid[]) {
    if (grid==NULL) {
        cerr << "Allocating Grid..." << endl;
        grid = (gridptr*) malloc (NUMBINS);
        if (grid==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }
    }
    cerr << "Zero-ing All Voxels in the Grid..." << flush;
    for(unsigned int pt=0; pt<NUMBINS; pt++) {
        grid[pt] = 0;
    }
    cerr << "done " << endl << endl;
    return;
};

int copyGridFromTo (gridpt oldgrid[], gridpt newgrid[]) {
    return copyGrid(oldgrid,newgrid);
}

int copyGrid (gridpt oldgrid[], gridpt newgrid[]) {
    //Zero Grid Not Required
    int voxels=0;
    if (newgrid==NULL) {
        cerr << "Allocating Grid..." << endl;
        newgrid = (gridptr*) malloc (NUMBINS);
        if (newgrid==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }
    }
    cerr << "Duplicating Grid and Counting up Voxels..." << flush;
    for(unsigned int pt=0; pt<NUMBINS; pt++) {
        if(oldgrid[pt]) {
            voxels++;
            newgrid[pt] = 1;
        } else {
            newgrid[pt] = 0;
        }
        cerr << "done " << endl << endl;
        return voxels;
    }
}

void inverseGrid (gridpt grid[]) {
    if (grid==NULL) {
        cerr << "Allocating Grid..." << endl;
        grid = (gridptr*) malloc (NUMBINS);
        if (grid==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }
    }
}

*****  

***** FILE BASED FUNCTIONS *****  

*****  

int readNumAtoms (char file[]) {
    ifstream infile;
    int count = 0;
    char line[256];
    float minmax[6];
    minmax[0] = 100; minmax[1] = 100; minmax[2] = 100;
    minmax[3] = -100; minmax[4] = -100; minmax[5] = -100;
    strcpy(XYZRFILE,file);
    cerr << "Reading file for Min/Max: " << file << endl;
    infile.open(file);
    while(infile.getline(line,255)) {
        float x,y,z,r;
        count++;
        sscanf(line, "%f %f %f %f",&x,&y,&z,&r);
        if(count % 3000 == 0) { cerr << " atoms " << endl; }
        if(x < minmax[0]) { minmax[0] = x; }
        if(x > minmax[3]) { minmax[3] = x; }
        if(y < minmax[1]) { minmax[1] = y; }
        if(y > minmax[4]) { minmax[4] = y; }
        if(z < minmax[2]) { minmax[2] = z; }
        if(z > minmax[5]) { minmax[5] = z; }
    }
    infile.close();
    cerr << endl << "[ read " << count << " atoms ] " << endl << endl;
}

//INCREASE GRID SIZE TO ACCOMODATE SPHERES
//ALSO ROUND GRID SIZE SO THE XMIN = INTEGER * GRID (FOR BETTER OUTPUT)
float FACT = MAXDIV + MAXPROBE + 2*GRID;
for(int i=0;i<2;i++) {
    minmax[i] -= FACT;
    minmax[i] = int(minmax[i]/(4*GRID)-1)*4*GRID;
}
for(int i=3;i<5;i++) {
    minmax[i] += FACT;
    minmax[i] = int(minmax[i]/(4*GRID)+1)*4*GRID;
}
if(minmax[0] < XMIN) { XMIN = minmax[0]; }
if(minmax[1] < YMIN) { YMIN = minmax[1]; }
if(minmax[2] < ZMIN) { ZMIN = minmax[2]; }
if(minmax[3] > XMAX) { XMAX = minmax[3]; }
if(minmax[4] > YMAX) { YMAX = minmax[4]; }
if(minmax[5] > ZMAX) { ZMAX = minmax[5]; }

cerr << "Now Run AssignLimits() to Get NUMBINS Variable" << endl << endl;
return count;
};

```

```

int fillAccessGridFromFile (int numatoms, const float probe, char file[],  

    gridpt grid[]) {  

    ifstream infile;  

    char line[256];  

    if( (grid==NULL) {  

        cerr << "Allocating Grid..." << endl;  

        grid = (gridpt*) malloc (NOMBINS);  

        if (grid==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }  

    }  

    zeroGrid(grid);  

    infile.open(file);  

    if(strcpy(XYZRFILE10) { strcpy(XYZRFILE,file); }  

    float count = 0;  

    const float cat = float(numatoms)/60.0;  

    float cut = cat;  

    printBar();  

    infile.open(file);  

    int filled=0;  

    while(infile.getline(line,255)) {  

        float x,y,z,r;  

        count++;  

        if(count > cut) {  

            cerr << "Filling Atoms into Grid (probe " << probe << ")"...<< endl;  

            printBar();  

            cut += cat;  

        }  

        sscanf(line,"%f %f %f %f",&x,&y,&z,&r);  

        filled+= fill_AccessGrid(x,y,z,r,probe,grid);  

    }  

    infile.close();  

    cerr << endl << "[ read " << count << " atoms ]" << endl;  

    printVol(filled);  

    cerr << endl;  

    return filled;
}

//OUTPUT INFO
cerr << endl << "Access volume for probe " << probe << flush;
cerr << " voxels " << voxels << flush;
cerr << " x gridvol " << GRIDVOL << endl;
cerr << " y gridvol " << GRIDVOL << endl;
cerr << " z gridvol " << GRIDVOL << endl;
cerr << endl << "ACCESS VOL: " ;
printVol(filled);
cerr << endl;

int get_ExcludeGrid_fromFile (int numatoms, const float probe,
    char file[], gridpt EXCgrid[]) {
    //READ FILE INTO ACCGRID
    gridpt *ACCgrid;
    cerr << "Allocating Grid..." << endl;
    ACCgrid = (gridpt*) malloc (NOMBINS);
    if (ACCgrid==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }
    fillAccessGridFromFile(numatoms,probe,file,ACCgrid);
}

//TRUNCATE GRID
trun.ExcludeGrid(probe,ACCgrid,EXCgrid);

//RELEASE ACCGRID
free (ACCgrid);

//OUTPUT INFO
int voxels = countGrid(EXCgrid);
cerr << endl << "voxels " << voxels << endl;
cerr << "Excluded Volume for Probe " << probe << flush;
cerr << endl << "Excluded Volume for Probe " << probe << flush;

```

```

    cerr << " voxels " << voxels << flush;
    cerr << " x gridvol " << GRIDVOL << endl;
    cerr << " y gridvol " << GRIDVOL << endl;
    cerr << " z gridvol " << GRIDVOL << endl;
    cerr << endl << "*****" << endl;

    ifstream infile;
    char line[256];
    if(XYZRFILE10) { strcpy(XYZRFILE,file); }

    float count = 0;
    const float cat = float(numatoms)/60.0;
    float cut = cat;

    const float jmin = 0;
    const float kmin = 0;
    const float lmin = 0;
    const float jmax = DX;
    const float kmax = DY;
    const float lmax = DZ;

    float count = 0;
    float cut = cat;

    if(EXCgrid==NULL) {
        cerr << "Allocating Grid..." << endl;
        EXCgrid = (gridpt*) malloc (NOMBINS);
        if (EXCgrid==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }
    }

    copyGridFromTo(ACCgrid,EXCgrid);

    cerr << "Truncating Excluded Grid from Accessible"
    << "Grid by Probe " << probe << endl;
    printBar();

    for(int k=kmin; k<kmax; k+=DZ) {
        count++;
        if(count > cut) {
            cerr << "Grid by Probe " << probe << endl;
            cut += cat;
        }
        for(int j=jmin; j<jmax; j+=DX) {
            for(int i=imin; i<imax; i++) {
                if( (ACCgrid[i][j][k]) & (EXCgrid[i][j][k]) ) {
                    const int k2 = k/DZ;
                    const int j2 = j/DX;
                    if(isEdgePoint(i,j,k,ACCgrid)) {
                        empty_ExcludeGrid(i,j2,k2,probe,EXCgrid);
                    }
                }
            }
        }
    }
    cerr << endl << "done" << endl << endl;
    return;
}

void grow_ExcludeGrid (const float probe, gridpt ACCgrid[],
    gridpt EXCgrid[]) {
    //expands
    //limit grid search
}

```

```

LIST[1] = 0;
while(last != 0) {
    //cerr << "n" << flush;
    int newlast = 0;
    int NEWLIST[MAXLIST];
    for(int n=0; n<last; n++) {
        steps++;
        int p = LIST[n];
        for(int i=1; i<1; i++) {
            for(int j=i-DX; j<=DX; j+=DX) {
                for(int k=j-DY; k<=DY; k+=DY) {
                    int pt = p + i + j + k;
                    if(gridpt[i] && !connect[pt]) { //isCloseZTunnel???
                        connect[pt] = 1;
                        connected++;
                        if(newlast < MAXLIST-10) {
                            NEWLIST[newLast] = pt;
                            newlast++;
                        }
                    }
                }
            }
        }
    }
    for(int n=0; n<newlast; n++) {
        LIST[n] = NEWLIST[n];
    }
    last=newlast;
    LIST[last]=0;
}
//cerr << endl;
if(steps > 1) {
    cerr << " performed " << steps << endl;
} else {
    cerr << " done" << endl;
}
else if(gp > 0 && gp < max) {
    cerr << "GetConnected: Point OUT OF RANGE" << endl;
} else {
    cerr << "GetConnected: Point is NOT FILLED" << endl;
}
return connected;
}

int get_ConnectedRange (gridpt grid[], gridpt connect[], const float x, const float y, const float z) {
//Get selected point in Grid
const int ip = int((x-XMIN)/GRID+0.5);
const int jp = int((y-YMIN)/GRID+0.5);
const int kp = int((z-ZMIN)/GRID+0.5);
if (connect==NULL) {
    cerr << "Allocating Grid..." << endl;
    connect = (gridpt*) malloc (NUMBINS);
    if (connect==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }
    zeroGrid(connect);
}
if (connect==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }
}

//Oops selected point isn't open! Better get new one
if(grid[gp]) {
    const int delta = int(1.50/GRID);
    const int ip = int((x-XMIN)/GRID+0.5);
    const int jp = int((y-YMIN)/GRID+0.5);
    const int kp = int((z-ZMIN)/GRID+0.5);
    if (connect==NULL) {
        cerr << "Allocating Grid..." << endl;
        connect = (gridpt*) malloc (NUMBINS);
        if (connect==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }
    }
}

int get_Connected (gridpt grid[], gridpt connect[], const float x, const float y, const float z) {
const int ip = int((x-XMIN)/GRID+0.5);
const int jp = int((y-YMIN)/GRID+0.5);
const int kp = int((z-ZMIN)/GRID+0.5);
if (connect==NULL) {
    cerr << "Allocating Grid..." << endl;
    connect = (gridpt*) malloc (NUMBINS);
    if (connect==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }
    zeroGrid(connect);
}
const int max = NUMBINS;
int steps=0;
int connected=0;
if(gp>=0 && gp <= max && grid[gp]) {
    connect[gp] = 1;
    cerr << "GetConnected..." << flush;
}
int LIST[MAXLIST];
int last = 1;
LIST[0] = gp;
stop=1;
bool stop=0;
int gd=gp;
for(int jd=delta; !stop && id<delta; id++) {
    for(int jd=delta; !stop && jd<delta; jd++) {
        for(int kd=delta; !stop && kd<delta; kd++) {
            gd = ijk2pt(ip+id,jp+jd,kp+kd);
            if (grid[gd]) {
                stop=1;
            }
        }
    }
}
}

```

```

    gp=gd;
}
}

const int max = NUMBINS;
int steps=0;
int connected=0;
if(gp > 0 && gp <= max && grid[gp]) {
    connect[gp] = 1;
    cerr << "GetConnected..." << flush;
}

int LIST[MAXLIST];
int last = -1;
LIST[0] = gp;
LIST[1] = 0;

while(last != 0) {
    while(last != 0) {
        if(gp > 0 && gp <= max && grid[gp]) {
            connect[gp] = 1;
            cerr << "GetConnected..." << flush;
        }

        int LIST[MAXLIST];
        int last = -1;
        LIST[0] = gp;
        LIST[1] = 0;

        while(last != 0) {
//cerr << " " << flush;
        int newlast = 0;
        NEWLIST[MAXLIST];
        for(int n=0, n<last; n++) {
            steps++;
            int p = LIST[n];
            for(int i=-1; i<=1; i++) {
                for(int j=-DX; j<=DX; j+=DX) {
                    for(int k=-DX; k<=DX; k+=DX) {
                        int pt = p + i + j + k;
                        if(grid[pt] && !connect[pt]) {
                            connect[pt] = 1;
                            connected++;
                            if(newlast < MAXLIST-10) {
                                NEWLIST[newlast] = pt;
                                newlast++;
                            }
                        }
                    }
                }
            }
            if(newlast < MAXLIST-10) {
                NEWLIST[newlast] = p;
                newlast++;
            }
        }
        if(newlast < MAXLIST-10) {
            NEWLIST[newlast] = p;
            newlast++;
        }
    }
}

for(int n=0, n<newlast; n++) {
    LIST[n] = NEWLIST[n];
}
last=newlast;
LIST[last]=0;

//cerr << endl;
if(steps > 1) {
    cerr << " performed " << steps << " steps" << endl;
} else {
    cerr << " done" << endl;
}

return connected;
}

int get_Connected_Point (gridpt grid[], gridpt connect[], const int gp) {
    if(connect==NULL) {
        cerr << "Allocating Grid..." << endl;
        connect = (gridpt*) malloc (NUMBINS);
        if (connect==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }
        zeroGrid(connect);
    }
    const int max = NUMBINS;
    int steps=0;
}

int get_Connected_Point (gridpt grid[], gridpt connect[], const int gp) {
    if(connect==NULL) {
        cerr << "Allocating Grid..." << endl;
        connect = (gridpt*) malloc (NUMBINS);
        if (connect==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }
        zeroGrid(connect);
    }
    const int max = NUMBINS;
    int steps=0;
}

```

```

    } else {
        error+=;
    }
}

err << "done [ " << voxels << " vox changed ]" << endl;
//err << "done [ " << error << " errors : " <<
//int(1000.0*error/(voxels+error))/10.0 << "% ]" << endl;
err << endl;
return voxels;
};

int intersect_Grids (gridpt grid1[], gridpt grid2[]) {
    //GRID1 WILL CHANGE
    int voxels=0;
    int changed=0;
    //Float count = 0;
    //const float cat = NUMBINS/60.0;
    //Float cut = cat;
    //printf("BarO);

    for(unsigned int pt=0; pt<NUMBINS; pt++) {
        /*count++;
        if(count > cut) {
            cerr << "x" << flush;
            cut += cat;
        }*/
        if(grid1[pt]) {
            if(grid2[pt]) {
                changed++;
                grid1[pt] = 0;
            } else {
                voxels++;
            }
        }
    }
    //cerr << endl;
    cerr << "done [ " << changed << " vox changed ]" << flush;
    cerr << voxels << " vox overlap : " << flush;
    cerr << int(1000.0*voxels/(voxels+changed))/10.0 << "% ]" << flush;
    cerr << endl << endl;
    return voxels;
};

*****+
*****+ POINT BASED FUNCTIONS
*****+
int fill_AccessGrid (const float x, const float y, const float z,
                     const float R, gridpt grid1) {
    // could redo to simplify
    const float cutoff = (R / GRID)*R / GRID;
    const int imin = int((x - XMIN - R)/GRID - 1.0);
    const int jmin = int((y - YMIN - R)/GRID - 1.0);
    const int kmin = int((z - ZMIN - R)/GRID - 1.0);
    const int imax = int((x - XMIN + R)/GRID + 1.0);
    const int jmax = int((y - YMIN + R)/GRID + 1.0);
    const int kmax = int((z - ZMIN + R)/GRID + 1.0);

    const float xk = (x - XMIN)/GRID;
    const float yk = (y - YMIN)/GRID;
    const float zk = (z - ZMIN)/GRID;

    float distsq;
    int filled=0;
    for(int di=imin; di<imax; di++) {
        for(int dj=jmin; dj<jmax; dj++) {
            for(int dk=kmin; dk<kmax; dk++) {
                distsq = (xk-di)*(xk-di) + (yk-dj)*(yk-dj) + (zk-dk)*(zk-dk);
                if(distsq < cutoff) {
                    int pt = ijk2pt(di,dj,dk);
                    if(!grid1[pt]) {
                        grid1[pt] = 1;
                        filled++;
                    }
                }
            }
        }
    }
    return filled;
};

void empty_ExcludeGrid (const int i, const int j, const int k,
                       const float probe, gridpt grid[]) {
    //provides indexes (i,j,k) of grid where ijk2pt(i,j,k) = gridpt
    const float R = probe/GRID; //Aug 19: correction for oversize
    const int r = int(R+1);
    const float cutoff = R*R;
    int mri,nrj,nrk,prj,prk;
    float distsq;
    //overflow checks (let's not go off the grid)
    if(i < r) { mri = -i; } else { mri = -r; }
    if(j < r) { nrj = -j; } else { nrj = -r; }
    if(k < r) { nrk = -k; } else { nrk = -r; }
    if(i + r > DX) { prj = DX-i-1; } else { prj = r; }
    if(j + r > DY) { prj = DY-j-1; } else { prj = r; }
    if(k + r > DZ) { prk = DZ-k-1; } else { prk = r; }

    int ind;
    for(int di=mri; di<pri; di++) {
        for(int dj=nrj; dj<prj; dj++) {
            for(int dk=nrk; dk<prk; dk++) {
                ind = ijk2pt(di,dj,dk);
                if(grid[ind]) {
                    distsq = di*di + dj*dj + dk*dk;
                    if(distsq < cutoff) {
                        grid[ind] = 0;
                    }
                }
            }
        }
    }
    return;
};

void fill_ExcludeGrid (const int i, const int j, const int k,
                      const float probe, gridpt grid[]) {
    //provides indexes (i,j,k) of grid where ijk2pt(i,j,k) = gridpt
    const float R = probe/GRID; //Aug 19: correction for oversize
    const int r = int(R+1);
    const float cutoff = R*R;
    int mri,nrj,nrk,prj,prk;
    float distsq;
    //overflow checks (let's not go off the grid)
    if(i < r) { mri = -i; } else { mri = -r; }
    if(j < r) { nrj = -j; } else { nrj = -r; }
    if(k < r) { nrk = -k; } else { nrk = -r; }
    if(i + r > DX-1) { prj = DX-i-1; } else { prj = r; }
    if(j + r > DY) { prj = DY-j-1; } else { prj = r; }
    if(k + r > DZ) { prk = DZ-k-1; } else { prk = r; }

    int ind;
    for(int di=mri; di<pri; di++) {
        for(int dj=nrj; dj<prj; dj++) {
            for(int dk=nrk; dk<prk; dk++) {
                ind = ijk2pt(di,dj,dk);
                if(grid[ind]) {
                    distsq = di*di + dj*dj + dk*dk;
                    if(distsq < cutoff) {
                        grid[ind] = 0;
                    }
                }
            }
        }
    }
    return;
};

```

```

    distsq = di*di + dj*dj + dk*dk;
    if(distsq < cutoff) {
        grid[ind] = 1;
    }
}
return 1;
}

int ijk2pt(int i, int j, int k) {
    return int(i+j*DXY+k*DXY);
}

bool isEdgePoint (const int i, const int j, const int k, gridpt grid[]) {
    //look at neighbors
    short int count=0;
    for(int di=k-DXY; di<=k+DXY; di+=2*DXY) {
        count++;
        if(grid[di+j+dk]) {
            return 1;
        }
    }
    for(int dj=j-DX; dj<=j+DX; dj+=2*DX) {
        count++;
        if(grid[di+dj+k]) {
            return 1;
        }
    }
    for(int di=i-1; di<=i+1; di+=2) {
        count++;
        if(grid[di+j+k]) {
            return 1;
        }
    }
    if(count != 6) { cerr << "EdgePoint count " << count << " != 6" << endl; }
    return 0;
}

bool isEdgePoint_Fill (const int pt, gridpt grid[]) {//look at neighbors
short int count=0;
for(int di=-1; di<=1; di++) {
    for(int dj=-DX; dj<=DX; dj+=DX) {
        for(int dk=-DXY; dk<=DXY; dk+=DXY) {
            count++;
            if(!grid[pt+di+dj+dk]) {
                return 1;
            }
        }
        //cerr << "!" << endl;
        if(count != 27) { cerr << "EdgePoint count " << count << " != 27" << endl; }
        return 0;
    }
}

bool isEdgePoint_Star (const int pt, gridpt grid[]) {
    //look at neighbors
    short int count=0;
    for(int di=-1; di<=1; di+=2) {
        count++;
        if(!grid[pt+di]) {
            return 1;
        }
    }
    for(int dj=-DX; dj<=DX; dj+=2*DX) {
        count++;
        if(!grid[pt+dj]) {
            return 1;
        }
    }
}

for(int di=-DXY; di<=DXY; di+=2*DXY) {
    if(count != 6) { cerr << "EdgePoint count " << count << " != 6" << endl; }
    return 0;
}

void expandPoint (const int pt, gridpt grid[]) {
    if(count != 6) { cerr << "EdgePoint count " << count << " != 6" << endl; }
    return 0;
}

//void contract_Paint (const int pt, gridpt grid[]);
//void contract_IN
strcpy(line, "HETATM");

//ATOM NUMBER
char temp[128];
temp[0] = 'O';
sprintf(temp, "%d", n%99999+1);
padleft(temp, 5);
strcat(line,temp);

//LEAD IN
strcat(line,'HETATM');

//ATOM & RESIDUE TYPES
strcat(line, " O H OH ");
//RESIDUE NUMBER
sprintf(temp, "%d", (n/10)%9999+1);
padleft(temp, 4);
strcat(line,temp);

//Gap
strcat(line, "  ");

//XYZ COORDINATES 4.3
float x = float(i)*GRID + XMIN;
sprintf(temp, "%f", x);
padleft(temp, 8);
strcat(line,temp);
float Y = float(j)*GRID + YMIN;
sprintf(temp, "%f", y);
padleft(temp, 8);
strcat(line,temp);
float z = float(k)*GRID + ZMIN;
sprintf(temp, "%f", z);
padleft(temp, 8);
strcat(line,temp);

//OCCUPANCY
sprintf(temp, " 1.00");
strcat(line,temp);

//TEMPERATURE
float dist = distFromPt(x,y,z);
sprintf(temp, "%f", dist);
padleft(temp, 6);
strcat(line,temp);

//PRINT OUT
//cerr << line << endl;
return;
}

void limitToTunnelArea(const float radius, gridpt grid[]) {

```

```

cerr << "Limiting to Cylinder Around Exit Tunnel..." << flush;

for(int pt=0; pt<=DXNZ; pt++) {
    grid[pt] = 0;
}
cerr << "done" << endl << endl;
return;
}

bool isCloseToVector (const float radius, const int pt) {
    if(isCloseToVector(radius,pt)) {
        grid[pt] = 0;
    }
    cerr << endl << endl << endl;
    return;
}

//GET QUERY POINT
const float x = int(pt % DX) * GRID + XMN;
const float y = int(pt % DXY) / DX * GRID + YMN;
const float z = int(pt / DXY) * GRID + ZMN;

//GET DISTANCE
float dist = distFromPt(x,y,z);

//RETURN
if(dist < radius) {
    return 1;
}
return 0;
}

float distFromPt (const float x, const float y, const float z) {
    //INIT POINT
    const float xp = 53.652;
    const float yp = 141.358;
    const float zp = 66.460;
    const float xp = 58.320;
    const float yp = 140.063;
    const float zp = 80.060;

    //VECTOR
    const float xv = 0.58092;
    const float yv = -0.60342;
    const float zv = 0.54627;

    //DIFFERENCE VECTOR
    const float dx = x - xp;
    const float dy = y - yp;
    const float dz = z - zp;
    const float lensq = dx*dx + dy*dy + dz*dz;
    const float dot = dx*xv + dy*yv + dz*zv;

    //GET CROSS PRODUCT
    const float cross = sqrt(lensq - dot*dot);
    return cross;
}

float crossSection (const real p, const vector v, const gridpt grid[])
{
    return crossSection(grid);
}

float crossSection (const gridpt grid[])
{
    //INIT POINT
    struct real p;
    p.x = 77.0;
    p.y = 124.0;
}
```

```

    p.z = 99.0;

    //TUNNEL VECTOR
    struct vector v1;
    v1.x = -0.58092;
    v1.y = 0.60342;
    v1.z = -0.54627;

    //GENERATE 2D GRID
    //FIND 2 PERP VECTORS
    struct vector v1, v2;
    v1.x = 0.60342; //=v.y
    v1.y = 0.58092; //=-v.x
    v1.z = 0.00000; //=0
    v2.x = -0.31734; //=0
    v2.y = 0.32963;
    v2.z = 0.70159;

    //const float x = int(pt % DX) * GRID + XMN;
    //const float y = int(pt % DXY) / DX * GRID + YMN;
    //const float z = int(pt / DXY) * GRID + ZMN;

    struct real r;
    struct gridpt rp;
    int pt;
    float count;
    double mult = GRID*GRID*.5*.5*2.0/3.0;
    const float total = 0.0;
    for(float i=200; i<=200; i+=GRID*.5) {
        for(float j=200; j<=200; j+=GRID*.5) {
            cerr << "Stepping" << flush;
            for(float k=5; k<100; k+=0.5) {
                const int ip = int((x-XMIN)/GRID+0.5);
                const int jp = int((y-YMIN)/GRID+0.5);
                const int kp = int((z-ZMIN)/GRID+0.5);
                const int count = 0;
                float total = 0.0;
                for(float i=200; i<=200; i+=GRID*.5) {
                    for(float j=200; j<=200; j+=GRID*.5) {
                        r.x = p.x + v1.x*i + v2.x*j + v1.y*k;
                        r.y = p.y + v1.y*i + v2.y*j + v1.z*k;
                        r.z = p.z + v1.z*i + v2.z*j + v1.w*k;
                        if(r.x >= XMN && r.x <= XMAX &&
                           r.y >= YMN && r.y <= YMAX &&
                           r.z >= ZMN && r.z <= ZMAX) {
                            rp.i = int((r.x-XMIN)/GRID+0.5);
                            rp.j = int((r.y-YMIN)/GRID+0.5);
                            rp.k = int((r.z-ZMIN)/GRID+0.5);
                            pt = rp.i + rp.j*DX + rp.k*DXY;
                            if(pt >= 0 && pt < DXNZ) {
                                total++;
                                if(grid[pt]) { count++; }
                            }
                        }
                    }
                }
                if(count > 0) {
                    cout << k << endl;
                }
            }
            cout << endl;
        }
    }
    return count;
}

```

Page 17 of 22

```

wt[0]=0.0; wt[1]=0.894; wt[2]=1.349; wt[3]=1.587;
wt[4]=4.0; wt[5]=2.6667; wt[6]=3.3333; wt[7]=1.79; wt[8]=2.68; wt[9]=4.08;
*/
int type; // for return variables
int edges[10]; //count types
for(int i=0, i<9, i++) { edges[i] = 0; }
float count = 0;
const float cat = DZ/60.0;
float cut = cat;

//cerr << "DXY: " << DXY << "\tDX: " << DX << endl;
//cerr << "Count Surface Voxels for Surface Area..." << endl;
printBar();

int sk=-1, sj=-1;
for(int k=0, k<DXY; k+=DXY) {
    sk++;
    if(count > cut) {
        cerr << "X" << flush;
        cut += cat;
    }
    sj=-1;
    for(int j=0; j<DXY; j+=DX) {
        sj++;
        for(int i=0; i<DX; i++) {
            int pt = i+j+k;
            if(grid[pt]) {
                /cerr << pt << ":";
                type = classifyEdgePoint(pt, grid);
                /surf += wt[type];
                edges[type]++;
            }
        }
    }
    cerr << endl << "EDGES: ";
    float totedge = 0;
    for(int i=1; i<9, i++) {
        totedge += float(edges[i]);
    }
    for(int i=1; i<9; i++) {
        cerr << "S" << i << ":" << int((Float(1000*edges[i])/totedge)/10.0));
        surf += edges[i]*wt[i];
    }
    cerr << endl;
    return surf*GRID*GRID;
}

int classifyEdgePoint (const int pt, gridpt grid[])
{
    //look at neighbors
    short int count=0;
    short int nb=0; //num of empty neighbors
    for(int di=-1; di<=1; di+=2) {
        count++;
        if((grid[pt+di])) {
            nb++;
        }
    }
    for(int dj=-DX; dj<=DX; dj+=2*DX) {
        count++;
        if((grid[pt+dj])) {
            nb++;
        }
    }
    for(int dk=-DXY; dk<=DXY; dk+=2*DXY) {
        count++;
        if((grid[pt+dk])) {
            nb++;
        }
    }
}

```

```

int fill_cavities(gridpt grid[]) {
    gridpt cavACC=NULL;
    cavACC = (gridpt) malloc(NUMBINS);
    bounding_box(grid, cavACC);

    //Create inverse access map
    subt_grid(cavACC, grid); // modifies cavACC;
    /int chanAC_C_voxels = countGrid(cavACC);

    //Get first point
    bool stop = 1; int firstpt = 0;
    for(unsigned int pt=0; pt<NUMBINS && stop; pt++)
        if(cavACC[pt]) { stop = 0; firstpt = pt; }

    cerr << "FIRST POINT: " << firstpt << endl;

    //LAST POINT
    stop = 1; int lastpt = 0;
    for(unsigned int pt=NUMBINS-10; pt>0 && stop
        if(cavACC[pt]) { stop = 0; lastpt = pt; }

    cerr << "LAST POINT: " << lastpt << endl;

    //Pull channels out of inverse access map
    gridpt* chanACC=NULL;
    chanACC = (gridpt*) malloc(NUMBINS);
    zeroGrid(chanACC);
    get_Connected_Point(cavACC, chanACC, firstpt);
    get_Connected_Point(cavACC, chanACC, lastpt); /int chanAC_C_voxels = countGrid(chanACC);

    //Subtract channels from access map leaving cavACC
    subt_grid(cavACC, chanACC); // modifies cavACC;
    int cavACC_voxels = countGrid(cavACC);

    int grid_before = countGrid(grid);
    //Fill cavities in grid[];
    for(unsigned int pt=0; pt<NUMBINS; pt++) {
        if(cavACC[pt]) { grid[pt]=-1; }
    }
    int grid_after = countGrid(grid);
    free(cavACC);

    cerr << endl << "CAVITY VOLUME: ";
    printVol(cavACC_voxels);
    cerr << endl << "BEFORE VOLUME: ";
    printVol(grid.before);
    cerr << endl << "AFTER VOLUME: ";
    printVol(grid.after);
    cerr << endl << "DIFFERENCE: ";
    printVol(grid.after-grid.before);
    cerr << endl << endl;

    return cavACC_voxels;
}

int bounding_box(gridpt grid[], gridpt bbox[])
//find min x,y,z and max x,y,z
zeroGrid(bbox);

//PART I: Determine Extrema
float D_center = 0.5;

```

```

const float cat = DZ/60.0f;
float zmin=cat;
err << "catminating Minima and Maxima . . ." << endl;
printBarO();
for(int k=0; k<DXY; k+=DX) {
    int xmin=0, ymin=0, ymax=0;
    for(int j=0; j<DXY; j+=DX, zmin+=DXZ) {
        count++;
        if(count > cut) {
            cerr << "x" << flush;
            cut += cat;
        }
        for(int i=j; i<DXY; i+=k) {
            int pt = i+j+k;
            if(grid[pt]) {
                if(i < xmin) { xmin = i; }
                if(j < ymin) { ymin = j; }
                if(k < zmin) { zmin = k; }
                if(i > xmax) { xmax = i; }
                if(j > ymax) { ymax = j; }
                if(k > zmax) { zmax = k; }
            }
        }
        cerr << endl << "DONE";
        cerr << endl << endl;
    }
}
/*Grow by one
xmin-=1;
ymin-=DX;
zmin-=DX;
xmax+=1;
ymax+=DX;
zmax+=DX;
*/
//PART II: FILL BOX
int vol=0;
count = 0;
cut = cat;
err << "Fill Box . . ." << endl;
printBarO();
for(int k=zmin; k<zmax; k+=DX) {
    count++;
    if(count > cut) {
        cerr << "x" << flush;
        cut += cat;
    }
    for(int j=ymin; j<ymax; j+=DX) {
        for(int i=xmin; i<xmax; i++) {
            bbox[i+j+k] = 1;
        }
    }
    cerr << endl << "BOX VOXELS: ";
    printVol(vol);
    cerr << endl << endl;
}
return vol;

```

```

/*
 ** output.cpp
 * /
#include "utils.h"

const float INV_SORT_2 = 0.7071;
const float INV_SORT_3 = 0.5774;

***** OUTPUT FUNCTIONS *****
void writeEzd (gridpt grid[], char outfile[]) {
    cerr << "Writing RIGID EZD to file:" << endl;
//starts
    int is=NUMBINS,js=NUMBINS,ks=NUMBINS;
//ends
    int ie=0,je=0,ke=0;
//PARSE
    const float pcat = NUMBINS/60.0;
    float pout = pcat;
    cerr << "Finding limits of the GRID..." << endl;
    printBar();
    for(unsigned int ind=0; ind<NUMBINS; ind++) {
        if(float(ind) > pcat) {
            cerr << "\n" << flush;
            pout += pcat;
        }
        if(grid[ind]) {
            const int i = int(ind % DO);
            const int j = int((ind % DXY) / DX);
            const int k = int(ind / DXY);
            if(i < is) { is = i; }
            if(j < js) { js = j; }
            if(k < ks) { ks = k; }
            if(i > ie) { ie = i; }
            if(j > je) { je = j; }
            if(k > ke) { ke = k; }
        }
    }
    cerr << endl;
//extend
    is--; js--; ie++; ke++;
//minmaxs
    const float xmin = is * GRID + XMIN;
    const float ymin = js * GRID + YMIN;
    const float zmin = ks * GRID + ZMIN;
    const float xmax = ie * GRID + XMIN;
    const float ymax = je * GRID + YMIN;
    const float zmax = ke * GRID + ZMIN;
    std::ofstream out;
    out << "Time(" << std::time(&t) << ")" << endl;
    //Den: 0.1 CELL = 10.0 (or maybe not, might be vector pointing axes)
    out << "CELL " << int(xmax-xmin+1) << ".0" << int(ymax-ymin+1) << ".0" << int(zmax-zmin+1) << ".0" << endl;
    //ORIGIN (x,y,z) in voxels (mult by $gridDens to get Angstroms)
}

//must cover min (x,y,z)
int OX = int(floor(xmin)/GRID-1.0);
int OY = int(floor(ymin)/GRID-1.0);
int OZ = int(floor(zmin)/GRID-1.0);
out << "ORIGIN " << OX << " " << OY << " " << OZ << endl;
//EXTENT maximum dimensions of actual cell (in voxels)
//must cover max (x,y,z)/defines range()
int MX = ie-is+1; //int(floor(xmax-xmin)/GRID+1.0);
int MY = je-js+1; //int(floor(ymax-ymin)/GRID+1.0);
int MZ = ke-ks+1; //int(floor(zmax-zmin)/GRID+1.0);
out << "EXTENT " << MX << " " << MY << " " << MZ << endl;
// out << "GRID " << MX << " " << DY << " " << DZ << endl; //big version
out << "SCALE 1.0" << endl;
out << "MAP" << endl;
int outcnt = 0;
float count = 0;
const float cat = MZ/60.0;
float cut = cat;
cut << endl;
cerr << "Writing the Grid to [" << outfile << "]..." << endl;
printBar();
for(int ks=ks; ks=ke; ks++) {
    count++;
    if(count > cut) {
        cerr << "\n" << flush;
        cut += cat;
    }
    for(int jj=js; jj=je; jj++) {
        if(grid[i+jDX+k:DXY]) {
            for(int jjj=j-1; jjj=je; jjj++) {
                if(grid[i+jDX+k:DXY]) {
                    for(int ii=i; ii=ie; ii++) {
                        if(grid[ii+jj+1]) {
                            out << "1";
                        } else {
                            out << "0";
                        }
                    }
                    out << endl;
                    if(outcnt % 7 == 0) { out << endl; }
                }
            }
            out << endl << "END" << endl;
            out.close();
            cerr << endl << "done [ wrote " << count << " lines ]"
        }
    }
}
};

void writeHalfEzd (gridpt grid[], char outfile[]) {
    cerr << "Calculating HALVED EZD for file:" << outfile << endl;
    printBar();
    for(unsigned int ind=0; ind<NUMBINS; ind++) {
        if(grid(ind)) > pcat) {
            const float pcat = NUMBINS/60.0;
            const float neagrid = GRID/2.0;
            float pout = pcat;
            cerr << "Finding limits of the GRID..." << endl;
//starts
            int is=NUMBINS,js=NUMBINS,ks=NUMBINS;
//ends
            int ie=0,je=0,ke=0;
//PARSE
            const float pcat = NUMBINS/60.0;
            const float neagrid = GRID/2.0;
            float pout = pcat;
            cerr << "Finding limits of the GRID..." << endl;
            printBar();
            for(unsigned int ind=0; ind<NUMBINS; ind++) {
                if(grid(ind)) > pcat) {
                    cerr << "\n" << flush;
                    pout += pcat;
                }
            }
            out << "Time(" << std::time(&t) << ")" << endl;
            //Den: 0.1 CELL = 10.0 (or maybe not, might be vector pointing axes)
            out << "CELL " << int(xmax-xmin+1) << ".0" << int(ymax-ymin+1) << ".0" << int(zmax-zmin+1) << ".0" << endl;
            //ORIGIN (x,y,z) in voxels (mult by $gridDens to get Angstroms)
}

```

```

if(i < is) { is = i; }
if(j < js) { js = j; }
if(k < ks) { ks = k; }
if(i > ie) { ie = i; }
if(j > je) { je = j; }
if(k > ke) { ke = k; }
}
}
cerr << endl;
//extend
js-=3; ks-=3; ie+=3; je+=3; ke+=3;
//minmax
const float xmin = is * GRID + XMIN;
const float ymin = js * GRID + YMIN;
const float zmin = ks * GRID + ZMIN;
const float xmax = ie * GRID + XMIN;
const float ymax = je * GRID + YMIN;
const float zmax = ke * GRID + ZMIN;
std::ofstream out;
time t;
out.open(outfile);
out << "#EDZMAP" << endl;
out << "EDZ file (c) Neil Voss, 2005" << endl;
std::time(&t);
out << "Grid: " << "\tGRIDVOL" << GRIDVOL << "\tWater_Res: " << WATER_RES
out << "\tMaxProbe: " << MAXPROBE << "\tCutoff: " << CUTOFF << endl;
out << "Input File: " << XYZFILE << endl;
out << "THIS GRID IS HALVED" << endl;
out << "Date: " << std::ctime(&t) << endl;
out << "CELL size based on GRID NOT on EXTENT, given GRID 100 and"
///CELL: CELL = 10.0 (or maybe not, might be better)
out << "CELL" << int(xmax-xmin+1) << ".0" << int(ymax-ymin+1)
<< ".0" << int(zmax-zmin+1) << ".0" << "90.0 90.0" << endl;
///ORIGIN (x,y,z) in voxels (multi by $gridDens to get Angstroms)
//must cover min (x,y,z)
}

int OX = int((float(xmin)/GRID-1.0)/2.0+0.5);
int OY = int((float(ymin)/GRID-1.0)/2.0+0.5);
int OZ = int((float(zmin)/GRID-1.0)/2.0+0.5);
out << "ORIGIN" << OX << " " << OY << " " << OZ << endl;
///EXTENT maximum dimensions of actual cell (in voxels)
//must cover max (x,y,z) /defines range(r)
int MX = int((ie-is+1)/2.0+0.5); //int((float(xmax-xmin)/GRID)+1.0);
int MY = int((je-js+1)/2.0+0.5); //int((float(ymax-ymin)/GRID)+1.0);
int MZ = int((ke-ks+1)/2.0+0.5); //int((float(zmax-zmin)/GRID)+1.0);
out << "EXTENT" << MX << " " << MY << " " << MZ << endl;
// out << "EXTENT" << DX << " " << DY << " " << DZ << endl; //big version
out << "GRID" << MX << " " << MY << " " << MZ << endl; //doesnt matter
out << "SCALE 1.0" << endl;
out << "MAP" << endl;
int outcut = 0;
float count = 0;
const float cat = MZ/60.0;
float cut = cat;

cerr << "Halving GRID and Writing to EDZ File: " << outfile << endl;
cerr << "THIS MAY TAKE A LONG TIME" << endl;
printBarO;
for(int j=js; j<=je; j+=2) {
    for(int k=ks; k<=ke; k+=2) {
        count++;
        if(count > cut) {
            cerr << " " << flush;
            cut += cat;
        }
    }
}
}

void writeBlueEDZ (gridpt grid[], char outfile[]) {
    const int i = int(ind % DX);
    const int j = int((ind % DXY) / DX);
    const int k = int(ind / DXY);
    if(i < is) { is = i; }

    if(grid[i]) {
        const int l = int(ind % DX);
        const int m = int((ind % DXY) / DX);
        const int n = int(ind / DXY);
        if(l < ls) { ls = l; }

        if(l == ls) {
            if(m == ms) {
                if(n == ns) {
                    if(grid[l][m][n] > 0) {
                        cout << " " << flush;
                    }
                }
            }
        }
    }
}

void write_EDZ (gridpt grid[], char outfile[]) {
    const int i = int(ind % DX);
    const int j = int((ind % DXY) / DX);
    const int k = int(ind / DXY);
    if(i < is) { is = i; }

    if(grid[i]) {
        const int l = int(ind % DX);
        const int m = int((ind % DXY) / DX);
        const int n = int(ind / DXY);
        if(l < ls) { ls = l; }

        if(l == ls) {
            if(m == ms) {
                if(n == ns) {
                    if(grid[l][m][n] > 0) {
                        cout << " " << flush;
                    }
                }
            }
        }
    }
}

void write_EDZ (gridpt grid[], char outfile[]) {
    const int i = int(ind % DX);
    const int j = int((ind % DXY) / DX);
    const int k = int(ind / DXY);
    if(i < is) { is = i; }

    if(grid[i]) {
        const int l = int(ind % DX);
        const int m = int((ind % DXY) / DX);
        const int n = int(ind / DXY);
        if(l < ls) { ls = l; }

        if(l == ls) {
            if(m == ms) {
                if(n == ns) {
                    if(grid[l][m][n] > 0) {
                        cout << " " << flush;
                    }
                }
            }
        }
    }
}

```

```

    if(j < js) { js = j; }
    if(k < ks) { ks = k; }
    if(i > ie) { ie = i; }
    if(j > je) { je = j; }
    if(k > ke) { ke = k; }
}
cerr << endl;
//extend
is-=3; js-=3; ks-=3; ie+=3; je+=3; ke+=3;
//minmaxs
const float xmin = is * GRID + XMIN;
const float ymin = js * GRID + YMIN;
const float zmin = ks * GRID + ZMIN;
const float xmax = ie * GRID + XMIN;
const float ymax = je * GRID + YMIN;
const float zmax = ke * GRID + ZMIN;

std::ofstream out;
time_t t;
out.open(outfile);
out << "#ED_MAP" << endl;
out << "# ED File (c) Neil Voss, 2005" << endl;
out << "# Grid: " << GRID << "\tGRIDVOL" << GRIDVOL << "\tWater Res: " << WATER_RES
out << "# MaxProbe: " << MAXPROBE << "\tCutoff: " << CUTOFF << endl;
out << "# Cell: " << GRID << "\tMAXPROBE" << endl;
out << "# Input File: " << XYZFILE << endl;
out << "# Date: " << std::ctime(&t) << endl;
//CELL: cell size based on GRID NOT on EXTENT, given GRID 100 and
//CELL: (or maybe not, might need to give vector pointing axes)
out << "#CELL: " << int(xmax-xmin+1) << "\t0.0" << int(ymax-ymin+1)
<< "# CELL: " << int(zmax-zmin+1) << "\t0 90.0 90.0" << endl;
//ORIGIN (x,y,z) in voxels (multiplied by $gridDens to get Angstroms)
//must cover min (x,y,z)
int OX = int(float(xmin)/GRID-1.0);
int OY = int(float(ymin)/GRID-1.0);
int OZ = int(float(zmin)/GRID-1.0);
out << "#ORIGIN " << OX << " " << OY << " " << OZ << endl;
//EXTENT: maximum dimensions of actual cell (in voxels)
//must cover max (x,y,z) //defines range()
int MX = ie-1; //int(float(xmax-xmin)/GRID)+1.0;
int MY = je-1; //int(float(ymax-ymin)/GRID)+1.0;
int MZ = ke-1; //int(float(zmax-zmin)/GRID)+1.0;
out << "#EXTENT " << MX << " " << MY << " " << MZ << endl;
// out << "#EXTENT " << DX << " " << DY << " " << DZ << endl; //big version
out << "#GRID " << MX << " " << MY << " " << MZ << endl; //doesn't matter
out << "#MAP" << endl;
int outcut = 0;
float count = 0;
const float cat = MZ/60.0;
float cut = cat;
const float fill = 21.1044; //8*INV_SQRT_3 + 12*INV_SQRT_2 + 6*1 + 2;
printBar();
for(int k=ks; k<ke; k++) {
    count++;
    if(count > cut) {
        cerr << "\n" << flush;
    }
}
for(int j=js; j<je; j++) {
    for(int i=is; i<ie; i++) {
        int index = i+j*DX+k*DXY;
        float sum = 0.0;
        for(int sk=1; sk<=1; sk++) {
            if(grid[index+i*DXY+k*DXY] < 0.5) {
                sum += abs(index+i*DXY+k*DXY);
            }
        }
        if(sum > 0.5) {
            if(sum <= 21) {
                out << int(10000.0*sum/fill+0.5)/10000.0 << " " << flush;
            } else if(sum >= 21) {
                out << "1" << endl;
            } else if(sum == 1) {
                out << "0" << endl;
            } else if(sum == 2) {
                out << "INV_SORT_2;" << endl;
            } else if(sum == 3) {
                out << "INV_SORT_3;" << endl;
            }
        }
    }
}
if(sum > 0.5 && sum < 21) {
    if(sum > int(10000.0*sum/fill+0.5)) {
        out << int(10000.0*sum/fill+0.5)/10000.0 << " " << flush;
    } else if(sum >= 21) {
        out << "1" << endl;
    } else if(sum == 1) {
        out << "0" << endl;
    } else if(sum == 2) {
        out << "INV_SORT_2;" << endl;
    } else if(sum == 3) {
        out << "INV_SORT_3;" << endl;
    }
}

void write_PDB (gridpt grid[], char outfile[], char file) {
    std::ofstream out;
    out << "Writing PDB to file:" << outfile << endl;
    std::ofstream std::ostream out;
    out << "#REMARK (c) Neil Voss, 2005" << endl;
    out << "#REMARK PDB file created from " << XYZFILE << endl;
    out << "#REMARK Grid: " << GRID << "\tGRIDVOL: " << GRIDVOL << endl;
    out << "#REMARK Cutoff: " << CUTOFF << endl;
    std::time_t t;
    out << "#REMARK Date: " << std::ctime(&t) << endl;
    out << "#REMARK MaxProbe: " << MAXPROBE << endl;
    out << "#REMARK Anum: " << anum << endl;
    const float cat = DZ/60.0;
    float cut = cat;
    char line[128];
    cerr << "Writing the grid to [ " << outfile << " ]..." << endl;
    printBar();
    int sk=-1;
    for(int k=0; k<XNZ; k+=DXY) {
        sk++;
        if(count > cut) {
            cerr << "\n" << flush;
        }
        if(count > cut) {
            cut += cat;
        }
        sk--;
        for(int j=0; j<DXY; j+=DXY) {
            sk++;
            if(grid[i+j*DXY] < 0.5) {
                cerr << "\n" << flush;
            }
        }
    }
}

```

```

    ijk2pdb(line.i,sj,sk,anum);
    out << line << endl;
}
}

out << endl;
out.close();
cerr << endl << "done wrote " << anum << " atoms" << endl << endl;
return;
};

void writeSurfPDB (gridpt grid[], char outfile[]) {
    cerr << "Writing SURFACE PDB to file:" << outfile << endl;
    std::ofstream out;
    time_t t;
    out.open(outfile);
    out << "REMARK (C) Neil Voss, 2005" << endl;
    out << "REMARK PDB file created from " << XYZBFILE << endl;
    out << "REMARK Grid: " << GRID << "\tGRIDVOL: " << GRIDVOL;
    out << "REMARK Res: " << WATER_RES;
    out << "\tMaxProbe: " << MAXPROBE << "\tCutoff: " << CUTOFF << endl;
    std::time(&t);
    out << "REMARK Date: " << std::ctime(&t) << flush;
    float count = 0;
    int anum=0, pnum=0;
    const float cat = DZ/60.0;
    float cut = cat;
    char line[128];
    cerr << "Writing the grid to [" << outfile << "]..." << endl;
    printBar();
    int sj=-1, sk=-1;
    for(int k=0; k<DXZ; k+=DXY) {
        sk++;
        count++;
        if(count > cut) {
            cerr << " " << flush;
            cut += cat;
        }
        sj++;
        for(int j=0; j<DXY; j+=DX) {
            for(int i=0; i<DX; i++) {
                int pt = i+j+k;
                if(grid[pt]) {
                    pnum++;
                    if(isEdgePoint_Star(pt,grid)) {
                        anum++;
                        ijk2pdb(line.i,sj,sk,anum);
                        out << line << endl;
                    }
                }
            }
        }
    }
    out << endl;
    out.close();
    cerr << endl << "done wrote " << anum << " of " << pnum << endl << endl;
    return;
};

int ijk2pdb(char line[], int i, int sj, int sk, int anum) {
    cout << "ijk2pdb(" << line << ", " << i << ", " << sj << ", " << sk << ", " << anum << ")" << endl;
}

void writeThirdED (gridpt grid[], char outfile[]) {
    cerr << "Calculating THIRD ED for file: " << outfile << endl;
    //starts
    int is=NUMBINS,js=NUMBINS,ks=NUMBINS;
    //ends
    int ie=0,je=0,ke=0;
}

```

```

//PARSE
{
    const float pcat = NUMBINS/60.0;
    const float negrid = GRID*3.0;
    float pcut = pcat;
    cerr << "Finding limits of the GRID..." << endl;
    printBar();
    for(unsigned int ind=0; ind<NUMBINS; ind++) {
        if(floating(ind) > pcut) {
            cerr << " " << flush;
            pcut += pcat;
        }
    }
}

if(grid[ind]) {
    const int i = int(ind % DX);
    const int j = int((ind % DXY) / DX);
    const int k = int(ind / DXY);
    if(i < is) { is = i; }
    if(j < js) { js = j; }
    if(k < ks) { ks = k; }
    if(i > ie) { ie = i; }
    if(j > je) { je = j; }
    if(k > ke) { ke = k; }
}

err << endl;
//extend
is-=4; js-=4; ks-=4; ie+=4; je+=4; ke+=4;
//minmax
const float xmin = is * GRID + XMIN;
const float ymin = js * GRID + YMIN;
const float zmin = ks * GRID + ZMIN;
const float xmax = ie * GRID + XMAX;
const float ymax = je * GRID + YMAX;
const float zmax = ke * GRID + ZMAX;

std::ofstream out;
time_t t;
out.open(outfile);
out << "EZD_MAP" << endl;
out << "EZD file (C) Neil Voss, 2005" << endl;
std::time(&t);
out << "Grid: " << GRID << "\tGRIDVOL" << GRIDVOL << "\tWater_Res: " << WATER_RES
     << "\tMaxProbe: " << MAXPROBE << "\tCutoff: " << CUTOFF << endl;
out << "Input File: " << XYZFILE << endl;
out << "THIS GRID IS THIRDED: NEW GRID SIZE: " << newgrid << endl;
out << "Date: " << std::time(&t) << endl;
//CELL: cell size based on GRID NOT on EXTENT, given GRID 100 and
//Dens 0.1 CELL = 10.0 (or maybe not, might be vector pointing axes)
out << "CELL: " << int((xmax-xmin)/10.0) << "\t0" << int((ymax-ymin)/10.0) << "\t0"
     << "\t0" << int((zmax-zmin)/10.0) << "\t0\t90.0\t90.0" << endl;
//ORIGIN (x,y,z) in voxels (mult by $gridBox to get Angstroms)
//must cover min (x,y,z)

int ox = int((float(xmin)/GRID-1.0)/(3.0+0.5));
int oy = int((float(ymin)/GRID-1.0)/(3.0+0.5));
int oz = int((float(zmin)/GRID-1.0)/(3.0+0.5));
out << "ORIGIN " << ox << " " << oy << " " << oz << endl;
//EXTENT maximum dimensions of actual cell (in voxels)
int mx = int((float(xmax-xmin)/GRID+1.0));
int my = int((float(ymax-ymin)/GRID+1.0));
int mz = int((float(zmax-zmin)/GRID+1.0));
out << "EXTENT " << mx << " " << my << " " << mz << endl;
// out << "GRID " << dx << " " << dy << " " << dz << endl; //big version
out << "SCALE 1.0" << endl;
out << "MAP" << endl;
int outcut = 0;
}

```

```

float count = 0;
const float cat = MZ/60.0;
float cut = cat;

cerr << "Thirding GRID and Writing to EZD File: " << outfile << endl;
cerr << ""THIS MAY TAKE A LONG TIME" << endl;
printBar();
for(int k=ks; k<ke; k+=3) {
    if(count > cut) {
        cerr << "x" << flush;
        cut += cat;
    }
    for(int j=is; j<=je; j+=3) {
        for(int i=is; i<=ie; i+=3) {
            int index = i+j*DX+*DX;
            //INTERIOR FACES: 3 vox * 3 vox = 27 vox
            //OUTSIDE EDGES: 9 vox * 6 faces = 54 vox
            //OUTSIDE CORNERS: 3 vox * 12 edges = 36 vox
            int interior = 0;
            int face = 0;
            int edge = 0;
            int corner = 0;
            for(int di=2; di<=2; di++) {
                for(int dj=2; dj<=2; dj++) {
                    for(int dk=2; dk<=2; dk++) {
                        int dist = di*di + dj*dj + dk*dk;
                        if(dist > 11) { //corner
                            if(gridIndex + di + dj*DX + dk*DX) { corner++; }
                        } else if(dist > 7) { //edge
                            if(gridIndex + di + dj*DX + dk*DX) { edge++; }
                        } else if(dist > 3) { //face
                            if(gridIndex + di + dj*DX + dk*DX) { face++; }
                        } else { //interior
                            if(gridIndex + di + dj*DX + dk*DX) { interior++; }
                        }
                    }
                }
            }
            int total = corner + edge + face + interior;
            //OUTPUT RANGE FROM ZERO TO ONE
            if(total == 0) {
                out << "0" << flush;
            } else if(total==125) {
                out << "1" << flush;
            } else {
                //INTERIOR = 6000/27 = 222.2 => 0.538
                //FACE = 2500/54 = 46.3 => 0.135
                //EDGE = 1300/36 = 36.1 => 0.100
                //CORNER = 200/8 = 25.0 => 0.227
                long int sum = interior*222.2 + face*46.3 +
                edge*36.1 + corner*36.1;
                out << double(sum)/100000.0 << " " << flush;
            }
            outcount++;
            if(outcnt % 7 == 0) { out << endl; }
        }
    }
    cerr << endl << "END" << endl;
    out.close();
    cerr << endl << "done [ wrote " << count << " lines ]"
    return;
}

void writeFifthED (gridpt grid[], char outfile[]) {
    cerr << "Calculating FIFTHED EZD for file: " << outfile << endl;
    cerr << "Writing FIFTHED EZD to file: " << outfile << endl;
    const float newgrid = GRID*5.0;
    const float pcat = NUMBINS/60.0;
    float pcut = pcat;
    cerr << "Finding Limits of the GRID..." << endl;
    printBarO();
    for(unsigned int ind=0; ind<NUMBINS; ind++) {
        if(float(ind) > pcut) {
            cerr << "x" << flush;
            pcut += pcat;
        }
        if(grid[ind]) {
            const int i = int(ind % DX);
            const int j = int((ind % DX)/ DX);
            const int k = int(ind / DX);
            if(i < is) { is = i; }
            if(i < js) { js = i; }
            if(k < ks) { ks = k; }
            if(k > ie) { ie = i; }
            if(j > je) { je = j; }
            if(k > ke) { ke = k; }
        }
    }
    cerr << endl;
    //extend
    is+=6; ks+=6; ie+=6; je+=6; ke+=6;
    //minmax
    const float xmin = is * GRID + YMIN;
    const float ymin = js * GRID + YMIN;
    const float zmin = ks * GRID + ZMIN;
    const float xmax = ie * GRID + YMIN;
    const float ymax = je * GRID + YMIN;
    const float zmax = ke * GRID + ZMIN;
    std::ofstream out;
    time_t t;
    out.open(outfile);
    out << "#EZD_MAP" << endl;
    out << "1 EZD file (c) Neil Voss, 2005" << endl;
    std::time(&t);
    out << "1 Grid;" << GRID << "\tGRIDVOL" << GRIDVOL << "\tWater_Res;" << WATER_RES
    out << "1 MaxProbe;" << MAXPROBE << endl;
    out << "1 XYZFILE" << endl;
    out << "1 Input File;" << INPUTFILE << endl;
    out << "1 THIS GRID IS FIFTHED; NEW GRID SIZE;" << newgrid << endl;
    out << "1 Date;" << std::date("%Y-%m-%d") << endl;
    //CELLS: cell size based on GRID NOT on EXTENT, given GRID 100 and
    //Dens: 0.1 CELL = 10.0 (or maybe not, might be vector pointing axes)
    out << "CELL" << int((xmax-xmin)/10.0) << ".0" << int((ymax-ymin)/10.0) << ".0" << int((zmax-zmin)/10.0) << ".0" << endl;
    //ORIGIN (x,y,z) in voxels (mult by $gridDens to get Angstroms)
    //must cover min (x,y,z)
    int OX = int((float(xmin)/GRID-1.0)/5.0+0.5);
    int OY = int((float(ymin)/GRID-1.0)/5.0+0.5);
    int OZ = int((float(zmin)/GRID-1.0)/5.0+0.5);
    out << "ORIGIN" << OX << " " << OY << " " << OZ << endl;
    //EXTENT: maximum dimensions of actual cell (in voxels)
    //must cover max (x,y,z) /defines range()
    int MX = int((je-is+1)/5.0+0.5); //int((float(xmax-xmin)/GRID+1.0));
    int MY = int((je-js+1)/5.0+0.5); //int((float(ymax-ymin)/GRID+1.0));
    int MZ = int((ke-ks+1)/5.0+0.5); //int((float(zmax-zmin)/GRID+1.0));
    out << "EXTENT" << MX << " " << MY << " " << MZ << endl;
    // out << "EXTENT" << DX << " " << DY << " " << DZ << endl; //big version
}

```

```

out << "GRID " << MX << " " << MY << " " << MZ << endl; //doesn't matter
out << "SCALE 1.0" << endl;
out << "MAP" << endl;
int outcnt = 0;
float count = 0;
const float cat = MZ/60.0;
float cut = cat;

cerr << "Thirding GRID and Writing to EDD File: " << outfile << endl;
cerr << "THIS MAY TAKE A LONG TIME" << endl;
printBarO;

for(int k=ks; k=ke; k+=5) {
    count++;
    if(count > cut) {
        cerr << " " << flush;
        cut += cat;
    }
    for(int jjs; j<je; j+=5) {
        for(int i=ls; i=ie; i+=5) {
            int index = i-j*DX+k*DXY;
            //INTERIOR:   5 * 5 vox - 8 + 6   = 123 vox
            //OUTSIDE FACES: 25 v * 6 faces + 8 - 6   = 152 vox
            //OUTSIDE EDGES: 5 vox * 12 edges   = 60 vox
            //OUTSIDE CORNERS: 8 vox
            int interior = 0;
            int face = 0;
            int edge = 0;
            int corner = 0;
            for(int di=3; di<=3; di++) {
                for(int dj=3; dj<=3; dj++) {
                    for(int dk=3; dk<=3; dk++) {
                        int dist = di*di + dj*dj + dk*dk;
                        if(dist > 26) { //corner
                            if(grdindex + di + dj*DXY + dk*DXY) { corner++; }
                            else if(dist > 17) { //edge
                                if(grdindex + di + dj*DXY + dk*DXY) { edge++; }
                            }
                            else if(dist > 9) { //face
                                if(grdindex + di + dj*DXY + dk*DXY) { face++; }
                            }
                            else { //interior, not perfect
                                if(grdindex + di + dj*DXY + dk*DXY) { interior++; }
                            }
                        }
                    }
                }
            }
            int total = corner + edge + face + interior;
            //OUTPUT RANGE FROM ZERO TO ONE
            if(total == 0) {
                out << "0" << flush;
            } else if(total == 343) {
                out << "1" << flush;
            } else {
                //INTERIOR   = 6000/123 = 48.8 => 0.538
                //FACE      = 3000/152 = 19.7 => 0.135
                //EDGE      = 900/60  = 15.0 => 0.100
                //CORNER,   = 100/8   = 12.5 => 0.227
                long int sum = interior*488 + face*197 +
                    edge*150 + corner*125; out << " " << flush;
            }
            outcnt++;
            if(outcnt % 7 == 0) { out << endl; }
        }
    }
    out << endl << "END" << endl;
    out.close();
    cerr << endl << "done [ wrote " << count << " lines ]"
    return;
}

```

```

#!/bin/sh
# extract x y z from PDB file, generate radius of each atom
# Hydrogens are presumed to be missing ("united atom" approach) unless -h given
# later: options for alternate radius and pattern files
# --- Mike Pique, The Scripps Research Institute
# input: pdb file as argument or stdin
# output: new xyzr file to stdout
# Options:
#   -h: use explicit hydrogen radii instead of default united atom radii
# examples:
#   # pdb_to_xyzr crambin.pdb > crambin.xyzr
#   # foo_to_pdb -h mol.foo | pdb_to_xyzr > mol.xyzr
#   # foo_to_pdb mol.foo | pdb_to_xyzr | awk '{print $4}' > mol.r
#   # set which field to use for radii:
#     h_select=5
#     if test $# -ge 1 ; then
#       if test ${1} = "-h" ; then
#         h_select=4
#       shift
#     fi
#   BEGIN{
#     # read radius table and patterns from supplied file
#     npats=0
#     numfile = "/atmtnumbers.dat"
#     while (getline < numfile) > 0 {
#       if(NF==0 || substr($1,1,1)=="#") continue;
#       if($1=="radius") {
#         n=$2 # atom number key
#         explicit_rad[$2] = $4
#       }
#     }
#     if(NF<=4 || substr($5,1,1)=="#") united_rad[n]=explicit_rad[n]
#     else united_rad[n] = $5
#     continue;
#   }
#   respat[npats] = ${1}
#   if(respat[npats] == "*") respat[npats] = ".*"
#   respat[npats] = ".\n" respat[npats] = "$"
#   atmpat[npats] = ".\n" atmpat[npats] = "$"
#   gsub("\n", "", atmpat[npats])
#   atmnum[npats] = $3
#   if( ! (atmnum[npats] in explicit_rad) ) {
#     # the key has no radius --- complain and fake one
#     print "pdb-to-xyzr: error in library file",numfile,
#     "entry '$1', '$2', '$3', 'has no corresponding radius value' \
#     | "cat 1>&2" # write to stderr
#     explicit_rad[atmnum[npats]] = 0.01
#     united_rad[atmnum[npats]] = 0.01
#   }
#   npats++
# }

# for(pat=0;pat<npats;pat++) print pat,respat[pat],atmpat[pat],atmnum[pat]

```

```

$1=="ATOM" || $1=="ATOM" || $1=="HETATM" || $1=="HETATM" {
  x = substr($0,31,8)
  y = substr($0,39,8)
  z = substr($0,47,8)
  resname=substr($0,18,3)

```

```

# atmtpenumerbers
# $Revision: 1.13 $
# Maps residue type and atom name pairs into Connolly "atm" numeric codes
# as used in MS and AIMS, and into actual radius values
#
# Format: (blank lines and lines beginning with # are ignored)
# Any number of blanks or tabs separate fields.
#
# Format of "radius" entries:
# Field 1 : keyword "radius"
# Field 2 : atom number (key)
# Field 3: covalent bond distance taken from Connolly "ms.rad" file (unchecked)
# Field 4: atomic radius (angstroms) taken from Connolly "ms.explicit.rad" file
# Field 5: optional atomic radius for united-atom ('no hydrogens') approach
# taken from Connolly "ms.rad" file
#
# By convention, radius entries come before atom type entries.
#
# Format of atom type entries:
# Field 1 : residue name pattern
# Field 2 : atom name pattern
# Field 3 : atom number (key value)
#
# Patterns are as in 'awk' or 'egrep', with the convention that '*' by itself
# as a field means '.', i.e., matches anything
# and that underscores in the atom name represent blanks
# and that underscores in the atom name represent blanks
# so 'P[0-99]' does not do what you might expect.
#
# First match is used, so put catch-all's at the end.
#
# Adapted from file "protein.typ" by Michael Connolly, John Tainer,
# and Elizabeth Getzoff.
# Michael Pique, Scripps Clinic
#
# Revision 1.1 90/02/27 12:34:46 mp
# Initial revision
#
#      #   atom num dist    Replicits Runited-atom
radius 1   0.57  1.40
radius 2   0.66  1.40  1.60
radius 3   0.57  1.40
radius 4   0.70  1.54  1.70
radius 5   0.70  1.54  1.80
radius 6   0.70  1.54  2.00
radius 7   0.77  1.74  2.00
radius 8   0.77  1.74  2.00
radius 9   0.77  1.74  2.00
radius 10  0.67  1.74  1.86
radius 11  0.70  1.74  1.85
radius 12  0.70  1.80  #
radius 13  1.04  1.80  # P, S, and LonePairs
radius 14  0.70  1.54  # non-protonated nitrogens
radius 15  0.37  1.20  # H, D hydrogen and deuterium
radius 16  0.70  0.00  1.50 # obsolete entry, purpose unknown
radius 17  3.50  5.00  # pseudatom - big ball
radius 18  1.74  1.97  # Ca calcium
radius 19  1.25  1.40  # Zn zinc (traditional radius)
radius 20  1.17  1.40  # Cu copper (traditional radius)
radius 21  1.45  1.30  # Fe heme iron
radius 22  1.41  1.49  # Cd cadmium
radius 23  0.01  0.01  # pseudatom - tiny dot
radius 24  0.37  1.20  0.00 # hydrogen vanishing if united-atoms
radius 25  1.16  1.24  # Fe not in heme
radius 26  1.36  1.60  # Mg magnesium
radius 27  1.17  1.24  # Mn manganese
radius 28  1.16  1.25  # Co cobalt

```

```

radius 29  1.17  2.15 # Se selenium
radius 30  3.00  3.00 # obsolete entry, original purpose unknown
radius 31  1.15  1.15 # Yb Ytterbium +3 ion --- wild guess only
radius 38  0.95  1.80 # obsolete entry, original purpose unknown
#
# note that the metal values are UNCHARGED radii, see
# http://www.shef.ac.uk/chemistry/web-elements for info - Mike Pique
#
# Hydrogens - not differentiated here
*  *(0-1)*H.* 15
*  *(0-9)*D.* 15
#
# Waters and confusingly-named metals
WAT1OH|H2O|DOP1DLS 0.* 2
CA  CA 18
CD  CD 22
*  CD_ 22
#
#Ribonucleotides added by Neil Voss ca. 2003
#
#sugars
#A|G|C|U|ADE|GUA|CYT|UR1|OMG|UR3|OMU|PSU|5MU|A23|T6A CL'
#A|G|C|U|ADE|GUA|CYT|UR1|OMG|UR3|OMU|PSU|5MU|A23|T6A C2'
#A|G|C|U|ADE|GUA|CYT|UR1|OMG|UR3|OMU|PSU|5MU|A23|T6A C3'
#A|G|C|U|ADE|GUA|CYT|UR1|OMG|UR3|OMU|PSU|5MU|A23|T6A C4'
#A|G|C|U|ADE|GUA|CYT|UR1|OMG|UR3|OMU|PSU|5MU|A23|T6A C5'
#A|G|C|U|ADE|GUA|CYT|UR1|OMG|UR3|OMU|PSU|5MU|A23|T6A O2'
#A|G|C|U|ADE|GUA|CYT|UR1|OMG|UR3|OMU|PSU|5MU|A23|T6A 02'
#A|G|C|U|ADE|GUA|CYT|UR1|OMG|UR3|OMU|PSU|5MU|A23|T6A 03'
#A|G|C|U|ADE|GUA|CYT|UR1|OMG|UR3|OMU|PSU|5MU|A23|T6A 04'
#A|G|C|U|ADE|GUA|CYT|UR1|OMG|UR3|OMU|PSU|5MU|A23|T6A 05'
#A|G|C|U|ADE|GUA|CYT|UR1|OMG|UR3|OMU|PSU|5MU|A23|T6A P 13
#A|G|C|U|ADE|GUA|CYT|UR1|OMG|UR3|OMU|PSU|5MU|A23|T6A O1P 1
#A|G|C|U|ADE|GUA|CYT|UR1|OMG|UR3|OMU|PSU|5MU|A23|T6A O2P 1
*
*  CL' 7
*  C2' 7
*  C3' 7
*  C4' 7
*  C5' 8
*  O2' 2
*  C1\* 7
*  C2\* 7
*  C3\* 7
*  C4\* 7
*  C5\* 8
*  O2\* 2
*  C3\* 1
*  C4\* 1
*  O5' 1
*  C1\*\* 7
*  C2\*\* 7
*  C3\*\* 7
*  C4\*\* 7
*  C5\*\* 8
*  O2\*\* 2
*  C3\*\* 1
*  C4\*\* 1
*  O5\*\* 1
*  P 13
*  O1P 1
*  O2P 1
GTP  O,A 1
GTP  O,B 1
GTP  O,G 1
#
#purines
1MA  CM1  11
A|G|G|A|ADE|OMG|A23|T6A|GTP|IMA N2  5
A|ADE|A23|T6A|IMA C2  10
G|G|A|GTP|OMG N1  4
G|G|A|GTP|OMG A|G|G|A|ADE|OMG|A23|T6A|GTP|IMA C4  10
A|G|G|A|ADE|OMG|A23|T6A|GTP|IMA A|G|G|A|ADE|OMG|A23|T6A|GTP|IMA C5  10
A|G|G|A|ADE|OMG|A23|T6A|GTP|IMA A|G|G|A|ADE|OMG|A23|T6A|GTP|IMA C6  10
G|G|A|GTP|OMG 06  1

```

```

A|ADE|A23|TGA|1MA          N6   5
A|G|GIA|ADE|OMG|A23|T6A|GTP|1MA      N7   4
A|G|GIA|ADE|OMG|A23|T6A|GTP|1MA      C8   11
OMG                           CM2   11
A|G|GIA|ADE|OMG|A23|T6A|GTP|1MA      N9   4

#pyrrolidines
U|C|URU|CYT|UR3|OMU|PSU|5MU      N1   4
U|C|URU|CYT|UR3|OMU|PSU|5MU      C2   10
U|C|URU|CYT|UR3|OMU|PSU|5MU      O2   1
U|C|URU|CYT|UR3|OMU|PSU|5MU      N3   4
U|C|URU|CYT|UR3|OMU|PSU|5MU      C4   10
U|C|URU|CYT|UR3|OMU|PSU|5MU      C3U  10
U|C|URU|CYT|UR3|OMU|PSU|5MU      CM2  10
U|C|URU|CYT|UR3|OMU|PSU|5MU      N4   5
U|C|URU|CYT|UR3|OMU|PSU|5MU      C5   11
CMU                           C5M  10
U|C|URU|CYT|UR3|OMU|PSU|5MU      C6   11

# Atoms that are sometimes named like mainchain atoms but aren't really
# get caught here
ACE  CA   9
*   CG   8

# Mainchain atoms - invariant by residue/nucleotide type
*   N    4
*   CA   7
*   C    10
*   O    1
*   P   13

# CB - C beta
ALA  CB   9
ILE|THR|VAL  CB   7
*   CB   8

# CG - C gamma
ASN|ASP|ASX|HIS|HIP|HIE|HID|HISL|HISN|HTSL|LEU|PHE|TRP|TYR  CG   10
ARG|GLU|GLN|GLX|MET  CG   8
LEU  CG   7
*   CG   8

# Other amino acid residues listed by residue type
# note the "question mark" matches zero or one occurrences of pattern
GLN  O1   3
GLN  O2   3
ACE  CH3  9
ARG  CD   8
ARG  NE   4
ARG  RE   4
ARG  CZ   10
ARG  NH[12][AB]?  5
ARG  RH[12][AB]?  5
ARG  OD1  1
ASN  ND2  5
ASN  AD1  3
ASN  AD2  3
ASN  OD2  3
ASP  OD[12][AB]?  3
ASP  ED[12][AB]?  3
ASP  ED[12][AB]?  3
ASN  OD1  1
ASX  ND2  5
ASX  AD1  3
ASX  AD2  3
ASX  OD2  3
CYS|MET  LP[12]  13
CY|SXM] SG   13

CYH  SG   12
GLU  OE[12][AB]?  3
GLU  EE[12][AB]?  3
GLU|GLN|GLX  CD   10
GLN  OEL  1
GLN  NE2  5
GLN|GLX  AE[12]  3

# His and relatives
# There are 4 kinds of HIS rings: HIS (no Protons), HID (proton on Delta),
# HIE (proton on epsilon), and HIP (protons on both)
# Protonated nitrogens are numbered 4, else 14
# HIS is treated here as the same as HIE
# # HIS is a deprotonated HIS (the L means liganded)
HIS|HID|HIE|HIP|HISL  CEL|CD2  11
HIS|HIE|HISL  ND1   14
HDI|HIP  RD1   4
HIS|HIE|HIP  NE2   4
HIS|HIE|HIP  RE2   4
HDI|HISL  NE2   14
HIS|HISL  RE2   14
HIS|HID|HIP|HISD  A|DE|[12]  4

ILE  CG1  8
ILE  CG2  9
ILE  CD|CD1  9
LEU  CD1  9
LEU  CD2  9
LYS  CL|GDE]  8
LYS  NZ   6
LYS  KZ   6
MET  SD   13
MET  CE   9

PHE  C|DE|[12]  11
PHE  CZ   11

PRO|CPR  C[GD]  8
CSO  SE   9
CSO  SEC  9
CSO  OD1  3
CSO  OD2  3
SER  OG   2
THR  OC1  2
THR  CG2  9
TRP  CD1  11
TRP  CD2  10
TRP  CE2  10
TRP  NE1  4
TRP  CE3  11
TRP  C22  11
TRP  CG3  11
TRP  CH2  11
TYR  C|DE|[12]  11
TYR  CZ   10
TYR  OH   2
VAL  CG1  9
VAL  CG2  9

# catch common atom names for non-standard residue names
*   CD   8
*   CE   8
#

```

```

# checking these next two with JAT & EDG mp
# (numbering up to 7 is on suggestion of Dave Stout 20 Feb 90 mp)
FS[34] FE[1-7] 21
FS[34] SL1-7] 13
FS3 OXO 1
FFO FF1 21
FFO FE2 21

```

```

FMN C[2478] 10
FMN C[2478] 10
FMN 02 1
FMN N3 14
FMN 04 1
FMN C[459]A 10
FMN NS 4
FMN C[169] 11
FMN C[169] 11
FMN C[78]M 9
FMN NL0 4
FMN C[10] 10
FMN C[12345]\* 8
FMN C[12345]\* 8
FMN O[234]\* 2
FMN O[234]\* 2
FMN OP[-3] 3
ALK[MNR OT1 3
ALK[MNR C01 10
ALK C16 9
MYR C14 9
ALK[MNR C.* 8
ALK[MNR C.* 8
# Catch-all
* SEG 9
* OXT 3
* OT.* 3
* E.* 3
* S.* 13
* C.* 7
# A is for aromatic carbons - GMW/DSG/A10.
* A.* 11
* O.* 1
* N.* 4
* R.* 4
* K.* 6
# PB might be Pb (lead) but its academic... Many PDB files have named P atoms
* P[A-D] 13
# Added even more general phosphorus - GMW/DSG/A10.
* P.* 13
# further catch-all for HETATM entries, e.g. AC1*, A05, PC14, PO10 (MP 199709)
FAD|NAD|AMX|APU .O.* 1
FAD|NAD|AMX|APU .N.* 4
FAD|NAD|AMX|APU .C.* 7
FAD|NAD|AMX|APU .P.* 13
FAD|NAD|AMX|APU .H.* 15

```

Metals:

```

# Heroes of SOD
* CU 20
* ZN 19
# Note 24 and 25 have not been OKed by John & Libby - MP, September 1992
* MN 24
# This free FE is not the same number as heme FE
* FE 25
# These are metals, considered uncharged so use at your own risk
* MG 26
* MN 27
* CO 28
* SE 29
* YB 31
# Unknown soldiers:
# FMN is cofactor Flavin mononucleotide
FMN NI 4

```

F.2 Example Programs

There are seven programs in the Voss Volume Voxelator (³V) package:

- Cavity Extractor: extracts the cavities for a given probe radius
- Channel Extractor: extracts a particular channel from the solvent
- FSV Calculator: calculates the Fractional Solvent Volume
- Solvent Extractor: extracts all of the solvent
- Tunnel Extractor: extracts the ribosomal exit tunnel from the *H. marismortui* structure
- VDW Calculator: calculate the van der Waals (VDW) volume and surface area
- Volume Calculator: calculate the volume and surface area for any probe radius

```

#include <iostream>
#include "utils.h"

extern float XMIN, YMIN, ZMIN;
extern float XMAX, YMAX, ZMAX;
extern int DX, DY, DZ;
extern unsigned int NUMBINS;
extern float MAXPROBE;
extern float GRIDRAD;
extern float WATERRES;
extern float CUTOFF;
extern char XYZRFILE[256];
extern char XYZWFILE[256];

int getcavitiesBothMeth(const float probe, gridpt shellACC[], gridpt shellEXC[], const int natoms, char file[], char edzfile[], char pbfile[]);

int main(int argc, char *argv[])
{
    cerr << endl;
    // *****
    // USER INPUT
    // *****

    // COMPILE INFO:
    CITATION;

    char file[256]; file[0] = '\0';
    char edzfile[256]; edzfile[0] = '\0';
    char pbfile[256]; pbfile[0] = '\0';
    double shell_rad = 10.0;
    double probe_rad = 3.0;
    double trim_rad = 3.0;

    while(argc > 1 && argv[1][0] == '-') {
        if(argv[1][1] == 'r') {
            sprintf(file, "%s", argv[1]);
        } else if(argv[1][1] == 'e') {
            GRIDRAD = atof(argv[1]);
        } else if(argv[1][1] == 's') {
            shell_rad = atof(argv[1]);
        } else if(argv[1][1] == 'p') {
            probe_rad = atof(argv[1]);
        } else if(argv[1][1] == 't') {
            trim_rad = atof(argv[1]);
        } else if(argv[1][1] == 'o') {
            sprintf(edzfile, "%s", argv[1]);
        } else if(argv[1][1] == 'a') {
            sprintf(pbfile, "%s", argv[1]);
        } else if(argv[1][1] == 'h') {
            cerr << endl;
            cerr << "Cavities.exe -e <file> -g <grid spacing> -s <shell radius>" << endl;
            cerr << "Cavities.exe -t <trim_probe_rad> -e <EDZ outfile> -o <PDB outfile>" << endl;
        }
    }

    cerr << "Cavities.exe -- Extracts the cavities for a given probe radius" << endl;
    cerr << endl;
    return 1;
}

--argc; --argc;
+argc; +argc;
}

// INITIALIZATION
// *****
// INITIALIZE GRID
finalGridBins(shell_rad*2);
// HEADER CHECK

```

```

    }  

    cerr << "LAST POINT: " << lastpt << endl;  

    //Pull channels out of inverse excluded map  

    gridpt *chanEXC=NULL;  

    chanEXC = (gridpt*) malloc (NUMBINS);  

    zeroGrid (chanEXC);  

    cerr << "getting Connected Next" << endl;  

    get_Connected_Point (cavEXC, chanEXC, firstpt); //modifies chanEXC  

    get_Connected_Point (cavEXC, chanEXC, lastpt); //modifies chanEXC  

    int chanEXC_voxels = countGrid(chanEXC);  

    //Subtract channels from exclude map leaving cavities  

    sub_Grids(cavEXC, chanEXC); //modifies cavEXC  

    free (chanEXC);  

    int cavEXC_voxels = countGrid(cavEXC);  

    //Write out exclude cavities  

    if(pdffile[0] != '\0') {  

        write_SurfPBC(cavEXC, pdffile);  

    }  

    if(edffile[0] != '\0') {  

        write_HalfED(cavEXC, edffile);  

    }  

    //float surfEXC = surface_area(cavEXC);  

    free (cavEXC);  

    cerr << endl;  

    cerr << "achanACC_voxels = " << achanACC_voxels << endl;  

    cerr << "chanACC_voxels = " << chanACC_voxels << endl;  

    cerr << "cavACC_voxels = " << cavACC_voxels << endl;  

    cerr << "scavACC_voxels = " << scavACC_voxels << endl;  

    cerr << "ecavACC_voxels = " << ecavACC_voxels << endl << endl;  

    cerr << "echanEXC_voxels = " << echanEXC_voxels << endl;  

    cerr << "chanEXC_voxels = " << chanEXC_voxels << endl;  

    cerr << "cavEXC_voxels = " << cavEXC_voxels << endl;  

    cout << probe << "\t";  

    printVolCount(cavACC_voxels);  

    cout << "\t";  

    printVolCount(cavEXC_voxels);  

    cout << "\t";  

    printVolCount(chanEXC_voxels);  

    cout << "\t";  

    printVolCount(scavACC_voxels);  

    cout << endl;  

    // float perACC = 100*float(tunnACC_voxels) / float(chanACC_voxels);  

    // float perEXC = 100*float(tunnEXC_voxels) / float(chanEXC_voxels);  

    return cavACC_voxels+ecavACC_voxels;
}

//Create access map  

gridpt *access2=NULL;  

access2 = (gridpt*) malloc (NUMBINS);  

fill_AccessGridFromFile(natoms, probe, file, access2);

//Create exclude map  

gridpt *exclude=NULL;  

exclude = (gridpt*) malloc (NUMBINS);  

trun_ExcludeGrid(probe, access2, exclude);  

free (access2);

int echanEXC_voxels = countGrid(cavEXC);

//Get first point  

stop = 1; firstpt = 0;  

for(unsigned int pt=0; pt<NUMBINS && stop; pt++) {  

    if(cavEXC[pt]) { stop = 0; firstpt = pt; }
}  

cerr << "FIRST POINT: " << firstpt << endl;  

//LAST POINT  

stop = 1; lastpt = NUMBINS-1;  

for(unsigned int pt=NUMBINS-1; pt>0 && stop; pt--) {  

    if(cavEXC[pt]) { stop = 0; lastpt = pt; }
}

```

```

    }
}

//INITIALIZE GRID
finalGriddims(BIGPROBE);

//HEADER CHECK
cerr << "Probe Radius: " << BIGPROBE << endl;
cerr << "Grid Spacing: " << GRID << endl;
cerr << "Resolution: " << int(1000.0/float(GRIDVOL))/1000.0 << " voxels per A^3" << endl;
cerr << "Resolution: " << int(11494.0/float(GRIDVOL))/1000.0 << " voxels per water
molecule" << endl;
cerr << "Input file: " << file << endl;
cerr << "Input file: " << file << endl;

//FIRST PASS, MINMAX
int numatoms = read_NumAtoms(file);

//CHECK LIMITS & SIZE
assignlimits();

COMPILE_INFO;
CITATION;

// *****
// INITIALIZATION
// *****

//HEADER INFO
char edffile[256]; file[0] = '\0';
char pdffile[256]; edffile[0] = '\0';
char pdffile[256]; pdffile[0] = '\0';
double SMPROBE=.9;
double TRIMPROBE=.4;
double x=1000,y=1000,z=1000;
double x1=1000,y1=1000,z1=1000;

while(argc > 1 && argv[1][0] == '-' ) {
    if(argv[1][1] == '-' ) {
        sprintf(argv[1], "%s", argv[1]);
    } else if(argv[1][1] == 't' ) {
        BIGPROBE = atof(argv[2][0]);
    } else if(argv[1][1] == 's' ) {
        SMPROBE = atof(argv[2][0]);
    } else if(argv[1][1] == 'f' ) {
        TRIMPROBE = atof(argv[2][0]);
    } else if(argv[1][1] == 'x' ) {
        x = atof(argv[2][0]);
    } else if(argv[1][1] == 'y' ) {
        y = atof(argv[2][0]);
    } else if(argv[1][1] == 'z' ) {
        z = atof(argv[2][0]);
    } else if(argv[1][1] == 'b' ) {
        sprintf(file,&argv[2][0]);
    } else if(argv[1][1] == 'o' ) {
        sprintf(pdffile,&argv[2][0]);
    } else if(argv[1][1] == 'e' ) {
        sprintf(edffile,&argv[2][0]);
    } else if(argv[1][1] == 'g' ) {
        GRID = atof(&argv[2][0]);
    } else if(argv[1][1] == 'h' ) {
        cerr << "./Channel.exe -i <file> -b <big_probe> -s <small_probe>" << endl
             << "\t-t <trim probe> -x <x-coord> -y <y-coord> -z <z-coord>" << endl
             << "\t-e <EDB outfile> -o <PDB outfile> -g <gridspace>" << endl;
        cerr << "Channel.exe -- Extracts a particular channel from the solvent" << endl;
        cerr << endl;
    }
}

--arg; --arg;
++arg; ++arg;
}

```

```
/*
 * ***** SELECT PARTICULAR CHANNEL *****
 */
gridpt *channelACC;
channelACC = (gridpt*) malloc (NUMBINS);
if (channelACC==NULL) { cerr << "GRID IS NULL" << endl; return 1; }

main channel
{
    get_Connected(solventACC,channelACC, x, y, z);

    free (solventACC);

    /*
     * ***** GETTING CONTACT CHANNEL *****
     */
    gridpt *channeLEXC;
    channelLEXC = (gridpt*) malloc (NUMBINS);
    if (channeLEXC==NULL) { cerr << "GRID IS NULL" << endl; return 1; }

    int channelACCvol = copyGrid(channelACC,channeLEXC);
    cerr << "Accessible Channel1 Volume " ;
    grow_IncludeGrid(SNPROBE,channelACC,channeLEXC);
    free (channelACC);

    //limit growth to inside trimgrid
    intersect_Grids(channeLEXC,trimgrid); //modifies channeLEXC
    free (trimgrid);

    /*
     * ***** OUTPUT RESULTS *****
     */
    cout << BIGPROBE << "\t" << SNPROBE << "\t" << GRID << "\t" << flush;
    int channelLEXCvol = countGrid(channeLEXC);
    printVolOut(channeLEXCvol);
    long double surf = surface.area (channeLEXC);
    cout << "\t" << surf << "\t" << flush;
    printVolOut(channelACCvol);
    cout << "\t" << file << endl;
    if (pdffile[0] != '\0') {
        write_SurfPDF(channeLEXC,pdffile);
    }
    if (ezdfile[0] != '\0') {
        write_HalfED(channeLEXC,ezdfile);
    }

    //RELEASE TEMPGRID
    free (channeLEXC);
    cerr << endl;
}

free (trimgrid);
cerr << endl << "Program Completed Sucessfully" << endl << endl;
return 0;
};
```

```

#include <iostream>
#include "utils.h"

extern float XMIN, YMIN, ZMIN;
extern float XMAX, YMAX, ZMAX;
extern int DX, DY, DZ;
extern unsigned int NUMBINS;
extern float MAXPROBE;
extern float GRID;
extern float GRIDVOL;
extern float WATER_RES;
extern float CUTOFF;
extern char XYZRFILE[256];
extern char XYZWFILE[256];

int main(int argc, char *argv[]) {
    cerr << endl;
    COMPILE_INFO;
    CITATION;
    // *****
    // INITIALIZATION
    // *****

    //HEADER INFO
    char edffile[256]; edffile[0] = '\0';
    char pbffile[256]; pbffile[0] = '\0';
    double SMPROBE=.10;
    double TRIMPROBE=.1.5; //HEADER INFO

    while(argc > 1 && argv[1][0] == '-' ) {
        if(argv[1][1] == 'i') {
            sprintf(file,kargv[2][0]);
        } else if(argv[1][1] == 's') {
            SMPROBE = atof(&argv[2][0]);
        } else if(argv[1][1] == 'b') {
            BIGROBE = atof(kargv[2][0]);
        } else if(argv[1][1] == 't') {
            TRIMPROBE = atof(&argv[2][0]);
        } else if(argv[1][1] == 'g') {
            GRID = atof(&argv[2][0]);
        } else if(argv[1][1] == 'o') {
            sprintf(pbffile,kargv[2][0]);
        } else if(argv[1][1] == 'e') {
            sprintf(ezadfile,kargv[2][0]);
        } else if(argv[1][1] == 'h') {
            cerr << "./fsvac.exe -1 <file> -b <big_probe> -s <small_probe>" << endl;
            cerr << "\t\t<trim_probe> -e <PDB outfile> -g <gridSpace>" << endl;
            cerr << "FsvCalc.exe -- Calculates the Fractional Solvent Volume" << endl;
        }
        return 1;
    }
    --argc; --argv;
    ++argv; ++argv;
}

//INITIALIZE GRID
finalGridDims(BIGPROBE);

//HEADER CHECK
cerr << "Grid Spacing: " << GRID << endl;
cerr << "Resolution: " << int(1000.0/float(GRIDVOL))/1000.0 << " voxels per A^3" << endl;
cerr << "Resolution: " << int(11494.0/float(GRIDVOL))/1000.0 << " voxels per water

```

```

molecule" << endl;
    cerr << "Complexity: " << int(8000000/float(GRIDVOL))/1000.0 << endl;
    cerr << "Input file: " << file << endl;

//FIRST PASS, MINMAX
int numatoms = read_NumAtoms(file);

//CHECK LIMITS & SIZE
assignLimits();

// *****
// STARTING FIRST FILE
// *****

//GET SHELL
gridpt *shell;
shell = (gridpt*) malloc (NUMBINS);
if (shell==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }

zeroGrid(shell);
int shellvol = getFile(numatoms,BIGPROBE,file,shell);

//INT NEW chanACC GRID
cerr << "Trimming Radius: " << TRIMPROBE << endl;
gridpt *smShell;
smShell = (gridpt*) malloc (NUMBINS);
if (smShell==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }

//COPY AND TRUNCATE (IF NECESSARY)
copyGrid(shell,smShell);
if(TRIMPROBE > 0) {
    trim_ExcludeGrid(TRIMPROBE,shell,smShell);
}

free (shell);

//COPY SMHELL INTO CHANACC
gridpt *chanACC;
chanACC = (gridpt*) malloc (NUMBINS);
if (chanACC==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }
int smshellvol = copyGrid(smShell,chanACC);

//SUBTRACT WATER.ACC FROM SHELL TO GET ACC CHANNELS
//GET ACCESS WATER.VOLUME
gridpt *waterACC;
waterACC = (gridpt*) malloc (NUMBINS);
if (waterACC==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }
zeroGrid(waterACC);
fill_AccessGrid(frontFile(numatoms,SMPROBE,file,waterACC));
sub_Grids(chanACC,waterACC);
free (waterACC);

//INT NEW chanEXC GRID
gridpt *chanEXC;
chanEXC = (gridpt*) malloc (NUMBINS);
if (chanEXC==NULL) { cerr << "GRID IS NULL" << endl; exit (1); }

//GROW EXCLUDED SURFACE FROM ACCESSIBLE
copyGrid(chanACC,chanEXC);
grow_ExcludedGrid(SMPROBE,chanACC,chanEXC);
free (chanACC);

//INTERSECT
intersectGrids(chanEXC,smShell); //modifies chanACC
free (smShell);

//OUTPUT
int chanvol = countGrid(chanEXC);
cout << "BIG PROBE (R = " << BIGROBE << ") VOLUME: " ;

```

```
printVolCout(shellvol);
cout << endl << "TRIMMED (BY R = " << TRIMPROBE << ") VOLUME: ";
printVolCout(smishelevol);
cout << endl << "SOLVENT (R = " << SMPROBE << ") VOLUME: ";
printVolCout(chanvol);
cout << endl << "FRACTIONAL SOLVENT VOLUME: " << double(100.0*chanvol) / double(shellvol)
<< "%";
<< endl << endl;

if(pdbfile[0] != '\0') {
    write.SurfPDB(chanEXC,pdbfile);
}
if(ezdfile[0] != '\0') {
    write.HalfZD(chanEXC,ezdfile);
}
free (chanEXC);

cerr << endl << "Program Completed Sucessfully" << endl << endl;
return 0;
};
```

```

#include <iostream>
#include "utils.h"

extern float XMIN, YMIN, ZMIN;
extern float XMAX, YMAX, ZMAX;
extern int DX, DY, DZ;

extern int dxy, dxz;
extern unsigned int NUMBINS;
extern float MAXPROBE;
extern float GRID;
extern float GRIDVOL;
extern float WATER_RES;
extern float CUTOFF;
extern char XYZRFILE[256];
extern char TACCVOLFILE[256];

int main(int argc, char *argv[]) {
    COMPILE_INFO;
    CITATION;
    // *****
    // INITIALIZATION
    // *****

    //HEADER INFO
    char file[256]; file[0] = '\0';
    char edffile[256]; edffile[0] = '\0';
    char pbffile[256]; pbffile[0] = '\0';
    double SMPROBE=.15; //HEADER INFO

    while(argc > 1 && argv[1][0] == '-') {
        if(argv[1][1] == 'i') {
            sprintf(file,kargv[2][0]);
        } else if(argv[1][1] == 's') {
            SMPROBE = atof(&argv[2][0]);
        } else if(argv[1][1] == 'b') {
            BIGPROBE = atof(kargv[2][0]);
        } else if(argv[1][1] == 't') {
            TRIMPROBE = atof(&argv[2][0]);
        } else if(argv[1][1] == 'e') {
            sprintf(edffile,kargv[2][0]);
        } else if(argv[1][1] == 'o') {
            sprintf(pbffile,kargv[2][0]);
        } else if(argv[1][1] == 'g') {
            GRID = atof(&argv[2][0]);
        } else if(argv[1][1] == 'h') {
            cerr << "./Solvent.exe -1 <file> -s <sm.probe_rads> -b <big.probe_rads>" << endl
            << "\t-t <trim.probe_rads> -g <grid spacing> -e <ZD outfile> -o <PDB outfile>" << endl;
        }
        cerr << "Solvent.exe -- Extracts the all of the solvent" << endl;
        return 1;
    }
    --argc; --argc;
    ++argv; ++argv;
}

//INITIALIZE GRID
finalGridBins(BIGPROBE);

//HEADER CHECK
if(SMPROBE > BIGPROBE) { cerr << "ERROR: SMPROBE > BIGPROBE" << endl; return 1; }
err << "Small Probe Radius: " << SMPROBE << endl;
err << "int(1000.0/float(GRIDVOL))/1000.0 << " voxels per A^3" << endl;
err << "int(1000.0/float(GRIDVOL))/1000.0 << " voxels per water molecule" << endl;

//FIRST PASS, MINMAX
int numatoms = read_NumAtoms(file);
//CHECK LIMITS & SIZE
assignLimits();
// *****
// STARTING LARGE PROBE
// *****
gridpt *biggrid; malloc (NUMBINS);
biggrid = (gridpt*) malloc (NUMBINS);
if(biggrid==NULL) { cerr << "GRID IS NULL" << endl; return 1; }

// *****
// EXCLUDING GRID
int bigvox;
if(BIGPROBE > 0.0) {
    bigvox = get_ExcludeGridFromFile(numatoms,BIGPROBE,file,biggrid);
} else {
    cerr << "BIGPROBE <= 0" << endl;
    return 1;
}

// *****
// TRIM LARGE PROBE SURFACE
// *****
// *****
// *****
gridpt *trimpgrid;
trimpgrid = (gridpt*) malloc (NUMBINS);
if (trimpgrid==NULL) { cerr << "GRID IS NULL" << endl; return 1; }

copyGrid(biggrid,trimpgrid);
if(TRIMPROBE > 0) {
    trun.ExcludeGrid(TRIMPROBE,biggrid,trimpgrid);
    free (biggrid);
    cout << "bg_pdb(tsm_prb\trgrid\textvol\tsurf\tacccv\ntfile" << endl;
}

// *****
// *****
// *****
gridpt *smgrid;
smgrid = (gridpt*) malloc (NUMBINS);
if (smgrid==NULL) { cerr << "GRID IS NULL" << endl; return 1; }

zeroGrid(smgrid);
int smvox;
smvox = fill_AccessGrid_fromFile(numatoms,SMPROBE,file,smgrid);

// *****
// GETTING ACCESSIBLE CHANNELS
// *****
// *****
gridpt *solventACC;
solventACC = (gridpt*) malloc (NUMBINS);
if (solventACC==NULL) { cerr << "GRID IS NULL" << endl; return 1; }

copyGrid(trimpgrid,solventACC); //copy trimpgrid into solventACC
sub.Grids(solventACC,smgrid); //modify solventACC
free (smgrid);

// *****
// GETTING CONTACT CHANNEL
// *****
gridpt *solventEXC;
solventEXC = (gridpt*) malloc (NUMBINS);
if (solventEXC==NULL) { cerr << "GRID IS NULL" << endl; return 1; }

copyGrid(trimpgrid,solventEXC); //copy trimpgrid into solventEXC
sub.Grids(solventEXC,smgrid); //modify solventEXC

```

```
if (solventEXC==NULL) { cerr << "GRID IS NULL" << endl; return 1; }

int solventACCvol = copyGrid(solventACC,solventEXC);
cout << "Accessible Channel1 Volume ";
printVol(solventACCvol);
grow.ExcludeGrid(SMPROBE,solventACC,solventEXC);
free (solventACC);

//limit growth to inside tragrid
intersect_Grids(solventEXC,tragrid); //modifies solventEXC
free (tragrid);

// **** OUTPUT RESULTS ****
// **** OUTPUT RESULTS ****
// **** OUTPUT RESULTS ****
cout << BIGPROBE << "\t" << SMPROBE << "\t" << GRID << "\t" << flush;
int solventEXCvol = countGrid(solventEXC);
printVolOut (solventEXCvol);
long double surf = surface.area (solventEXC);
cout << "\t" << surf << "\n" << flush;
printVolOut (solventACCvol);
cout << "\t" << file << endl;
if (pdbfile[0] != '\0') {
    write_SurfPDB(solventEXC,pdbfile);
}
if (ezdfile[0] != '\0') {
    write_HalfED(solventEXC,ezdfile);
}

free (solventEXC);
cerr << endl << "Program Completed Sucessfully" << endl << endl;
return 0;
};
```

```

#include <iostream>
#include "utils.h"

extern float XMIN, YMIN, ZMIN;
extern float XMAX, YMAX, ZMAX;
extern int DX, DY, DZ;
extern int dX, dY, dZ;
extern unsigned int NUMBINS;
extern float MAXPROBE;
extern float GRID;
extern float GRIDVOL;
extern float WATER_RES;
extern float CUTOFF;
extern char XYZFILE[256];
extern char XYZREFILE[256];

void printTun(const float probe,
    const float surfEXC, const int tunnEXC_voxels, const int chanEXC_voxels,
    const float surfACC, const int tunnACC_voxels, const int chanACC_voxels,
    char file[]);

void defineTunnel(gridpt tunnel[], gridpt channels[]);

int main(int argc, char *argv[]) {
    cerr << endl;

    COMPILE_INFO;
    CITATION;
    // **** INITIALIZATION :: REQUIRED
    // ****
}

//FIRST PASS, MINMAX
int numatoms = read_NumAtoms(file);
//CHECK LIMITS & SIZE
assignLimits();

//HEADER CHECK
cerr << "Grid Spacing: " << GRID << endl;
cerr << "Resolution: " << int(1000.0/Float(GRIDVOL))/1000.0 << " voxels per A^3" << endl;
cerr << "Resolution: " << int(11494.0/Float(GRIDVOL))/1000.0 << " voxels per water molecule" << endl;
cerr << "Input file: " << file << endl;

// **** BUSINESS PART
// ****
//Compute Shell
gridpt *shellACC=NULL;
shellACC = (gridpt*) malloc (NUMBINS);
fill_AccessGridFromFile(numatoms,shell_rad,file,shellACC);
fill_cavities(shellACC);

gridpt *shellEXC=NULL;
shellEXC = (gridpt*) malloc (NUMBINS);
trun.ExcludeGrid(shell_rad,shellACC,shellEXC);
free (shellACC);

//Trim Shell
if(trim_prob > 0.0) {
    gridpt *trimEXC;
    trimEXC = (gridpt*) malloc (NUMBINS);
    copyGrid(shellEXC,trimEXC);
    trim.ExcludeGrid(trim_prob,shellEXC,trimEXC); // TRIMMING PART
    zeroGrid(shellEXC);
    copyGrid(trimEXC,shellEXC);
    free (trimEXC);
}

//Get Shell Volume
int shell_vol = countGrid(shellEXC);
printVol(shell_vol); cerr << endl;

//Get Access Volume for "probe"
gridpt *access;
access = (gridpt*) malloc (NUMBINS);
fill_AccessGridFromFile(numatoms,tunnel_prob,file,access);

//Get Channels for "probe"
gridpt *chanACC;
chanACC = (gridpt*) malloc (NUMBINS);
copyGrid(shellEXC,chanACC);
sub_Grids(chanACC,access);
free (access);
int chanACC_voxels = countGrid(chanACC);
printVol(chanACC_voxels); cerr << endl;

//Extract Tunnel
gridpt *tunnACC;
tunnACC = (gridpt*) malloc (NUMBINS);
defineTunnel(tunnACC,chanACC);
free (chanACC);
int tunnACC_voxels = countGrid(tunnACC);
float surfFACC = surface_area(tunnACC);

//Grow Tunnel
gridpt *tunnEXC;

```

```

cout << probe << "\n";
printVolCout(tunnEXC_voxels);
printVolCout(chanACC_voxels);
cout << perEXC << "\n";
cout << surfEXC << "\n";
cout << GRID << endl;
//ACCESS
printVolCout(tunnACC_voxels);
printVolCout(chanACC_voxels);
cout << perACC << "\n";
cout << surfACC << "\n";
cout << GRID << endl;
return;
};

//Get EXC Props
int tunnEXC_voxels = countGrid(tunnEXC);
float surfEXC = surface_area(tunnEXC);

//Intersect Grids(tunnEXC,shellEXC); //modifies tunnEXC
intersect_Grids(tunnEXC,shellEXC);

//Get PDB
if(pdofile[0] != '\0') {
    write_SurpPDB(tunnEXC,pdofile);
}
if(edzfile[0] != '\0') {
    write_HalfEDZ(tunnEXC,edzfile);
}

free (tunnEXC);

cerr << endl << "Program Completed Successfully" << endl << endl;
return 0;
};

void defineTunnel(gridpt tunnel[], gridpt channels[])
{
    //NEW IDEAL TUNNEL POINTS
    get_Connected(chanACC,tunnACC,77.2,116.0,109.2); //tRNA cleft
    get_Connected(channels,tunnel,74.130,0.83,6); //highest tunnel pt
    get_Connected(channels,tunnel,68.3,132.2,85.6); //largest area
    get_Connected(channels,tunnel,53.6,144.8,69.6); //below main
    get_Connected(channels,tunnel,49.9,151.8,67.3); //2nd largest & low
    get_Connected(channels,tunnel,38.4,160.4,63.6); //low blob point
    get_Connected(channels,tunnel,35.6,163.6,61.6); //lowest pt
    //OLD POINTS : CAN'T HURT
    get_Connected(channels,tunnel,53.6,141.3,66.4);
    get_Connected(channels,tunnel,71.5,120.4,97.3);
    get_Connected(channels,tunnel,71.5,122.0,98.1);
    get_Connected(channels,tunnel,70.3,131.2,81.9);
    get_Connected(channels,tunnel,55.7,140.2,73.8);
    get_Connected(channels,tunnel,44.6,153.2,68.7);

    //get_Connected(channels,tunnel,0,0,0,0,0);
}

void printFun(const float probe,
const float surfEXC, const int tunnEXC_voxels,
const float surfACC, const int tunnACC_voxels, const int chanACC_voxels,
char file[])
{
    float perAC = 100*float(tunnACC_voxels) / (float(chanACC_voxels) + 0.01);
    float perEXC = 100*float(tunnEXC_voxels) / (float(chanEXC_voxels) + 0.01);

```

```

#include <iostream>
#include "utils.h"

extern float XMIN, YMIN, ZMIN;
extern float XMAX, YMAX, ZMAX;
extern int DX, DY, DZ;
extern unsigned int NUMBINS;
extern float MAXPROBE;
extern float GRID;
extern float GRIDVOL;
extern float WATER_RES;
extern float CUTOFF;
extern char XYZREFILE[2436];
extern char PDBREFILE[2436];

int main(int argc, char *argv[]) {
    cerr << endl;

    COMPILE_INFO;
    CITATION;
    // *****
    // INITIALIZATION
    // *****

    //HEADER INFO
    char file[256]; file[0] = '\0';
    char edzfile[256]; edzfile[0] = '\0';
    char pdbfile[256]; pdbfile[0] = '\0';
    double PROBE=0.0;

    while(argc > 1 && argv[1][0] == '-') {
        if(argv[1][1] == '-') {
            sprintf(file,kargv[2][0]);
        } else if(argv[1][1] == 'o') {
            sprintf(pdbfile,kargv[2][0]);
        } else if(argv[1][1] == 'e') {
            sprintf(edzfile,kargv[2][0]);
        } else if(argv[1][1] == 'g') {
            GRID = atof(&argv[2][0]);
        } else if(argv[1][1] == 'h') {
            cerr << "VDW exe -i <file> -g <gridspacing>" << endl;
            cerr << "\t-e <EDZ outfile> -o <PDB outfile>" << endl;
            cerr << "Calculate the VDW volume and surface area" << endl;
            cerr << endl;
            return 1;
        }
        --argc; --argc;
        +argv; +argv;
    }

    //INITIALIZE GRID
    finalGridDims(PROBE);

    //HEADER CHECK
    cerr << "Probe Radius: " << PROBE << endl;
    cerr << "Grid Spacing: " << GRID << endl;
    cerr << "Resolution: " << int(1000.0/float(GRIDVOL))/1000.0 << " voxels per A^3" << endl;
    cerr << "Resolution: " << int(11494.0/float(GRIDVOL))/1000.0 << " voxels per water molecule" << endl;
    cerr << "Input File: " << file << endl;
    //FIRST PASS, MINMAX
    int numatoms = read_NumAtoms(file);
    //CHECK LIMITS & SIZE
}

```

```

#include <iostream>
#include "utils.h"

extern float XMIN, YMIN, ZMIN;
extern float XMAX, YMAX, ZMAX;
extern int DX, DY, DZ;
extern unsigned int NUMBINS;

extern float MAXPROBE;
extern float GRIDVOL;
extern float WATER_RES;
extern float CUTOFF;
extern char XYZREFILE[256];
extern char EXCGRIDFILE[256];

int main(int argc, char *argv[]) {
    cerr << endl;
    COMPILE_INFO;
    CITATION;
    // *****
    // INITIALIZATION
    // *****

    //HEADER INFO
    char edzfile[256]; file[0] = '\0';
    char pdbfile[256]; edzfile[0] = '\0';
    char pdbfile[256]; pdbfile[0] = '\0';
    double PROBE=10.0;

    while(argc > 1 && argv[1][0] == '-') {
        if(argv[1][1] == 't') {
            sprintf(file,'karyv[2][0]');
        } else if(argv[1][1] == 'p') {
            PROBE = atof(argv[1][0]);
        } else if(argv[1][1] == 'o') {
            sprintf(pdbfile,'karyv[1][0]');
        } else if(argv[1][1] == 'e') {
            sprintf(edzfile,'karyv[2][0]');
        } else if(argv[1][1] == 'g') {
            GRID = atof(&argv[1][0]);
        } else if(argv[1][1] == 'h') {
            cerr << "\nVolume.exe -> gridsspacing -> probe.radius" << endl;
            << "\n-e <EDZ outfile" -> PDB outfile" << endl;
            cerr << "Volume.exe -- Calculate the volume and surface area for any probe radius" <<
endl;
            cerr << endl;
            return 1;
        }
        --argc; --argv;
        +argv; +argv;
    }

    //INITIALIZE GRID
    finalGridDims(PROBE);

    //HEADER CHECK
    cerr << "Probe Radius: " << PROBE << endl;
    cerr << "Grid Spacing: " << GRID << endl;
    cerr << "Resolution: " << int(1000.0/float(GRIDVOL))/1000.0 << " voxels per A^3" <<
endl;
    cerr << "Resolution: " << int(11494.0/float(GRIDVOL))/1000.0 << " voxels per water
molecule" << endl;
    cerr << "Input file: " << file << endl;
}

//FIRST PASS, MINMAX

```

Appendix G

GNU Free Documentation License

Version 1.2, November 2002
Copyright © 2000,2001,2002 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

G.1 APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTe $\mathrm{\acute{x}}$ input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

G.2 VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

G.3 COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

G.4 MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

G.5 COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

G.6 COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

G.7 AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

G.8 TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

G.9 TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

G.10 FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.