Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação

Uma Proposta para a Predição Computacional da Estrutura Terciária de Polipeptídeos

Marcos Borba Cardoso

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Prof. Dr. Osmar Norberto de Souza

Porto Alegre
2007

Dados Internacionais de Catalogação na Publicação (CIP)

C268p Cardoso, Marcos Borba
Uma proposta para a predição computacional da estrutura terciária de polipeptídeos / Marcos Borba Cardoso. – Porto Alegre, 2007.
63 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS
Orientador: Prof. Dr. Osmar Norberto de Souza

1. Informática. 2. Bioinformática. 3. Estrutura de Dados. 4. Algoritmo Recursivo. 5. Proteínas. I. Título.

CDD 005.73

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "***Uma Proposta para a Predição Computacional da Estrutura Terciária de Polipeptídeos***", apresentada por Marcos Borba Cardoso, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Computação Cientifica, aprovada em 12/07/2007 pela Comissão Examinadora:

Prof. Dr. Osmar Norberto de Souza – Orientador — PPGCC/PUCRS

Prof. Dr. Dalcidio Moraes Claudio — PPGCC/PUCRS

Prof. Dr. Carlos Augusto Prolo — FACIN/PUCRS

Homologada em 16/10/07, conforme Ata No. 22 pela Comissão Coordenadora.

Prof. Dr. Fernando Luís Dotti
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P. 32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@inf.pucrs.br
www.pucrs.br/facin/pos

Agradecimentos

Ao professor, orientador e amigo Osmar Norberto de Souza pelo incentivo, apoio, disponibilidade e dedicação durante o mestrado.

Aos amigos e pesquisadores pelas valiosas opiniões e paciência durante a elaboração deste trabalho. Em especial ao avaliador do Seminário de Andamento, Prof. Dalcidio Moraes Claudio.

Às colegas do Laboratório de Bioinformática, Modelagem e Simulação de Biossistemas pelo auxílio no entendimento dos conceitos de bioinformática estrutural.

À minha família pelo exemplo de honestidade, força, determinação, profissionalismo, respeito e apoio incondicional.

À DELL pelos servidores disponibilizados para que fossem utilizados na realização dos experimentos executados durante a pesquisa.

Ao Programa de Pós-Graduação em Ciência da Computação da PUC-RS pela bolsa concedida.

Resumo

Nos últimos anos, um dos grandes desafios da Ciência da Computação perante a Bioinformática é o desenvolvimento de algoritmos, os quais, em um tempo hábil, consigam gerar as estruturas terciárias de proteínas a partir da seqüência linear de seus aminoácidos. Embora existam alguns métodos que consigam gerar estruturas quando se possui outra proteína com um alto grau de similaridade, quando não se possui o mesmo, os métodos até então desenvolvidos não consigam realizar esta predição de forma não onerosa computacionalmente. Este trabalho apresenta um algoritmo recursivo capaz de predizer a topologia de polipeptídeos, utilizando apenas os ângulos da cadeia principal de proteínas com estruturas tridimensionais (3D) já conhecidas. O mesmo se mostra eficaz quando aplicado à mini-proteína Trp-Cage (código PDB 1L2Y) que possui apenas 20 aminoácidos, tendo uma estrutura predita de RMSD igual 3,7 Å; no entanto, para uma proteína de 34 aminoácidos – Mini-Proteína Estabilizada por Pontes Dissulfeto (código PDB 1ZDD) – o mesmo se mostra ineficiente, gerando a melhor proteína com o RMSD igual a 7,2 Å, devido ao fato de não ter sido percorrido todo o espaço conformacional esperado para a mesma. Os resultados e as suas conseqüências são discutidos no trabalho.

Palavras-chave: Bioinformática, Proteína, Predição de Estrutura 3D de Proteínas, Algoritmo Recursivo.

Abstract

In these last years one of the greatest challenges of the Computer Science in Bioinformatics is to develop algorithms, which, in a skillful time generate the tertiary protein structures from the linear sequence of its amino acids. Although there are methods to predict structures for target sequences when a similar protein of known structure (template) is available, this is not true when similarity can not be detected by sequence comparison alone. In the latter case, the methods are very computationally demanding. This work presents a recursive algorithm able to predict the topology of polypeptides of unknown structure using only the polypeptide main-chain torsion angles obtained from PDB templates. The algorithm revealed itself efficient when applied to the mini protein Trp-Cage (PDB ID: 1L2Y) composed of 20 amino acids, predicting its structure with a RMSD of 3,7 Å with respect to the experimental structure. However, for a protein of 34 amino acids – the Disulfide-Stabilized Mini Protein (PDB ID: 1ZDD) – the algorithm was not so efficient, generating the best polypeptide model with a RMSD of 7,2 Å with respect to the experimental structure. Due to the large increase in the possible conformations for the latter (20 to 34 amino acids), its conformational space was not spanned as was the conformational space of 1L2Y. These results and their consequences are discussed in the work.

Keywords: Bioinformatics, Protein, Predicting Protein 3D Structure, Recursive Algorithm.

Sumário

Lista de Figuras

Lista de Símbolos e Abreviaturas

3D	Tridimensional
AMBER	*Assisted Model Building with Energy Refinement*
BLAST	*Basic Local Alignment Search Tool*
BLOSUM	*BLocks SUbstitution Matrix*
CASP	*Critical Assessment of Techniques for Protein Structure Prediction*
DM	Dinâmica Molecular
DNA	*Deoxyribo Nucleic Acid*
DOM	*Document Object Model*
HSP	*High Scored Fragments*
NCBI	*National Center for Biotechnology Information*
NMR	*Nuclear Magnetic Resonance*
PAM	*Percent Accepted Mutation*
PDB	*Protein Data Bank*
RMSD	*Root Mean Square Deviation*
XML	*eXtensible Markup Language*

Capítulo 1. Introdução

Devido à enorme quantidade de seqüências gênicas produzidas anualmente, criou-se a necessidade de bancos de dados que facilitassem o acesso a essas seqüências pela comunidade científica. Com esse intuito, vários países elaboraram bancos de dados que disponibilizam, gratuitamente, seqüências e, também, ferramentas que auxiliam em suas análises [1].

Na segunda metade da década de 90, com o surgimento dos seqüenciadores automáticos de DNA, houve uma explosão na quantidade de seqüências – ver Figura 1 – que ao serem armazenadas, exigem recursos computacionais cada vez mais eficientes. Além do armazenamento ocorria, paralelamente, a necessidade de análise destes dados, o que tornava indispensável a utilização de plataformas computacionais eficientes para a interpretação dos resultados obtidos. Desta necessidade nasceu a Bioinformática, uma nova ciência que une diversas linhas de conhecimento: a engenharia de *software*, a matemática, a estatística, a ciência da computação e a biologia molecular [2].

O banco de dados mais conhecido envolvendo seqüências de nucleotídeos e aminoácidos é o GenBank [1], construído e administrado pelo *National Center for Biotechnology Information* (NCBI) dos Estados Unidos.

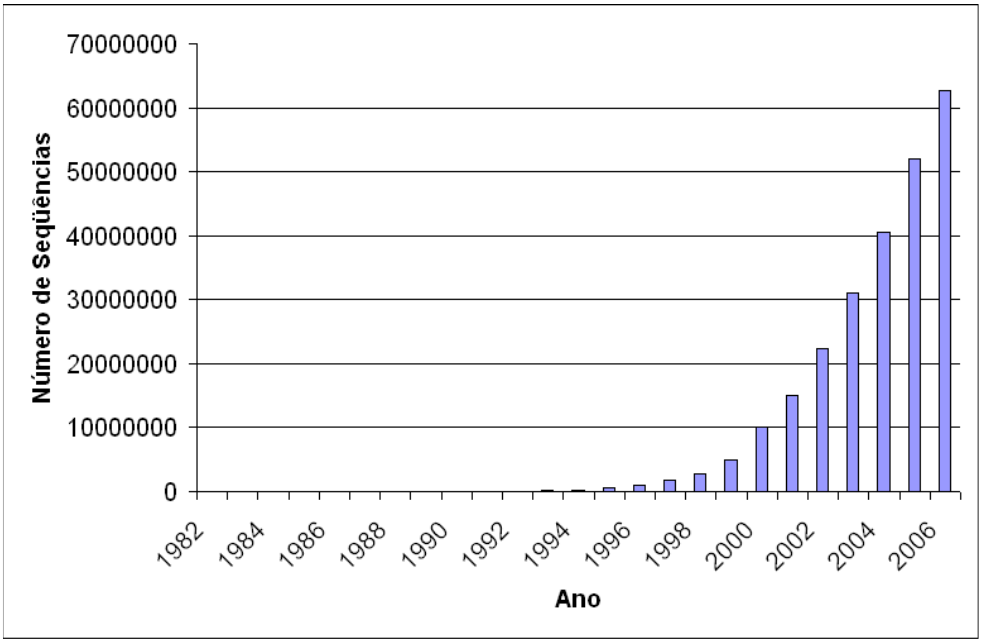


Figura 1: Evolução do número de seqüências de proteínas no GenBank. O seu aumento acentuado se deu a partir da segunda metade da década de 90, período conhecido como "explosão de dados biológicos". Fonte: ftp://ftp.ncbi.nih.gov/genbank/gbrel.txt, última atualização novembro de 2006.

Ao contrário do GenBank [1], o *Protein Data Bank* (PDB) [4] é responsável por coletar, armazenar e distribuir estruturas tridimensionais de proteínas e os conjuntos de dados subjacentes, que informam como as estruturas moleculares foram obtidas [3]. O PDB foi estabelecido no *Brookhaven National Laboratories* em 1971, como um repositório para estruturas de macromoléculas biológicas determinadas por vários métodos experimentais, principalmente cristalografia por difração de raios X e ressonância magnética nuclear (NMR) [4]. Apesar da primeira estrutura protéica ter sido determinada décadas antes da primeira seqüência de DNA, o banco de dados de estrutura protéica cresce mais lentamente do que o banco de dados de seqüências [3].

Utilizando estruturas não redundantes depositadas no PDB e as comparando com as seqüências completas de genomas, estima-se que existam apenas 30% de "moldes" para as proteínas destes genomas [5]. A discrepância entre o número de estruturas resolvidas e o número de novos motivos estruturais depositados no PDB pode ser visualizada, comparando-se a Figura 3 (motivos estruturais) e a Figura 2 (estruturas).

Tendo em vista esta problemática, surge uma nova tecnologia de estudo e caracterização de estruturas protéicas que seja capaz de inferir (com pequena margem de erro) a conformação tridimensional (3D) nativa de uma proteína, tendo como base apenas sua seqüência de aminoácidos, pois esta é a única informação disponível; e que possa, ainda, permitir a descoberta de novas formas de enovelamento ou dobramento (motivos estruturais). A técnica que permite esta abordagem é a metodologia de predição *ab initio*, que utiliza a seqüência linear de aminoácidos como ponto de partida para a construção do modelo 3D de uma proteína.

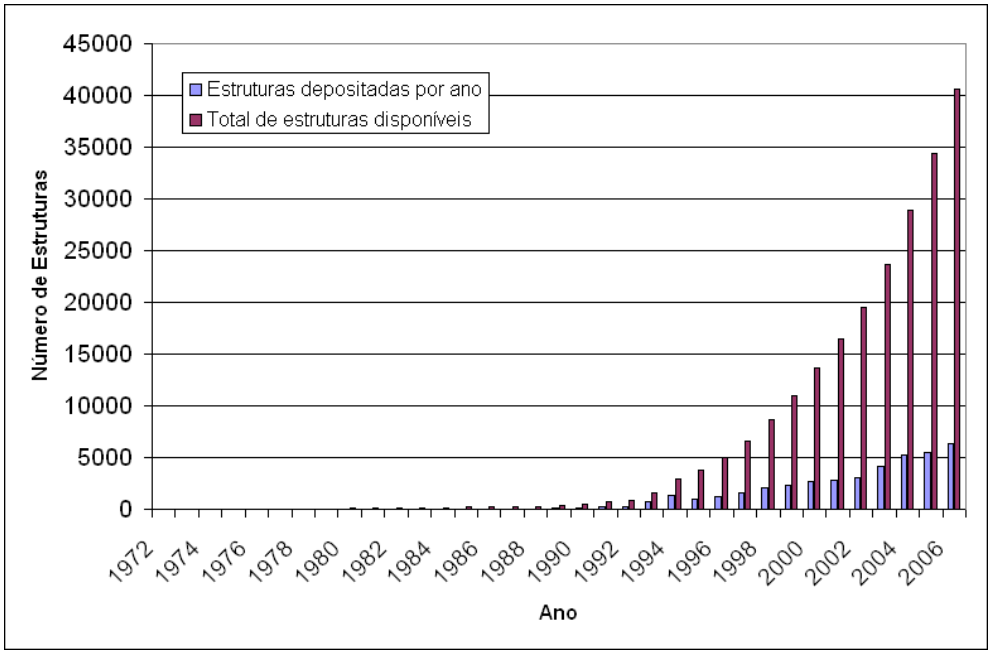


Figura 2: Número de estruturas protéicas depositadas no banco de dados PDB. A grande discrepância entre o número de estruturas depositadas e o número de motivos (Figura 3) ocorre uma vez que os motivos se repetem nas estruturas resolvidas. Adaptado de http://www.rcsb.org/pdb/statistics/contentGrowthChart.do?content=total&seqid=100, última atualização 16 de dezembro de 2006.

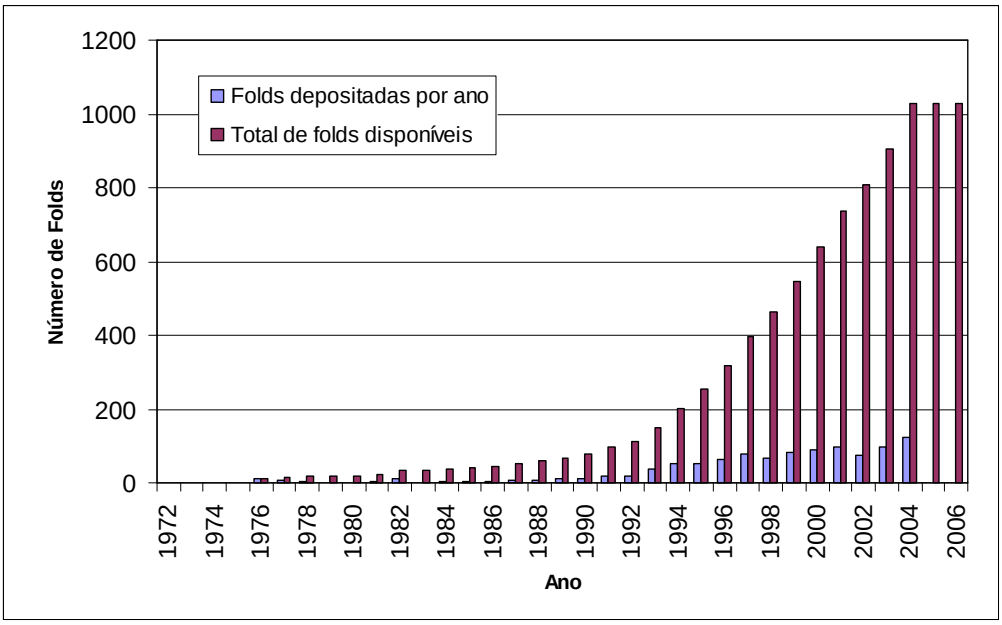


Figura 3: Número de motivos estruturais não redundantes (folds) classificados pelo SCOP. Adaptado de http://www.rcsb.org/pdb/contentGrowthChart.do?content=fold-scop, última atualização 11 de dezembro de 2006.

Neste trabalho, também, será utilizado o termo "polipeptídeo" ao fazer referência a uma proteína. Este trabalho é constituído de uma proposta inovadora na predição de estruturas 3D de polipeptídeos, onde a predição é realizada utilizando apenas ângulos de estruturas já conhecidas, o que não é observado em outros métodos observados na literatura. Esta metodologia se caracteriza por ser baseada em conhecimento (*knowledge-based*).

O Capítulo 1 apresenta uma revisão sobre os principais conceitos da área de Biologia Molecular que serão necessários para a compreensão do tema após apresentado. Neste sentido, no Capítulo 2 são apresentados os conceitos sobre os aminoácidos, as proteínas e sua organização estrutural hierárquica. Posteriormente, são abordadas as metodologias existentes para a predição de estruturas terciárias, como pode ser visto no Capítulo 3. No Capítulo 4 é apresentada a metodologia desenvolvida neste trabalho, como novo método para predição de estruturas protéicas. O mesmo também apresenta uma pequena simulação que serve como uma primeira validação da metodologia proposta. Os resultados deste trabalho são apresentados no Capítulo 5, que mostra imagens ilustrativas, gráficos explicativos e, por fim, uma comparação do trabalho desenvolvido com metodologias existentes. Enfim, no Capítulo 6 é apresentada uma conclusão que aborda uma crítica ao trabalho desenvolvido, assim como algumas das problemáticas do presente trabalho e de possíveis desdobramentos futuros.

Capítulo 2. Estado da Arte

Este Capítulo aborda como tema as proteínas. Uma vez que as proteínas e suas estruturas físico-químicas estão compreendidas, fica mais fácil o entendimento dos métodos computacionais para a predição de suas estruturas 3D.

As proteínas são compostos orgânicos que se caracterizam por possuírem diversas funções nos mais diversos organismos. Há vários tipos de proteínas com funções diferentes, podendo as mesmas ser de transporte, defesa, armazenamento, regulação, entre outras [7].

2.1 Proteínas

As proteínas são compostas por estruturas menores denominadas aminoácidos. A maioria das proteínas são formadas por mais de 100 aminoácidos, sendo algumas delas constituídas por mais de 4000 aminoácidos [8].

Os aminoácidos se caracterizam por compartilharem uma estrutura básica, que consiste em um átomo de carbono central (C, que, também, pode ser chamado de carbono alfa), um grupo amina (NH_2) e um grupo carboxila (COOH); toda esta estrutura não muda e constitui a cadeia principal das proteínas. A estrutura, ainda, possui uma cadeia lateral (radical R) que é variável. Esta estrutura pode ser vista na Figura 4.

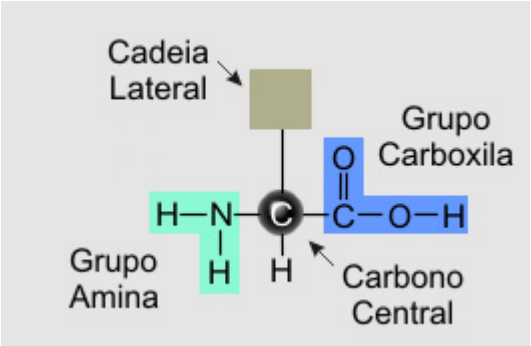


Figura 4: Estrutura de um aminoácido. Adaptado de Lehninger et al. [7].

As cadeias polipeptídicas possuem um sentido, uma vez que seus componentes têm extremidades diferentes. Por convenção, a ponta amídica é considerada o início da cadeia; portanto, uma seqüência de aminoácidos é escrita a partir da sua porção amino terminal (N-terminal) para a porção carboxi terminal (C-terminal) [8].

Na composição das proteínas, pode-se observar a existência de 20 tipos de aminoácidos, embora nem todos estejam presentes em todas as proteínas e, dependendo da cadeia lateral (radical R), os aminoácidos apresentam características peculiares. As cadeias laterais oferecem a versatilidade físico-química requerida para gerar os diferentes padrões de enovelamento (*folds* ou topologias) [9].

No final da década de 30, Linus Pauling e Robert Corey iniciaram estudos cristalográficos por difração de raios X da estrutura de aminoácidos e peptídeos cuja meta era a obtenção das distâncias e ângulos padrões das ligações entre os átomos de um aminoácido, e a utilização desta informação na predição de conformações protéicas [10,11] . O mais importante de seus achados foi que a unidade de ligação peptídica é rígida e plana. A ligação entre o carbono da carboxila e o nitrogênio da amina não é livre para rodar porque essa ligação tem um caráter parcial de ligação dupla e o mesmo é representado pelo ângulo de torsão ômega (ω) [9] (Figura 5).

A ligação entre o carbono alfa e o carbono carboxílico e entre o carbono alfa e o nitrogênio peptídico são ambas simples e, portanto, possuem considerável grau de liberdade de rotação em torno da ligação peptídica rígida (ângulos diedros (φ) e psi (ψ) da cadeia principal – Figura 5).

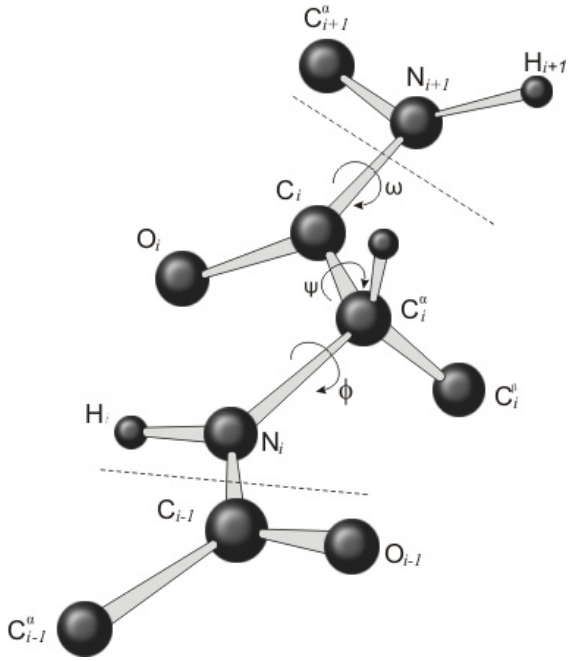


Figura 5: Definição dos ângulos de rotação da conformação de uma cadeia polipeptídica. Adaptado de Lesk [9].

A cadeia principal de uma proteína tipicamente possui centenas de ligações individuais. Porém, cada proteína tem uma função química ou estrutural específica, sugerindo que cada uma tenha uma única estrutura 3D estável nativa [7]. As

conformações nativas das proteínas são obtidas através de forças termodinâmicas agindo sobre a cadeia principal [12].

Se fosse possível calcular com suficiente exatidão as energias e as entropias de diferentes conformações, seria plausível predizer estruturas de proteínas a partir das seqüências de aminoácido apenas com base nos princípios físico-químicos [3].

2.2 Organização Estrutural das Proteínas

A natureza hierárquica da arquitetura de proteínas pode ser descrita em níveis de estruturas, classificando em estrutura primária, estrutura secundária, estrutura terciária e estrutura quaternária [7].

2.2.1 Estrutura Primária

A estrutura primária de uma proteína (Figura 6) é simplesmente a sua seqüência linear de aminoácidos.

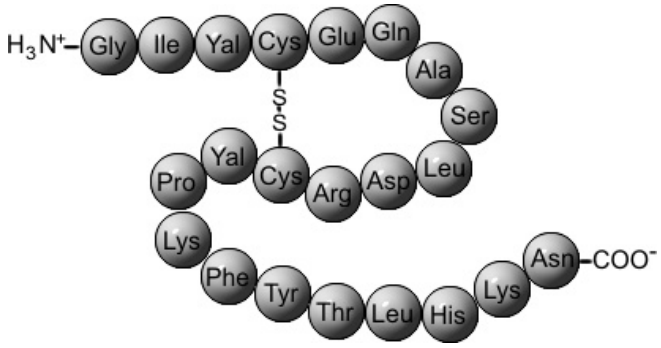


Figura 6: Estrutura primária de uma proteína. Adaptado de Anfinsen [13].

Este é o nível estrutural mais simples e mais importante. Definidos pela informação genética da célula a estrutura primária de uma proteína pode variar em três aspectos: o número de aminoácidos, a seqüência e a natureza dos aminoácidos [7].

2.2.2 Estrutura Secundária

O termo estrutura secundária se refere à conformação local de alguma parte de um polipeptídeo. Ainda refere-se aos padrões comuns de enovelamento regular de sua cadeia principal dos polipeptídeos. Alguns tipos de estruturas secundárias são particularmente estáveis e de alta freqüência em proteínas. As mais proeminentes conformações são as hélices α e as folhas β. Além destas, há, também, uma estrutura irregular chamada volta, como podem ser vistas na Figura 7 (imagem gerada a partir do *software* Swiss-PdbViewer [14]).

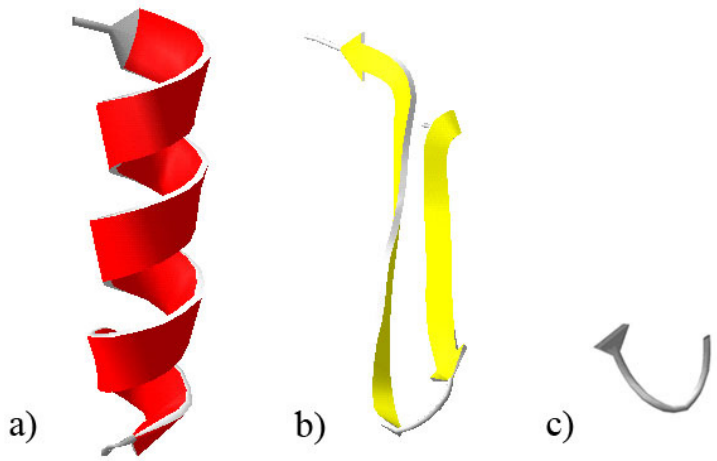


Figura 7: Representação do tipo *ribbons* das estruturas secundárias regulares presentes em proteínas: (a) hélice α, (b) folha β, e (c) volta que é uma estrutura secundária não regular.

A estrutura hélice α se caracteriza por ser uma estrutura helicoidal definida pelas interações entre o grupo carboxila (C=O) com o grupo amina (N-H), formando pontes de hidrogênio. Em uma hélice α o grupo carboxila do primeiro aminoácido se liga ao grupo amina do quinto aminoácido para formar a ponte de hidrogênio. Todos os C=O apontam para uma direção e os N-H para a direção oposta na hélice, formando pontes de hidrogênio paralelas ao eixo da hélice [9].

Na conformação de folha β, a cadeia principal do polipeptídeo é estendida em contraposição à estrutura helicoidal das hélices α. As cadeias do polipeptídeo, com a conformação de folha β, devem ser organizadas lado a lado. Neste arranjo, pontes de hidrogênio são formadas entre segmentos adjacentes da cadeia [9].

Um terceiro tipo de estrutura, mas irregular, denominada volta, surge dos resíduos de aminoácidos que estão em regiões onde o polipeptídeo muda a sua direção. Estes são os elementos que unem sucessivas estruturas secundárias regulares.

Além destas, ainda há um tipo de estrutura que se caracteriza por não possuir uma forma bem definida. A mesma é chamada de *coil* e não se classifica como estrutura secundária, mas aparece como parte de estruturas terciárias.

2.2.3 Estrutura Terciária

O arranjo 3D global de todos os átomos em uma proteína é chamado de estrutura terciária da proteína, como pode ser observado na Figura 8 (imagem gerada a partir do *software* Swiss-PdbViewer [14]). A estrutura terciária descreve como as estruturas secundárias se arranjam no espaço tridimensional para formar a estrutura nativa da proteína. Esta estrutura resulta de interações físicas e químicas que se estabelecem entre as cadeias laterais dos aminoácidos e, dessas, com o meio que as cerca. Essas interações estabilizam, termodinamicamente, a estrutura de forma apropriada, condições estas necessárias para a efetiva atividade biológica das proteínas [9].

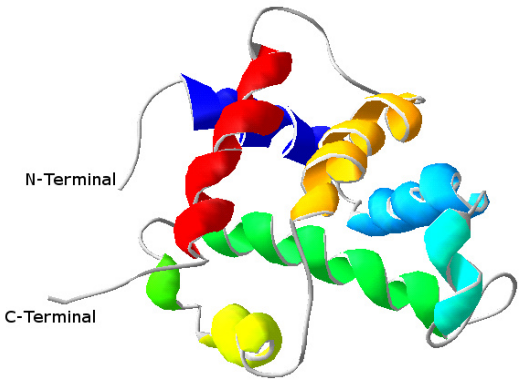


Figura 8: Representação do tipo *ribbons* da subunidade A da hemoglobina humana adulta (código PDB 2H35) sem o grupo heme. Estrutura colorida pela sucessão da estrutura secundária, onde a parte N-Terminal está em azul e a C-Terminal em vermelho.

2.2.4 Estrutura Quaternária

Muitas proteínas são compostas de duas ou mais cadeias polipeptídicas, normalmente referidas como subunidades. A estrutura quaternária se compreende à estrutura obtida por meio de associações de proteínas já organizadas em nível terciário, como se pode ver na Figura 9 (imagem gerada a partir do *software* Swiss-PdbViewer [14]).

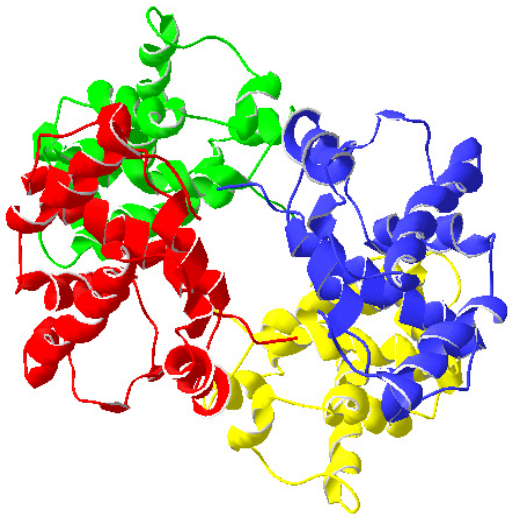


Figura 9: Representação do tipo *ribbons* de uma estrutura quartenária da proteína hemoglobina humana adulta (código PDB 2H35), sem o grupo heme. Estrutura colorida diferenciando cada subunidade.

Capítulo 3. Metodologias para Predição Computacional da Estrutura Terciária de Proteínas

O processo de enovelamento de uma proteína ainda é um problema que não possui solução. Os primeiros experimentos foram feitos por Anfinsen na década de 1960, mostrando que o enovelamento da maioria das proteínas globulares é um fenômeno puramente físico, dependendo somente da seqüência de aminoácidos da proteína e do solvente [13].

Se seqüências de aminoácidos contêm informações suficientes para especificar estruturas 3D de proteínas, deveria ser possível desenvolver um algoritmo para predizer a estrutura de uma proteína a partir da seqüência de aminoácido. Por conseguinte, o problema fundamental *a priori* é a predição da estrutura 3D de proteína a partir da seqüência de aminoácidos.

Assevera-se desejável a predição da estrutura 3D a partir da sua seqüência de aminoácidos, tendo em vista que, experimentalmente, é possível a determinação de sua seqüência de aminoácidos. Contudo, a determinação experimental da estrutura 3D de uma proteína se caracteriza por possuir custo oneroso e, às vezes, a mesma se mostra inviável [16].

Também seria interessante poder predizer com precisão a função protéica a partir da seqüência de aminoácidos, e, eventualmente, construir novas proteínas. No entanto, sem uma compreensão de como as seqüências determinam a estrutura, estas outras metas não podem ser alcançadas.

Anfinsen [13] postulou que é possível definir um campo de força baseado nas interações físico-químicas entre átomos, incluindo o solvente, e usar um método de procura de enovelamento, tal como a Dinâmica Molecular (DM) ou Monte Carlo, para determinar a estrutura mais estável da proteína em uma determinada temperatura e solvente [17].

Há duas básicas metodologias computacionais que modelam a estrutura de proteínas. A primeira metodologia está baseada em simulação de forças físicas e dinâmicas moleculares. A segunda é baseada em comparações.

Metodologias comparativas exploram parâmetros do banco de dados de estruturas já existentes, para avaliar e predizer as estruturas a partir da seqüência de

aminoácidos. A predição da estrutura da proteína pela seqüência de aminoácidos, desde seu início, permanece sendo um problema não solucionado pela Bioinformática [16].

Nas seções seguintes serão abordadas as metodologias computacionais de predição, como a modelagem comparativa por homologia, o reconhecimento de motivos via *threding* e a predição *ab initio*. Além destas três metodologias, é apresentada uma seção que trata do CASP [18] (*Critical Assessment of Techniques for Protein Structure Prediction*), um encontro onde as ferramentas de predição competem entre si, aos fins de determinar qual é a melhor ferramenta com a melhor capacidade de predizer sobre as proteínas e, conseqüentemente, o encontro acaba por apresentar o avanço na área de predições.

3.1 Modelagem Comparativa por Homologia

Duas proteínas se dizem homólogas, quando compartilham um antepassado comum. É freqüente se afirmar que existe homologia, quando se observa mais do que 30% de identidade entre as seqüências. Esta regra, ainda que limitada, revela-se extremamente útil, quando a única informação disponível é a estrutura primária, pois é improvável que duas proteínas, com as seqüências de aminoácidos muito parecidas, tenham evoluído independentemente [19].

Denomina-se alinhamento de proteínas o arranjo de seqüências em que os resíduos alinhados correspondem ao mesmo resíduo num antepassado comum. Enquanto um alinhamento possa utilizar apenas duas seqüências, um alinhamento múltiplo é mais confiável do ponto de vista biológico, e pode conter muito mais do que informação evolutiva. O alinhamento múltiplo pode revelar a localização de centros funcionais de proteínas homólogas, identificados por um ou mais grupos de resíduos consecutivos muito conservados [19].

A modelagem comparativa por homologia tem como escopo o uso de uma seqüência de aminoácidos a fim de gerar um molde com intuito de predizer a estrutura da proteína desejada. Para a obtenção do molde, é comparada a seqüência de aminoácidos com as estruturas experimentais de um determinado banco de dados [19].

Nesta técnica, primeiramente a seqüência de uma proteína é alinhada (uma seqüência “alvo”) contra a seqüência de outra proteína de estrutura conhecida – molde ou *template*. No alinhamento, é feita uma comparação que procura determinar o grau de similaridade (identidade) entre duas ou mais seqüências, ou a similaridade entre fragmentos destas seqüências [2].

3.2 Reconhecimento de Motivos via Threading

Threading é a metodologia que consiste na entrada de uma seqüência de aminoácidos, sem estruturas caracterizadas, e a partir da mesma é rapidamente computado um modelo baseado em um banco de topologias existentes, por exemplo, SCOP [20]. Logo após, o modelo é avaliado para determinar se o aminoácido desconhecido se ajusta a uma estrutura de referência.

Esta metodologia é freqüentemente usada com intuito de descobrir homologias remotas que não podem ser descobertas por alinhamento de seqüências padrão. Se fragmentos da seqüência se ajustam bem aos enovelamentos, um alinhamento geralmente pode ser deduzido, mesmo que não haja informação suficiente para construir um modelo completo [3].

3.3 Ab Initio

A predição da estrutura terciária de proteínas por meio da metodologia *ab initio* tenta predizer a conformação nativa de uma proteína apenas a partir da sua seqüência de aminoácidos. Esta predição se mostra um teste fundamental da compreensão do enovelamento da proteína e um desafio prático importante, tendo em vista que está sendo produzido um grande número de seqüências de proteínas sem que suas informações 3D sejam conhecidas [2].

A metodologia de predição *ab initio* requer três elementos: uma representação da geometria da proteína, uma função potencial e uma técnica de busca do espaço da superfície da energia livre [17].

A representação da geometria se dá por meio do cálculo da energia potencial apropriada para o modelo, a fim de procurar o espaço para o enovelamento do mesmo. A função potencial, podendo ser física ou estatística, é o que ao final determina se o método é ou não capaz de predizer a estrutura nativa de uma proteína [17]. Já as técnicas de buscas conformacionais se caracterizam por se dividirem em três principais técnicas, sendo Algoritmos Genéticos, Monte Carlo e DM [17-19]. Nesta metodologia, são calculadas as energias envolvidas no processo do enovelamento para encontrar a estrutura com a energia livre mais baixa. Esta aproximação é baseada na hipótese termodinâmica, a qual indica que a estrutura nativa de uma proteína corresponde ao mínimo global da superfície de energia livre [20,21]. Esta metodologia pode ter êxito em identificar enovelamentos de proteína, até mesmo quando não podem ser preditos os detalhes da estrutura [17].

3.4 CASP: Critical Assessment of Techniques for Protein Structure Prediction

Esta seção não trata de outra metodologia para predição de estruturas, mas, sim, discorre sobre um evento que ocorre a cada dois anos e é chamado de CASP [6]. O mesmo reúne vários grupos de pesquisa que desenvolvem métodos para predições de estruturas e, nesta ocasião, competem com o intuito de avaliar seus métodos.

Durante os últimos anos, o CASP vem monitorando o estado da arte em relação à modelagem de estrutura de proteína a partir de sua seqüência de aminoácidos. Por meio das sete edições do CASP, pôde-se ver um progresso significativo nas predições [18], sendo a última edição ocorrida em Asilomar, 26-30 de novembro de 2006.

O CASP possui três áreas de competição: modelagem por homologia, *threading* e predição *ab initio.* Ademais, vinha mostrando que as predições de estrutura baseadas

em homologia tinham grande destaque na produção de modelos razoáveis nos casos em que se possuía uma estrutura significativamente homóloga disponível [3].

Nestes últimos anos, a metodologia de predição *ab initio* apresentou-se bem classificada no CASP, tanto que a mesma se mostra como metodologia provavelmente mais promissora [18].

Capítulo 4. Metodologia Proposta

O referencial teórico, apresentado até então, oferece uma base comum de conceitos que são direta e indiretamente utilizados nesta proposta.

O Capítulo 2 definiu alguns conceitos de Bioinformática Estrutural, apresentou a estrutura das proteínas, suas conformações e os ângulos que as formam. Também foi explicado como as proteínas podem ser classificadas de acordo com sua organização hierárquica estrutural. Já o Capítulo 3 apresentou os métodos que são utilizados para, por meio de computadores, tentar descobrir a estrutura terciária de uma proteína, tendo como base apenas a seqüência de aminoácidos (estrutura primária).

A proposta deste trabalho é, a partir da seqüência dos aminoácidos de uma proteína ou polipeptídeo, predizer sua estrutura terciária. Para a predição, utilizam-se, basicamente, os ângulos φ, ψ e ω de proteínas que possuem suas estruturas terciárias já conhecidas do PDB [4,25].

A seguir é apresentada a descrição de como foi desenvolvido este trabalho para a predição computacional de estruturas terciárias de polipeptídeos.

4.1 Problema

Um dos grandes desafios da Bioinformática é a determinação e validação de novos genes e suas proteínas correspondentes, ou seja, a conversão dos dados obtidos a partir dos seqüenciamentos em informação útil. Um dos componentes cruciais deste desafio reside no estudo das estruturas terciárias das proteínas, sejam elas determinadas experimentalmente ou computacionalmente [26].

A estrutura primária de proteínas se caracteriza por ser de fácil obtenção por meio de métodos experimentais. Já a estrutura terciária se caracteriza não só por ser um processo demorado, mas por utilizar equipamentos extremamente caros.

De acordo com vários *benchmarks*, as metodologias de reconhecimento por homologia não selecionam o enovelamento correto na base de dados para aproximadamente 50% dos casos em que não existe similaridade significativa da seqüência. Neste caso, o reconhecimento tem, também, a limitação de que nenhum novo enovelamento (*fold*) pode ser proposto, já que todas as predições são baseadas em estruturas previamente conhecidas [27].

A predição da estrutura de proteínas, utilizando a metodologia *ab initio*, possui a característica de permitir que sejam gerados novos *folds*. Porém, mostra-se muito lenta, pois é necessário um grande número de cálculos para a obtenção de uma resposta. Isto ocorre, porque o método consiste em calcular todas as energias envolvidas no processo do enovelamento e, então, encontrar a estrutura com a energia livre mais baixa [22].

A partir destas questões, foi possível pensar em uma proposta de solução para o problema da predição computacional da estrutura terciária de polipeptídeos, que não apenas pudesse gerar novos *folds*, como também fosse mais rápida do que calcular todas as energias envolvidas no enovelamento.

Esta proposta se caracteriza pela utilização de fragmentos de estruturas protéicas já conhecidas, ângulos de torção da cadeia principal dos polipeptídeos e avaliação do modelo do polipeptídeo gerado.

4.2 Proposta de Solução

Este trabalho se constitui em uma proposta para predição de estruturas de polipeptídeos, baseada em conhecimentos (*knowledge-based*) com o uso do banco de dados PDB [4,25] de estruturas 3D, onde, com o uso de fragmentos de estruturas já conhecidas, calculam-se ângulos φ, ψ e ω para gerar os modelos de polipeptídeos.

O método toma como princípio analisar todos os possíveis fragmentos de cinco aminoácidos consecutivos (pentapeptídeo) de um polipeptídeo, a fim de descobrir possíveis proteínas com fragmentos iguais ou semelhantes no PDB, ver Figura 10.

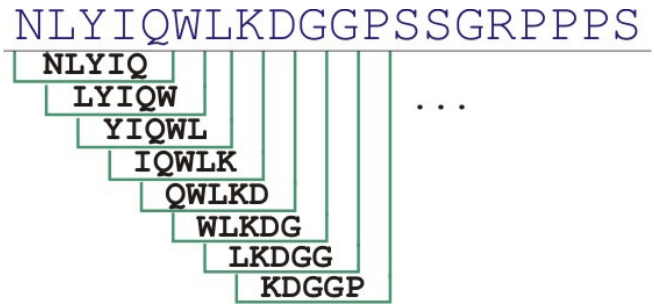


Figura 10: Representação de um polipeptídeo separado em fragmentos de cinco aminoácidos consecutivos (pentapeptídeos).

Para cada pentapeptídeo é feita uma busca no banco de dados estrutural 3D (PDB) com o intuito de descobrir se existem outras proteínas que possuem o mesmo fragmento. Mostra-se de grande importância que a proteína utilizada na simulação seja retirada do banco de dados, pois, no referente trabalho, o que se pretende provar é que, a partir de estruturas não homólogas, seja possível gerar a estrutura 3D de polipeptídeos.

Como resposta às buscas de cada pentapeptídeo, são retornadas proteínas que contenham fragmentos iguais ou que possuam um alto grau de identidade. Deste modo,

espera-se que as mesmas possuam características físico-químicas semelhantes à estrutura experimental.

Com os fragmentos identificados são analisados os aminoácidos centrais (3º aminoácido) de cada pentapeptídeo, como pode ser visto na Figura 11.

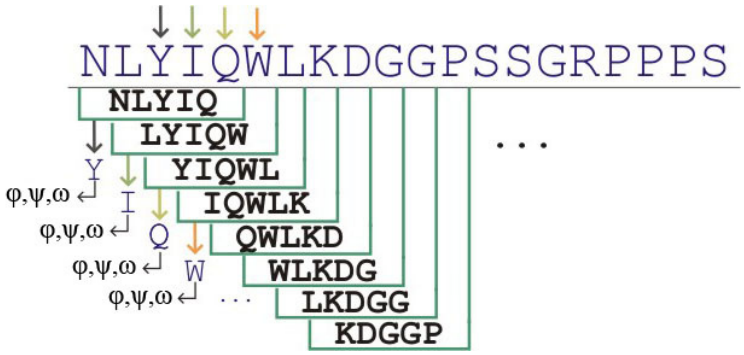


Figura 11: Representação de um polipeptídeo separado em pentapeptídeos. As setas indicam os aminoácidos centrais de cada pentapeptídeo e seus respectivos ângulos de rotação φ, ψ e também o ângulo da ligação peptídica ω.

Com os pentapeptídeos definidos e os aminoácidos centrais identificados, então são obtidos os ângulos φ, ψ e ω dos aminoácidos centrais, como pode ser visto na Figura 11.

Para cada pentapeptídeo analisado, como já dito anteriormente, espera-se várias proteínas com fragmentos similares. Ademais, cada proteína possuirá ao menos um dos pentapeptídeos que retornará de ângulos φ, ψ e ω. Portanto, um mesmo fragmento de cinco aminoácidos pode ser encontrado em mais de uma proteína e, em cada uma delas, possível de assumir valores diferentes.

Esta abordagem da utilização de ângulos φ, ψ e ω se justifica, pois, conforme Hovmöller *et al.* [28], cada aminoácido possui ângulos preferenciais. Os ângulos preferenciais – de cada um dos 20 aminoácidos – são apresentados por meio de gráficos de Ramachandran, conforme as estruturas que estavam depositadas no banco de dados do PDB em 03 de janeiro de 2002.

A abordagem atual possui como princípio não apenas levar em consideração cada aminoácido isolado, mas, sim, grupos de cinco aminoácidos, tendo como base que o aminoácido central carrega consigo possíveis influências de aminoácidos vizinhos com base nos princípios físico-químicos.

Após esta análise, são gerados modelos de polipeptídeos com os ângulos φ, ψ e ω do aminoácido central de cada fragmento. As estruturas geradas são, então, analisadas, a fim de encontrar quais foram os melhores modelos gerados.

A presente solução possui a característica de não utilizar os dois aminoácidos iniciais, nem os finais, pois a solução apresentada se caracteriza por analisar apenas o aminoácido central (3º aminoácido) de cada pentapeptídeo. Este problema foi solucionado, utilizando os ângulos dos dois primeiros aminoácidos do pentapeptídeo referente ao primeiro fragmento (utilizado na contrução do molde) e os dois últimos

aminoácidos do pentapeptídeo utilizado como último fragmento (utilizado na contrução do molde).

4.3 Simulação Manual

Esta etapa tem como principal objetivo provar que com o método proposto é possível predizer uma estrutura terciária com razoável índice de precisão. Logo, então, foi feita uma primeira busca manualmente, na qual foi utilizada a mini-proteína Trp-Cage (código PDB 1L2Y, [30]) que, por sua vez, caracteriza-se por ser uma estrutura de apenas 20 aminoácidos o que ajuda para uma primeira validação.

Para a realização da simulação manual, definiram-se os passos a serem realizados, uma vez que a ordem dos processos, ainda, não havia sido estabelecida. Esta primeira simulação utilizou como suporte a base de dados de estruturas 3D do PDB [25] e a busca na base de dados foi efetuada com o *software* BLAST [29].

Com o intuito de obter uma melhor idéia sobre as etapas a serem realizadas na simulação e a ordem em que as mesmas devem ocorrer, foi desenvolvido um fluxograma que apresenta a seqüência de cada etapa a ser realizada para obter a predição da estrutura 3D de um polipeptídeo. O fluxograma, ora analisado, pode ser visto na Figura 12.

Após definidos todos os passos a serem executados na simulação, pode-se observar que o primeiro a ser realizado é a fragmentação da seqüência de aminoácidos. A fragmentação proposta neste trabalho é simples, pois, tendo a seqüência de aminoácidos, cada fragmento se caracteriza por ser um conjunto de cinco aminoácidos consecutivos (pentapeptídeos), como explicado anteriormente na Figura 10.

Com os pentapeptídeos definidos é realizada uma busca na base de dados do PDB, utilizando o *software* BLASTP [29]. Tendo em vista que este tipo de busca não retorna nenhum tipo de resultado com apenas cinco aminoácidos (número de aminoácidos presente em cada pentapeptídeo), antes e depois de cada pentapeptídeo se colocou XXXXX, resultando em uma busca com 15 aminoácidos, ou seja, um número mínimo de aminoácidos para que a busca pudesse ser feita.

A escolha de utilizar cinco X foi adotada; logo, na busca, o X representa qualquer um dos 20 aminoácidos, o BLASTP, mesmo sendo feito com 15 aminoácidos apenas os cinco centrais serão realmente procurados. Ademais, com o intuito de melhorar e, também, aumentar o resultado da busca, modificaram-se outros campos. Ao invés do uso da matriz de substituição BLOSUM62, utilizou-se a BLOSUM50. Também foram removidos os filtros e, ainda, o *Expect Value* (função de escore) teve alteração de 10 para 1000. Todas estas mudanças foram realizadas com o objetivo de fazer com que a busca retornasse o maior número de proteínas referente ao fragmento procurado.

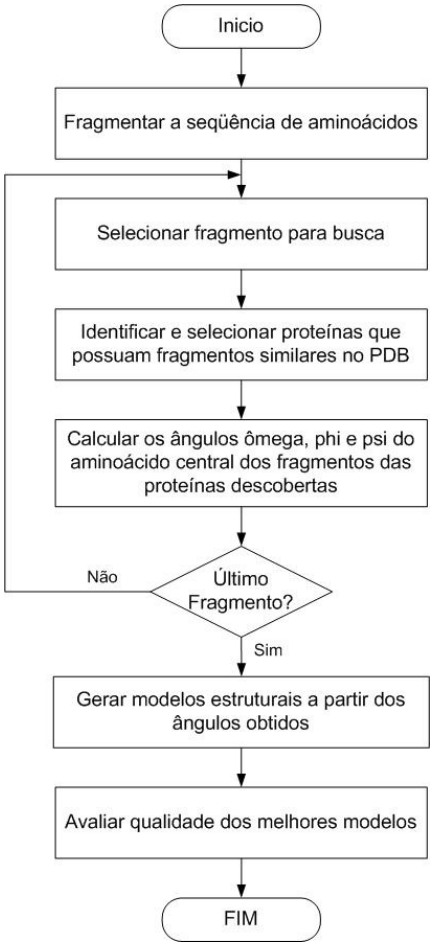


Figura 12: Fluxograma das etapas a serem realizadas para a predição da estrutura 3D do polipeptídeo, com base no método proposto.

Com as proteínas que possuem pentapeptídeos similares, extraiu-se os ângulos φ, ψ e ω dos pentapeptídeos, como pode ser observado na Figura 11. Com os ângulos φ, ψ e ω de todas as proteínas encontradas, foi construído o novo modelo.

A geração do primeiro modelo se deu utilizando os ângulos da primeira proteína encontrada em cada busca do BLASTP aplicada a cada um dos pentapeptídeos. Caso a primeira proteína encontrada na busca fosse a Trp-Cage (código PDB 1L2Y, [30]), então, utilizava-se a segunda proteína.

Determinadas todas as proteínas, foram utilizados os seus φ, ψ e ω (obtidos pelo Swiss PDBViewer) referentes ao aminoácido central (como explicitado anteriormente e ilustrado na Figura 11) para, então, junto ao programa Swiss PDBViewer, gerar o primeiro modelo.

Após a geração do modelo, pôde-se observar que a estrutura permaneceu similar, a obtida experimentalmente (Trp-Cage), pois a mesma se caracterizou por continuar possuindo apenas uma hélice e torções idêncas. Embora não muito exatas – para um primeiro experimento – resultaram em uma estrutura com um enovelamento semelhante, conforme a Figura 13.

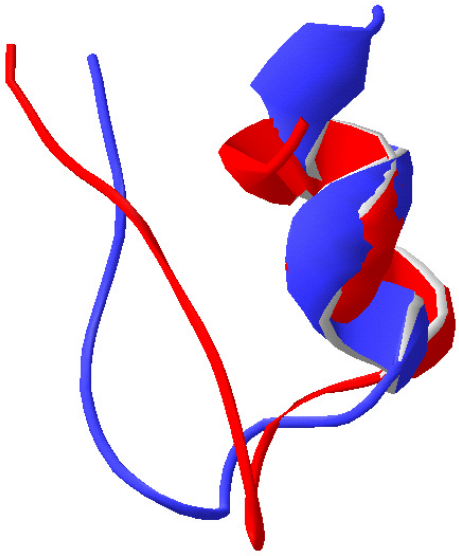


Figura 13: Representação do tipo *ribbons* da estrutura 3D (topologia) da mini-proteína Trp-Cage. Superposição da estrutura experimental (código PBD 1L2Y, [30]), em vermelho, e da estrutura predita (em azul) com a proposta apresentada neste trabalho.

Seguindo o fluxograma da Figura 12 para a avaliação, foi utilizada a métrica RMSD (*Root Mean Square Deviation* ou desvio médio quadrático) entre a proteína experimental e a predita, o qual retornou um RMSD de 4.1 Å.

4.4 Implementação

Tendo sido comprovado, por meio de uma simulação manual, que é possível se encontrar uma estrutura com um *fold* semelhante ao esperado, seguiu-se, então, à etapa de implementação.

Para a implementação, escolheu-se a linguagem de programação Python [31], visto que além de ser uma linguagem orientada a objetos, ainda possui grande facilidade de manipulação de arquivos, uma vez que seria necessário analisar arquivos retornados de buscas do BLASTP; os arquivos PDBs; os arquivos com os ângulos φ, ψ e ω; entre outros.

A linguagem Python se mostrou uma alternativa muito interessante, visto que a referente linguagem se caracteriza por permitir fácil manipulação de *strings*, construção de expressões regulares, criação de arquivos entre outros.

Com intuito de auxílio, foi utilizada a biblioteca BioPython [32,35], a qual possui várias funções prontas como por exemplo: realizar *downloads* de bases de dados – ou de apenas uma estrutura – biológica(s); realizar alinhamento múltiplo de seqüências; realizar buscas no BLAST (utilizada neste trabalho); entre outros. Esta biblioteca, por ter sido implementada em Python, possui, também, a característica de

oferecer recursos para manipulação de string e expressões regulares, o que se mostrou como facilitador para a tarefa de implementação do presente trabalho.

Depois de definidas as etapas (Figura 12) e a linguagem de programação, foram estabelecidas as seguintes funções:

fragmenta: caracteriza-se de uma pequena função que se encarrega de pegar qualquer número de aminoácidos e separá-los em cinco aminoácidos consecutivos, gerando, assim, os pentapeptídeos.

qblast: função responsável por fazer a busca no BLAST [29]. Esta função utiliza o BLAST do NCBI (http://www.ncbi.nlm.nih.gov/BLAST/). Além de possuir grande aceitação pela comunidade acadêmica, dispõe de facilidades para sua automatização, utilizando linguagens de programação.

A *qblast* se caracteriza por ter sido adaptada do conjunto de bibliotecas BioPython [35] (disponível em http://biopython.org/). A mesma sofreu alterações a fim de ajustar alguns parâmetros da busca para que fosse capaz de realizar a busca BLASTP para pequenos fragmentos. Com base em um tipo especial de BLAST (chamado *nearly exact matches*) para busca de pequenos fragmentos, a função qblast original do BioPython foi modificada para realizar a busca *nearly exact matches*. Contudo, alguns dos valores da busca *default* foram alterados, ficando os mesmos da seguinte maneira: Expect Value, 20000; Word Size, 2; Matriz de Segregação, PAM30; Descriptions, 1000; Alignments, 50.

analyseblast: se caracteriza por ser uma função que analisa o XML (gerado pela busca realizada na função qblast) e retorna uma lista com os códigos PDBs e seus referentes fragmentos, encontrados na busca do BLASTP. Com o intuito de auxiliar na análise do arquivo XML, utilizou-se a biblioteca DOM, que facilita o tratamento de arquivos XML em Python.

removeSeq: embora essa não seja uma das funções essenciais, a mesma se mostra importante, pois ela é capaz de remover uma proteína da busca, uma vez passando o seu código PDB.

A importância da presente função está balizada na idéia de mostrar que é possível reconstruir a proteína, estando esta excluída da simulação.

listadePDBs: função responsável por gerar uma lista que contenha todos os códigos PDBs que devem ser baixados para a simulação. Esta função se justifica, pois, após terem sido descobertos todos os arquivos PDBs, é necessário obter os mesmos para que seja possível o cálculo dos ângulos φ, ψ e ω. Tendo em vista que a busca do BLAST pode retornar códigos PDBs repetidos, esta função já se responsabiliza por remover possível códigos PDBs replicados, evitando, assim, *downloads* desnecessários.

downloadDosPDBs: a partir da lista dos códigos PDBs faz o *download* dos seus referentes arquivos.

geraPhiPsiOmega: responsável por obter os ângulos φ, ψ e ω do aminoácido central de cada pentapeptídeo nas estruturas obtidas por meio da busca do BLASTP. Para a realização desta tarefa, mostrou-se necessária a utilização do *software* Torsions (desenvolvido pelo Dr. Andrew C. R. Martin – *University College London & SciTech Software*) que calcula os ângulos φ, ψ e ω para cada aminoácido a partir de um dado arquivo PDB.

generateStructures: caracteriza-se por ser uma função recursiva a qual gera todas as estruturas, com base em todos os ângulos de seus referentes aminoácidos. Para a geração das estruturas, foi utilizado o *software* tleap, um módulo do conjunto de programas de simulação denominados AMBER [33].

Depois da geração do modelo, o mesmo é avaliado com o uso do *software* ptraj, distribuído juntamente com o AMBER. Para a avaliação da qualidade do modelo, está sendo utilizado o RMSD da estrutura predita *versus* a experimental.

Algoritmo 1: Chamada das funções para a simulação da 1L2Y

```
1.  seq ⇐ fragmenta('NLYIQWLKDGGPSSGRPPPS')
2.  estruturas ⇐ []
3.  para x in range(0,len(seq)):
4.        busca ⇐ qblast("blastp", "pdb", seq[x])
5.        busca2 ⇐ analyseblast(busca)
6.        removesimilar(busca2)
7.        estruturas.append(busca2)
8.  removeSeq(estruturas, '1L2Y')
9.  lista ⇐ listadePDBs(estruturas)
10. downloadDosPDBs(lista)
11. resultpar ⇐ geraPhiPsiOmega(estruturas, 'NLYIQWLKDGGPSSGRPPPS')
12. generateStructures(resultpar, 'NLYIQWLKDGGPSSGRPPPS')
```

Ao desenvolver o algoritmo, encontrou-se o problema de que para cada pentapeptídeo, o mesmo poderia possuir um número diferente de ângulos – devido ao número de fragmentos retornado pelo BLAST, ser variável, dependendo do fragmento utilizado na busca – o que dificultaria utilizar laços condicionais que são comumente utilizados na criação de algoritmos.

Chamemos os fragmentos de F_1, F_2,..., então em uma proteína com k aminoácidos irão existir $k-4$ fragmentos. Para cada fragmento F_i a busca no BLAST retornará um conjunto C de proteínas associadas, chamaremos então este conjunto de CF_i.

O que queremos é explorar as combinações de todos os elementos deste conjunto, ou seja, o conjunto cartesiano

$$CF_1 \times CF_2 \times CF_3 \times ... \times CF_{k-4}$$

A maneira computacional mais simples de explorar este produto é utilizando a recursão. O espaço combinacional acima é exponencial a *k*, por exemplo, se todos os CF_i tiverem tamanho 2 o número de combinações final será 2^{k-4}.

Não é possível explorar todas as combinações por problema de tempo e foge do escopo desta tese abordar sistematicamente todas as maneiras de fazer "prunning" no número de combinações, mas a título de experiência, limitou-se a dois tipos de experiências *ad hoc* simples de obter com um algoritmo recursivo, sendo elas utilizando a recursão trivial, início da variação pelas últimas entradas (chamada neste trabalho de normal ou *Backward*), e uma recursão cuja variação inicia nas primeiras entradas (chamada neste trabalho de inversa ou *Foreward*).

Para usuários leigos, à área da computação, segue abaixo uma explicação um pouco mais aprofundada do funcionamento da recursão.

A fim de exemplificar o processo de recursão, utiliza-se o Algoritmo 1 para simular um polipetídio com oito aminoácidos ($k = 8$). Durante o processo de simulação, ele iria fragmentar a proteína em pentapeptídeos, retornando em quatro fragmentos. Para cada um desses fragmentos (F_i), seria calculado os devidos ângulos.

Suponha-se que para os três primeiros pentapeptídeos tenha sido retornado dois ângulos ($CF_1 = CF_2 = CF_3 = 2$) e que para o quarto – e último fragmento – três ângulos ($CF_4 = 3$), como pode ser visto na Figura 14.

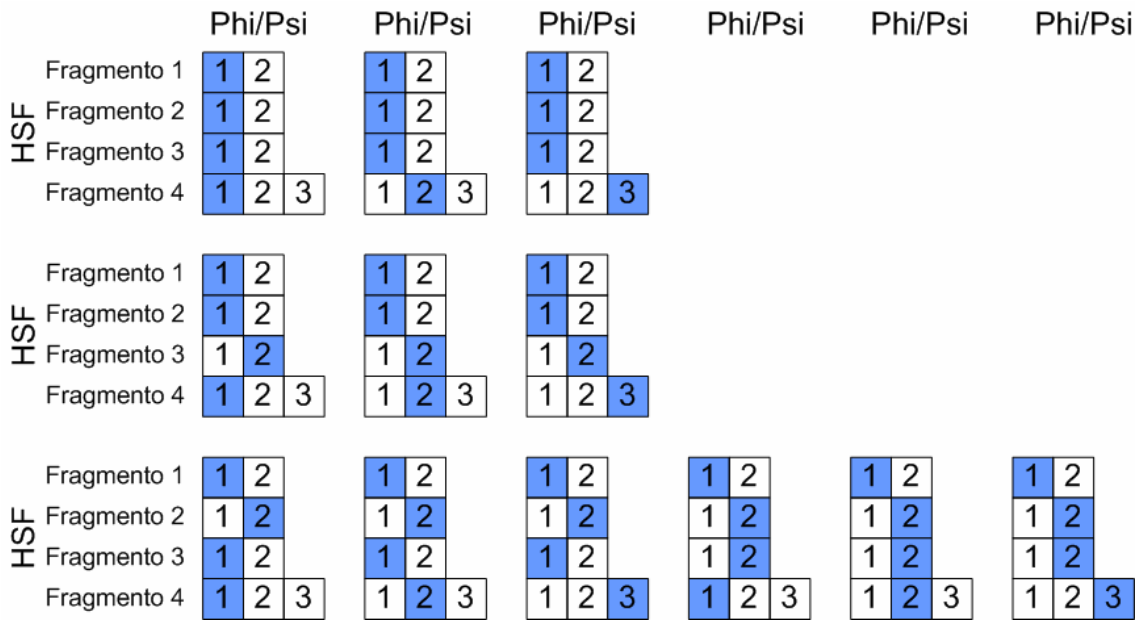


Figura 14: Recursão *Backward*, utilizando um polipeptídeo com apenas oito aminoácidos (quatro fragmentos).

O método da recursão se caracteriza por gerar todas as possibilidades existentes, primeiramente, fixando todos os fragmentos e, depois, variando os mesmos. Conforme ilustrado na Figura 14, a variação se dá do final para o início (recursão normal ou *Backward*), calculando o caso mais básico que consiste em utilizar os primeiros ângulos para cada um dos Fragmentos. Após o caso básico ser executado, inicia a recursão, vindo a esgotar todas as possibilidades de variação para o Fragmento 4 (parte superior da Figura 14).

Ao iniciar a variação do Fragmento 3, os fragmentos após eles (no caso desta simulação, apenas o Fragmento 4) são colocados novamente no caso base para então reiniciar a variação, como se pode ver na parte central da Figura 14.

Pode-se observar que na recursão para cada novo Fragmento que varia, todos os posteriores a ele são colocados em seus casos base. A variação só acaba quando todos os Fragmentos tenham sido utilizados. Por este motivo, a recursão se mostrou a melhor estratégia a ser aplicada, pois ela garante que todas as possibilidades sejam calculadas e, ao mesmo tempo, que nenhuma irá se repetir.

Capítulo 5. Resultados

Este capítulo apresenta os resultados obtidos nas simulações realizadas. As simulações – que o presente capítulo ressalva – serviram como método de validação à proposta sugerida neste trabalho.

Foram utilizadas as estruturas Trp-cage (código PDB 1L2Y, [30]) e a Mini-Proteína Estabilizada por Pontes Dissulfeto (código PDB 1ZDD, [36]), como base para os testes para o método de predição desenvolvido.

5.1 Simulações

Com intuito de validar a implementação, foram realizadas simulações, utilizando as proteínas Trp-Cage (código PDB 1L2Y) e Mini-Proteína Estabilizada por Pontes Dissulfeto (código PDB 1ZDD) como molde.

5.1.1 Simulação utilizando a proteína Trp-Cage

Esta simulação foi feita segundo o Algoritmo 1, com o cuidado de remover as estruturas idênticas – neste caso apenas a própria proteína Trp-Cage (código PDB 1L2Y) – e, também, apenas utilizar os primeiros cinco fragmentos que obtiveram o maior escore na busca do BLASTP (HSF – *High Scored Fragments*), ou seja, para cada busca feita no BLASTP apenas os cinco primeiros códigos PDBs foram utilizados. Salienta-se, no entanto, que a mini-proteína Trp-Cage se caracteriza por ser uma estrutura de apenas 20 aminoácidos o que ajuda para uma primeira simulação.

Ademais, com uma primeira simulação, sendo bem restrita, utilizando apenas os cinco HFS de cada pentapeptído, veio a gerar um número muito grande de estruturas, chegando, neste caso, a 20.774.983.680.000. Isto se mostra um problema uma vez que o tempo para a geração de cada estrutura deve ser levado em consideração (aproximadamente, um segundo e 50 centésimos de segundo) e, também, que o espaço para armazenamento das mesmas é restrito (cada estrutura gerada ocupa, aproximadamente, 27 Kb).

Para efetuar este trabalho, foram disponibilizados dois computadores biprocessados (dois processadores) e com disco rígido de 140 Gigabytes. Estes recursos técnicos possibilitaram serem feitas as simulações ora analisadas.

Com a finalidade de gerar os modelos de estruturas, foi utilizado o recurso técnico de dois servidores que, por meio da recursão, em um primeiro momento, variou os fragmentos finais para os iniciais (como apresentado no exemplo da Figura 14); em um segundo momento, a variação se deu dos fragmentos iniciais para os finais. Esta estratégia foi utilizada, pois foi observado que – utilizando o poder e armazenamento computacional disponível – não seria possível gerar todas estruturas sugeridas pelo modelo. Desta maneira ficou mais fácil observar as variações tendo em vista que não seria possível gerar todas estruturas.

A partir dos resultados da presente simulação, a qual gerou mais de 340.000 estruturas, conseguiu-se uma estrutura com uma melhor resolução do que a obtida na simulação manual e com o *fold* similar ao da estrutura base. Na simulação manual, obteve-se um RMSD de 4,1 Å (Figura 13); no entanto, na simulação computacional, conseguiu-se uma resolução de 3,7 Å (Figura 15).

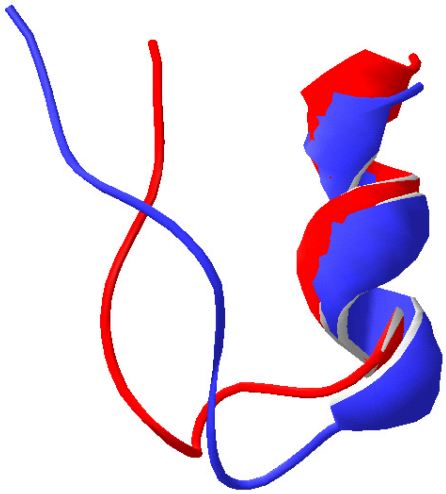


Figura 15: Superposição da estrutura 3D (topologia) experimental da proteína Trp-cage (código PBD 1L2Y, [30]), *ribbons* em vermelho, e da estrutura predita (em azul) com o método de simulação desenvolvido neste trabalho.

Com intuito de se obter uma melhor visão de como estavam sendo gerados os modelos e se os mesmos estavam convergindo para um melhor resultado (menor RMSD), foram gerados gráficos para análise.

Nas Figura 16 e Figura 17 podem ser observadas quatro análises sobre dados gerados, utilizando as recursões do final para o inicio (recursão normal ou *Backward*) e do início para o final (recursão inversa ou *Foreward*). No primeiro caso, recursão *Backward*, foram analisadas mais de 300.000 estruturas geradas, enquanto, no outro caso, foram apenas analisadas pouco mais de 43.000 estruturas.

Os gráficos, presentes nas figuras Figura 16 e Figura 17, apresentam uma análise ao método – aqui desenvolvido – sobre vários aspectos, sendo o principal deles a convergência. Com base no exposto, é possível ver quanto o método se aproxima ou se afasta da resposta ideal. Contudo, observa-se que na simulação da proteína Trp-Cage – partindo apenas de seus aminoácidos – a recursão *Foreward* (representado pela Figura 17), inicialmente, apresentou melhores resultados.

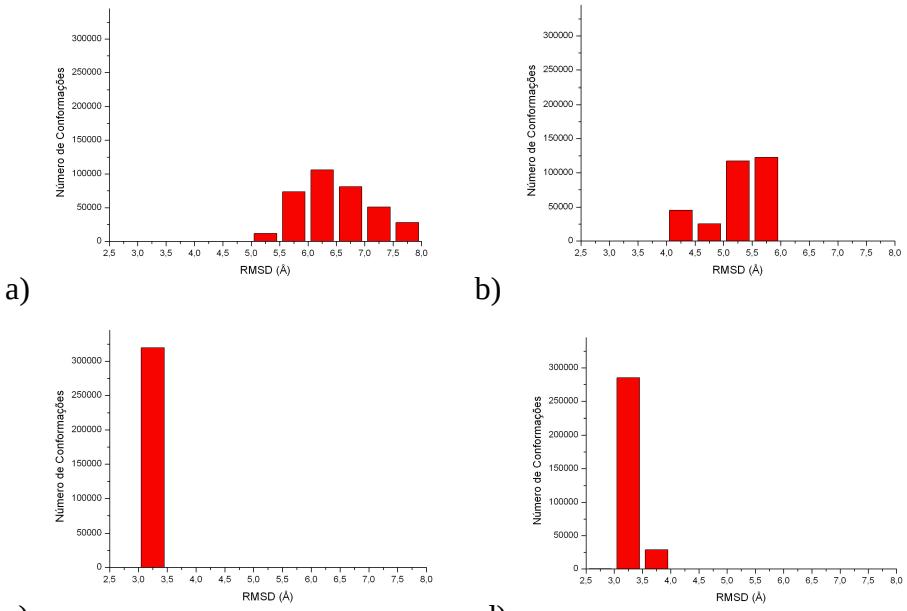


Figura 16: Análise do desvio médio quadrático (RMSD) da estrutura 3D da proteína Trp-Cage obtida pelo método de simulação proposto neste trabalho, utilizando a recursividade normal, com relação à estrutura experimental (código PDB 1L2Y). Mais de 300.000 conformações foram utilizadas nestas comparações e apenas os átomos da cadeia principal (N,Cα,C) foram utilizados no cálculo do RMSD. (a) RMSD da proteína Trp-cage completa (20 aminoácidos). (b) RMSD dos 16 aminoácidos centrais (da Y3 a P18). (c) RMSD da região N-terminal (aminoácidos Y3 a G10). (d) RMSD da região C-terminal (aminoácidos G11 a P18).

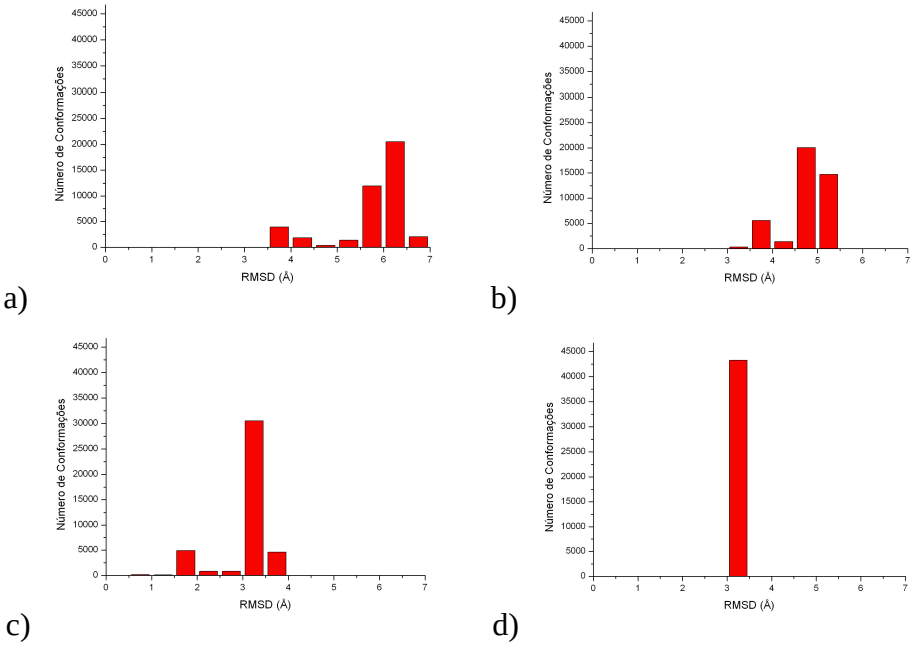


Figura 17: Análise do desvio médio quadrático (RMSD) da estrutura 3D da proteína Trp-Cage obtida pelo método de simulação proposto neste trabalho, utilizando a recursividade inversa, com relação à estrutura experimental (código PDB 1L2Y). Mais de 43.000 conformações foram utilizadas nestas comparações e apenas os átomos da cadeia principal (N,Cα,C) foram utilizados no cálculo do RMSD. (a) RMSD da proteína Trp-cage completa (20 aminoácidos). (b) RMSD dos 16 aminoácidos centrais (da Y3 a P18). (c) RMSD da região N-terminal (aminoácidos Y3 a G10). (d) RMSD da região C-terminal (aminoácidos G11 a P18).

Outro ponto de fácil observação perante Figura 16 e Figura 17, mais especificamente nos gráficos (c) e (d), é qual tipo de recursão foi adotada para obtenção das estruturas. Utilizando este recurso se torna possível uma análise na qual se mostra possível averiguar se o método desenvolvido está funcionando corretamente, uma vez que se espera que a variação ocorra da esquerda para a direita ou vice-versa (dependendo do tipo de recursão).

A apresentação do cálculo dos aminoácidos centrais – gráfico (b) das figuras Figura 16 e Figura 17 – mostrou-se muito relevante, pois o mesmo possui a característica de avaliar apenas os aminoácidos que realmente foram validados junto ao método, uma vez que o mesmo acaba por possuir uma menor precisão nos dois primeiros e últimos pares da seqüência.

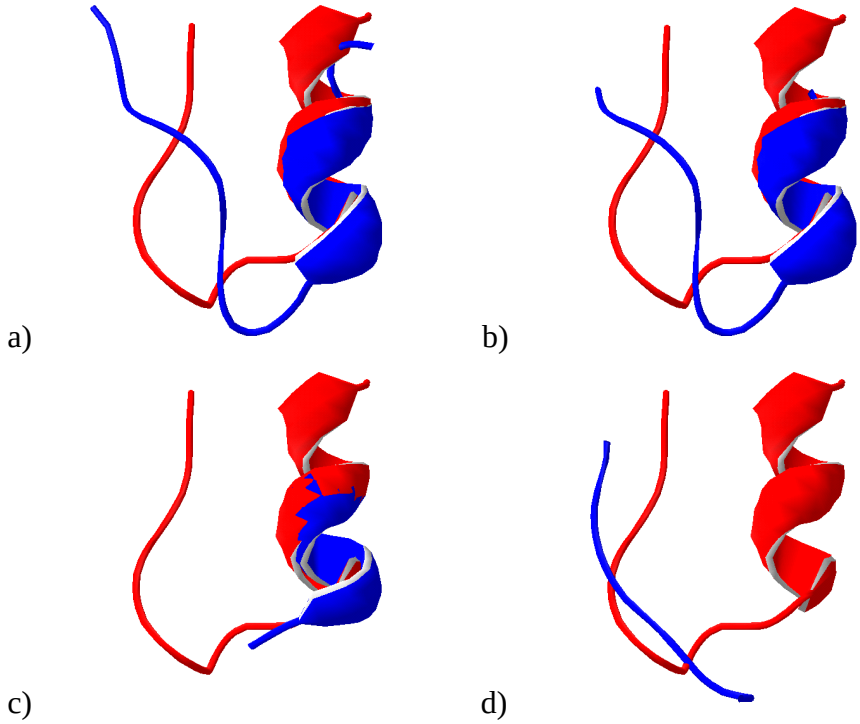


Figura 18: Superposição da estrutura 3D (topologia) experimental da proteína Trp-cage (código PBD 1L2Y, [30]), *ribbons* em vermelho, e da melhor estrutura predita (em azul) com o método de simulação desenvolvido neste trabalho. (A) RMSD de 3,7 Å da proteína Trp-cage completa (20 aminoácidos). (B) RMSD de 3,4 Å dos 16 aminoácidos centrais (da Y3 a P18). (C) RMSD de 0,9 Å da região N-terminal (aminoácidos Y3 a G10). (D) de 3,1 Å RMSD da região C-terminal (aminoácidos G11 a P18).

A Figura 18 serve como uma pequena exemplificação de qual parte da estrutura em análise foi comparada com a estrutura experimental Trp-Cage para a geração dos gráficos utilizados nas figuras Figura 16 e Figura 17.

A seguir, na Figura 19, é apresentado o Ramachandran Plot tanto da estrutura experimental Trp-Cage quanto da melhor estrutura gerada pelo método aqui apresentado. Em ambos os casos, é possível observar, que embora existam algumas pequenas diferenças em cada Ramachandran, os ângulos analisados se assemelham bastante, o que leva a deduzir que esta primeira simulação realmente gerou uma estrutura próxima a estrutura experimental.

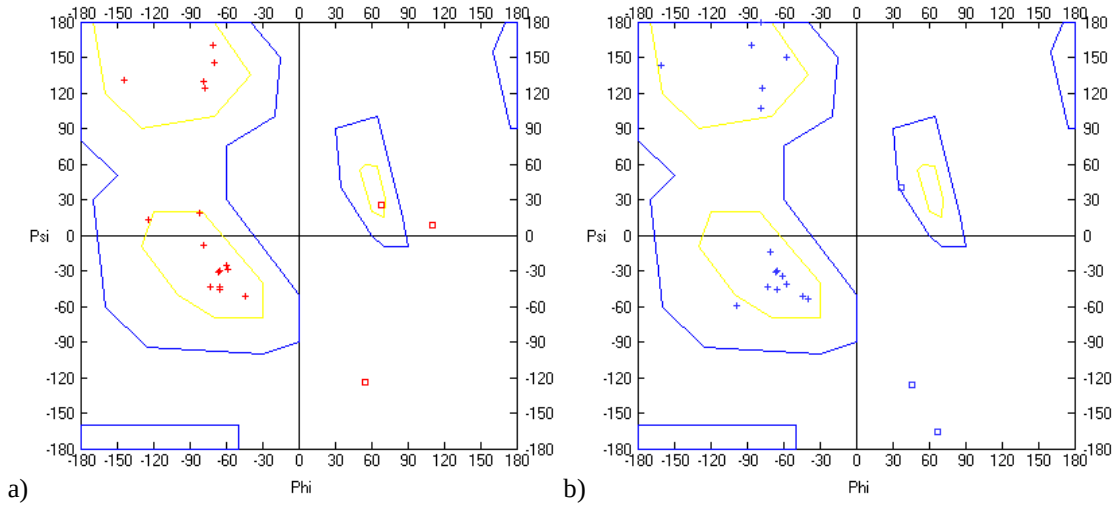


Figura 19: Gráficos de Ramachandran, os quais apresentam os ângulos φ (Phi) e ψ (Psi): (a) ângulos, da estrutura experimental da proteína Trp-cage (código PBD 1L2Y, [30]), demonstrados em vermelho; (b) ângulos da estrutura predita (ilustrados em azul) com o método de simulação desenvolvido neste trabalho.

5.1.2 Simulação utilizando a proteína 1ZDD

Tendo uma vez validado a metodologia proposta e a implementação, foram realizadas novas simulações. Devido a motivação adquirida pelos resultados obtidos nas simulações da Trp-Cage, decidiu-se investir em uma proteína ainda maior. A Mini-Proteína Estabilizada por Pontes Dissulfeto (código PDB 1ZDD), caracteriza-se por ser composto por 34 aminoácidos, assim aumentando a complexidade em relação a proteína anteriormente analisada, Trp-Cage, de 20 aminoácidos.

Antes da realização da busca, foram filtrados os moldes das estruturas idênticas à mini-proteína (utilizada como base), sendo os mesmos referentes aos seguintes códigos PDBs: 1ZDD, 1L6X, 1OQO, 1OQX, 1ZDA, 1ZDC e 1ZDB. Esta busca, também, caracterizou-se por usar apenas os primeiros cinco HSF. Mesmo realizando estas restrições, a busca pelo espaço conformacional resultou em 241.173.741.886.154.182.154.649.600.000 conformações.

Na presente simulação foram geradas pouco mais de 465.000 estruturas, ou seja, menos que 0,01% dos dados esperados e, como retorno, não se obtiveram estruturas com *folds* similares à estrutura utilizada como base, como pode ser observado na Figura 20.

A simulação da proteína 1ZDD se caracterizou por, nos melhores casos, apresentar um RMSD próximo a 7,5 Å. A melhor estrutura gerada se caracterizou por possuir exatamente 7,2 Å, como pode ser visto na Figura 20.

Figura 20: Superposição da estrutura 3D (topologia) experimental da proteína Mini-Proteína Estabilizada por Pontes Dissulfeto (código PBD 1ZDD, [36]), *ribbons* em vermelho, e da estrutura predita (em azul) com o método de simulação desenvolvido neste trabalho.

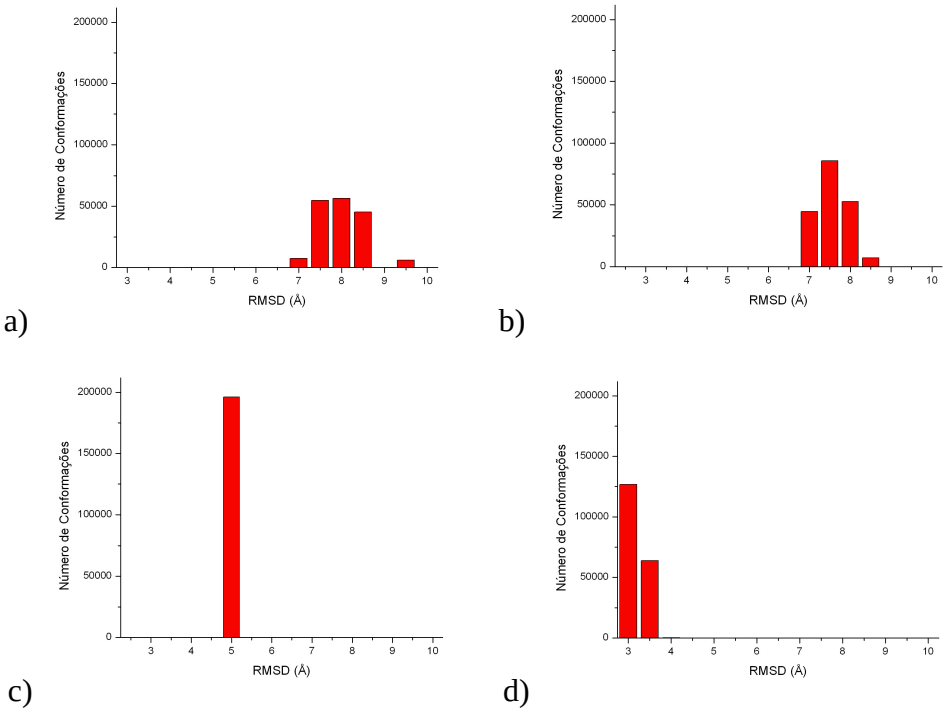


Figura 21: Análise do desvio médio quadrático (RMSD) da estrutura 3D Mini-Proteína Estabilizada por Pontes Dissulfeto obtida pelo método de simulação proposto neste trabalho, utilizando a recursividade normal, com relação à estrutura experimental (código PDB 1ZDD). Mais de 190.000 conformações foram utilizadas nestas comparações e apenas os átomos da cadeia principal (N,Cα,C) foram utilizados no cálculo do RMSD. (a) RMSD da proteína 1ZDD completa (34 aminoácidos). (b) RMSD dos 30 aminoácidos centrais (da M3 a D32). (c) RMSD da região N-terminal (aminoácidos M3 a N17). (d) RMSD da região C-terminal (aminoácidos L18 a D32).

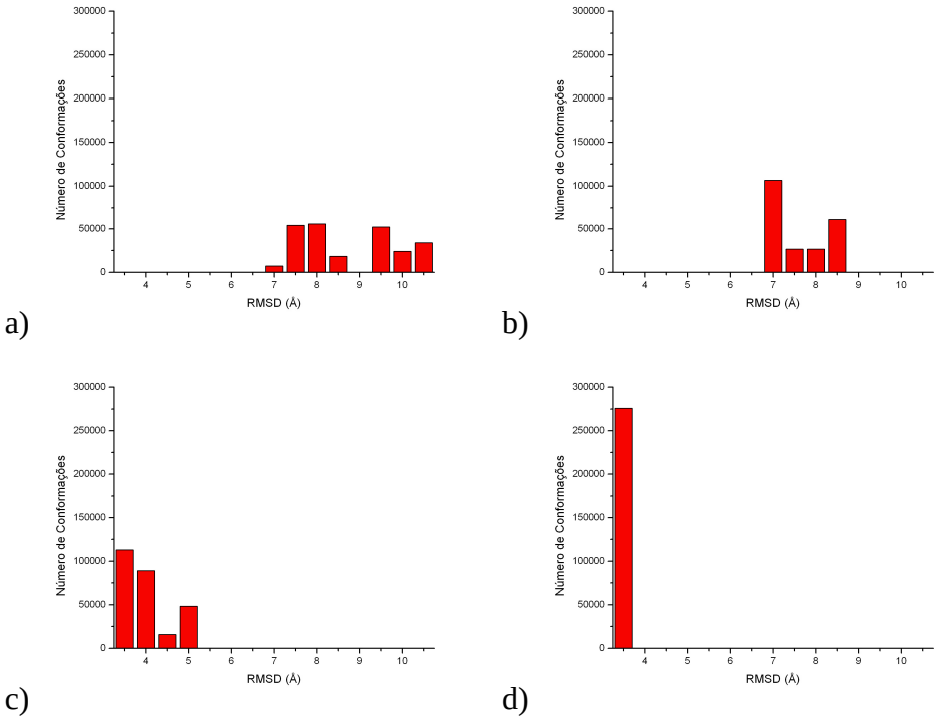
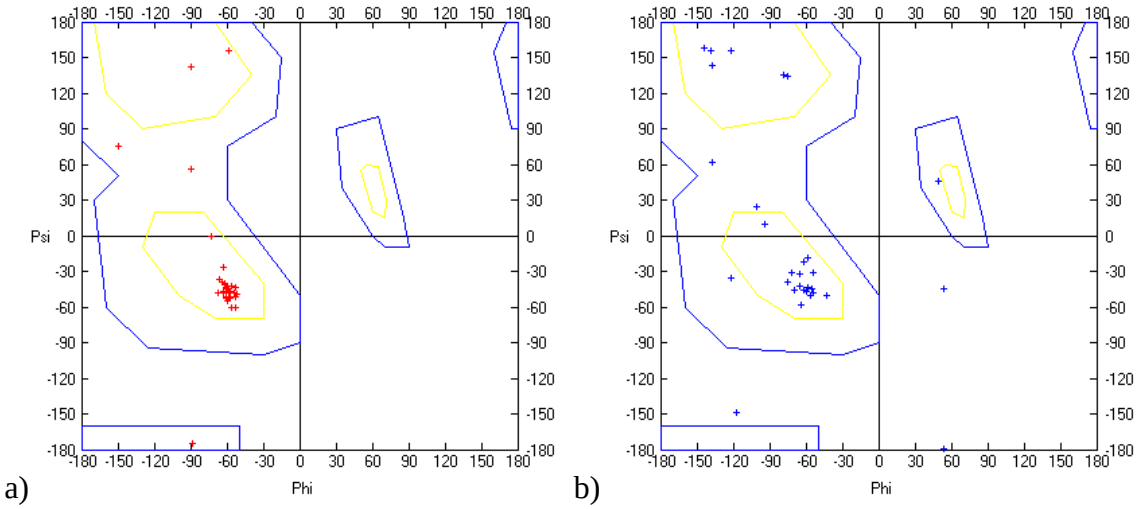


Figura 22: Análise do desvio médio quadrático (RMSD) da estrutura 3D Mini-Proteína Estabilizada por Pontes Dissulfeto obtida pelo método de simulação proposto neste trabalho, utilizando a recursividade inversa, com relação à estrutura experimental (código PDB 1ZDD). Pouco menos de 275.000 conformações foram utilizadas nestas comparações e apenas os átomos da cadeia principal (N,Cα,C) foram utilizados no cálculo do RMSD. (a) RMSD da proteína 1ZDD completa (34 aminoácidos). (b) RMSD dos 30 aminoácidos centrais (da M3 a D32). (c) RMSD da região N-terminal (aminoácidos M3 a N17). (d) RMSD da região C-terminal (aminoácidos L18 a D32).

Seguindo o padrão da análise realizada na subseção anterior, nas Figura 21 e Figura 22, também, podem ser observadas quatro análises sobre dados gerados, utilizando as recursões *Backward* e *Foreward*. No primeiro caso, recursão *Backward*, foram analisadas mais de 190.000 estruturas geradas, enquanto, no outro caso, analisadas pouco menos de 275.000 estruturas.

Já neste caso, observa-se que na simulação da proteína 1ZDD – partindo apenas de seus aminoácidos – utilizando a recursão *Backward* (representado pela Figura 22), apresentou a melhor variação em relação ao valor RMSD.

A Figura 23 se caracteriza por apresentar os Ramachandran Plot's referentes à estrutura experimental e à estrutura gerada com o melhor RMSD. Diferentemente do resultado obtido na simulação anterior, pode-se observar que os gráficos de Ramachandran, para a atual simulação, apresentaram valores para Phi e Psi bastante divergentes, comparado com o resultado obtido na simulação anterior, o que caracteriza que possivelmente o método – até então – não havia gerado nenhuma estrutura com os ângulos parecidos.



a) b)

Figura 23: Gráficos de Ramachandran, os quais apresentam os ângulos φ (Phi) e ψ (Psi): (a) ângulos, da estrutura experimental da proteína Mini-Proteína Estabilizada por Pontes dissulfeto (código PBD 1ZDD, [36]), demonstrados em vermelho; (b) ângulos da estrutura predita (ilustrados em azul) com o método de simulação desenvolvido neste trabalho.

5.2 Trabalhos Relacionados

Como explicado no Capítulo 3, há vários trabalhos em andamento no âmbito da predição da estrutura terciária de proteínas, utilizando como metodologia a modelagem comparativa por homologia. Esta metodologia funciona adequadamente ao proposto, mas possui dois empecilhos; o primeiro é que estruturas que não possuam uma identidade razoável com outras estruturas dos bancos de dados, não podem ser preditas; e, o outro, devido se basear, apenas, em estruturas já conhecidas, não permite que novos *folds* sejam encontrados.

Tendo em vista que a proposta feita se trata de um método *knowledge-based* e que possibilita o descobrimento de novos *folds*, a modelagem comparativa por homologia não se apresenta como trabalhos relacionados relevantes, pois há grandes diferenças.

A metodologia para predição da estrutura terciárias de proteínas mais relacionada com a proposta aqui apresentada seria a *ab initio*, pois, a mesma, além de possibilitar a descoberta de novos *folds*.

Na literatura são apresentados poucos programas que utilizam a metodologia *ab initio*. Tem-se como referência os programas: Touchstone [37,38] desenvolvido por Daisuke Kihara, Hui Lu, Andrzej Kolinski e Jeffrey Skolnick; Fragfold [39,40] desenvolvido por David Jones; Robetta[1] [41,3] desenvolvido por David Baker e Chris Bystroff.

A justificativa da pouca variedade de software encontrada sobre os métodos ab *initio* é conseqüência da falta de destaque dada, por muito tempo, deste método de predição. Isso se justifica, pois os modelos possuíam baixos índices de exatidão [18].

[1] O Robetta era anteriormente chamado de Rosetta e por isto suas referências fazem menção ao Rosetta.

A seguir, serão apresentadas algumas características dos *softwares* para predição *ab initio* citados acima.

1) Touchstone: Este *software* [37,38] representa a proteína conectando os vértices da cadeia em um reticulado, onde cada vértice se encontra no centro da massa de um resíduo de um dado carbono alfa.

Para melhorar a correlação da energia com a qualidade do enovelamento, são utilizadas tanto predições de estruturas secundárias, quanto ao contato da terciária. Para a introdução da seleção do enovelamento, são combinados os algoritmos de clustering de estrutura de Betancourt e Skolnick [42] com o conhecimento baseado no potencial do par de átomos pesados, para o procedimento de seleção, com intuito de selecionar estruturas representativas.

Na Figura 24 [38], pode ser observada uma comparação entre estruturas obtidas experimentalmente com estruturas obtidas por meio do Touchstone. Os experimentos obtidos pela ferramenta foram avaliados, utilizando o RMSD (medida que calcula o desvio médio quadrático) como medida de exatidão para os modelos gerados.

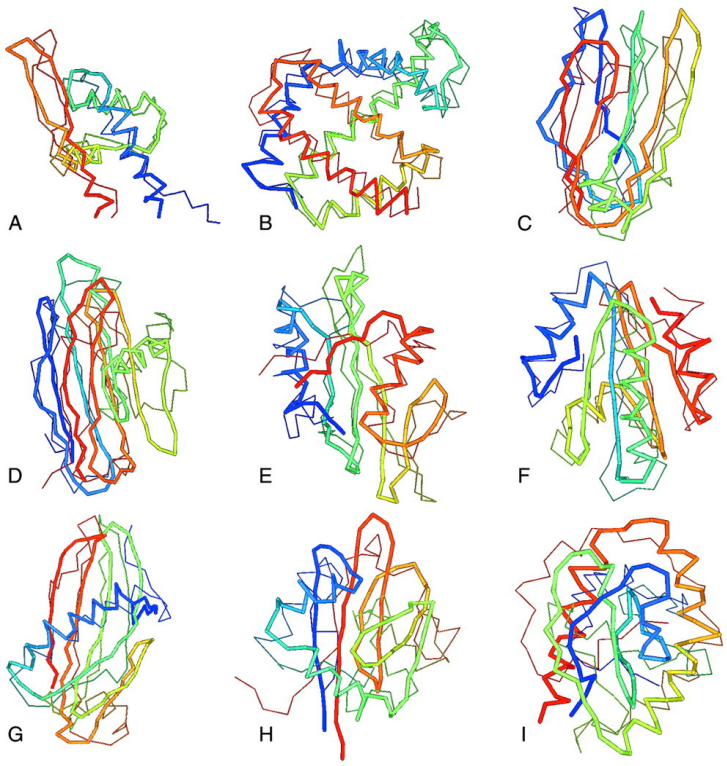


Figura 24: Superposição entre a estrutura gerada (linha grossa) e a estrutura obtida experimentalmente (linha fina). (A) rmsd 4.5 Å. (B) rmsd 2.7 Å. (C) rmsd 4.0 Å. (D) rmsd 4.5 Å. (E) rmsd 3.6 Å. (F) rmsd 2.3 Å. (G) rmsd 7.2 Å. (H) rmsd 8.7 Å. (I) rmsd 9.7 Å. Adaptado de Kihara *et al.* [38].

2) Fragfold: A maioria dos métodos bem sucedidos na predição de enovelamentos explora o fato de que, quando um novo enovelamento é descoberto, o mesmo

geralmente é composto por motivos estruturais já conhecidos no nível estrutural supersecundário.

O Fragfold, originalmente, explora este princípio [40]. Neste sentido, o método Fragfold restringe a busca no espaço conformacional, pré-selecionando fragmentos estruturais de uma biblioteca de estruturas de proteínas conhecidas.

A aproximação utilizada pelo método para predizer as estruturas secundárias deve avaliar o ajuste da seqüência nos enovelamentos conhecidos e selecionar o modelo que consegue a compatibilidade mais elevada com a seqüência linear da proteína.

O método se caracteriza por possuir dois tipos de potenciais de forças utilizados, sendo o primeiro dado pela análise estatística da estrutura de proteínas resolvidas experimentalmente e o segundo pela aplicação da equação de Boltzmann [43].

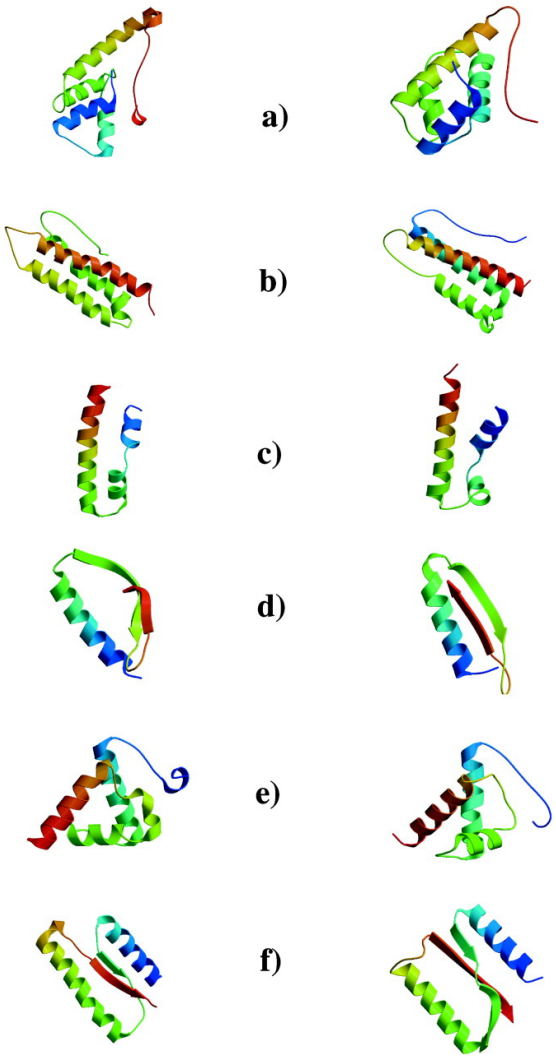


Figura 25: Estruturas preditas através de modelos experimentais, à esquerda, e obtidas através do Fragfold, à direita. Os resultados dos RMSD não foram divulgados. Adaptado de Jones *et al.* [40].

Ao final dos cálculos, é gerada uma estrutura onde são agregados todos os modelos de proteínas gerados. Assim, é possível identificar quais são os melhores modelos de proteínas gerados [40]. Na Figura 25 [40] estão representadas estruturas preditas pelo Fragfold durante o CASP 5.

3) Robetta: O *software* Robetta [41] é bastante conhecido na área de predição *ab initio* e o mesmo se caracteriza por ser a principal referência na área de predição. Grande parte dos artigos atuais sobre predição *ab initio* elegem Robetta como um método de grande destaque [17] e, ainda, o mesmo se caracteriza por possuir um bom desempenho no CASP [18]. Embora Robetta seja classificado como metodologia *ab initio*, o mesmo utiliza uma base de dados de estruturas conhecidas como o PDB, o que o caracteriza como metodologia *knowledge-based*.

O método de predição da estrutura de proteína, utilizado no Robetta, é baseado na suposição de que a distribuição das conformações disponíveis a cada segmento de três ou nove resíduos da cadeia, está razoavelmente próxima à distribuição das estruturas adotadas pela seqüência do segmento (e de seqüências altamente relacionadas) em estruturas de proteínas conhecidas. As bibliotecas do fragmento, para cada um dos segmentos de três ou nove resíduos da cadeia, são obtidas por meio de comparações de cada fragmento sobre uma base de dados com estruturas secundárias de proteínas [41]. As estruturas das proteínas são montadas randomicamente pela combinação dos fragmentos, utilizando o método de Monte Carlo. Para cada seqüência do alvo, um grande número de estruturas são geradas com este protocolo, e, então, “clusterizadas”; os cinco melhores clusters são escolhidos como as predições corretas.

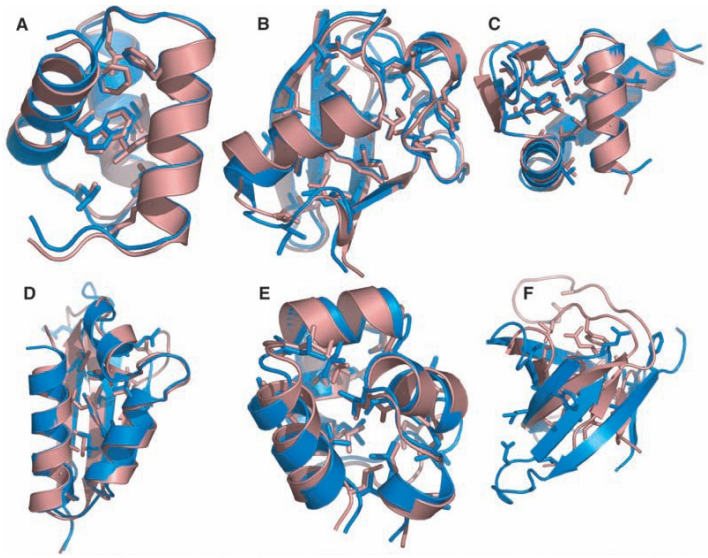


Figura 26: Superposição entre as estruturas obtidas experimentalmente (em vermelho), com as estruturas preditas pelo Robetta (em azul). É apresentado entre as estruturas A-E, proteínas preditas com o RMSD inferior à 1.5 Å, já a estrutura F mostra uma estrutura com RMSD 11.1 Å. Adaptado de Bradley et al. [44].

Robetta tem como objetivo alcançar um modelo final com alta resolução para a estrutura terciária de interesse [44].

Na Figura 26 [44], pode ser observada uma comparação entre estruturas experimentais e estruturas obtidas por meio do Robetta.

5.3 Considerações

O trabalho proposto se diferencia dos outros trabalhos, pois não são empregadas as estruturas terciárias ou fragmentos de estruturas secundárias – encontradas no banco de dados – para a construção do modelo, mas, sim, informação dos ângulos das estruturas protéicas 3D, como base para construção dos modelos.

Como explicado anteriormente, o trabalho aqui proposto utiliza fragmentos de cinco aminoácidos consecutivos de um dado polipeptídeo. Após, são feitas buscas para encontrar proteínas que contenham o mesmo pentapeptídeo.

A grande diferença entre a proposta desenvolvida e os outros métodos existentes é – enquanto o Fragfold e o Robetta buscam fragmentos de proteínas em banco de dados estruturais e utilizam a estrutura dos fragmentos encontrados para predizer os polipeptídeos – não utilizar as estruturas encontradas, mas, sim, gerar estruturas terciárias, utilizando apenas os ângulos φ, ψ e ω (do aminoácido central) de cada um dos pentapeptídeos encontrados, como apresentado na Figura 11.

Capítulo 6. Considerações Finais

Tendo em vista a problemática da explosão na quantidade de seqüências protéicas sem estrutura protéica conhecida, mostra-se necessária uma nova tecnologia de estudo e caracterização de estruturas protéicas, que seja capaz de inferir (com pequena margem de erro) a conformação 3D nativa de uma proteína, considerando como base apenas sua seqüência de aminoácidos.

Atualmente se mostra como tendência para predição de estruturas terciárias a utilização de metodologias computacionais. Ademais, não seria apenas um grande avanço para a bioinformática, mas também seria um grande progresso para a sociedade científica como um todo, já que ficaria mais fácil predizer estruturas de organismos de interesse econômico ou patogênico, bem como a inovação, a facilitação de processos de desenvolvimento de drogas para combater doenças.

O presente trabalho se mostrou um avanço na predição computacional de estruturas protéicas. Com base nos diversos trabalhos existentes, que mais se aproximam, observam-se combinações de fragmentos, armazenados em banco de dados de fragmentos tridimensionais A proposta, aqui, nesta dissertação desenvolvida, caracteriza-se por inovar a predição de polipeptídeos, utilizando apenas combinações dos ângulos de proteínas.

Verte-se que tanto na geração de estruturas, utilizando a combinação de fragmentos, quanto na combinação de ângulos, ambas se caracterizam por possuírem problemas. Nos dois casos, a maior dificuldade observada se dá pelo grande número de possibilidades de combinações para geração dos modelos de estruturas polipeptídicas. O método aqui discorrido (combinação de ângulos) se trata de uma modelagem pioneira na área, uma vez que é apresentada uma nova metodologia para predição de estruturas 3D. Assim, esta se apresenta, em uma primeira análise, uma metodologia promissora, acreditando-se, com isso, que, em trabalhos futuros, sejam possíveis otimizações do método (por exemplo, diminuir o número de HSF utilizado na busca) de maneira a cada vez aumentar a precisão com um menor esforço computacional, tanto em questão de espaço em disco quanto em tempo de processamento.

Embora as simulações realizadas tenham gerado um grande número de estruturas protéicas, não foi possível – em nenhuma das duas simulações – gerar o número de estruturas totais que o algoritmo se propõe a criar. A maior inviabilidade, do presente método, constitui-se na falta de espaço físico em disco rígido. Mesmo com os problemas assinalados, pode-se dizer que a simulação obtida – embora tenha sido uma pequena amostra do total de estruturas esperadas – apresentou-se eficiente. Assim, na simulação da Trp-Cage, foi possível chegar a um modelo com um *fold* similar ao da

proteína obtida experimentalmente, a qual foi usada como referência. No caso da 1ZDD, a busca não retornou nenhum resultado com *fold* similar ao da estrutura obtida experimentalmente, mas vale ressaltar que na simulação não foi percorrido nem 0,01% do total de estruturas a serem geradas, partindo apenas da seqüência de aminoácidos, como o modelo se propõe.

O atual trabalho tende a obter uma melhor predição em casos em que o número de aminoácidos é relativamente pequeno. Foi possível perceber, junto as simulações, que conforme o número de aminoácidos aumenta o número de estruturas a serem geradas, tende a aumentar, exponencialmente, e, com isto, interferindo diretamente tanto no número de estruturas a serem geradas quanto no espaço em disco a ser alocado.

Como alternativa aos problemas atualmente encontrados (muitas estruturas a serem geradas e muito espaço em disco a ser alocado), primeiramente, pensa-se na hipótese em ir removendo as estruturas que possuem o RMSD mais alto, aos fins de permitir sempre possuir dados para novas buscas, mantendo apenas os resultados com os menores RMSDs. Outra possibilidade seria analisar todos os ângulos que irão fazer parte da busca conformacional, agrupando os ângulos similares a partir de um desvio a ser determinado, conseqüentemente, diminuiria o número de estruturas a serem geradas implicando, também, em uma redução do espaço em disco a ser alocado.

Ressalva-se, no entanto, que não existe nenhum método computacional para predição de proteínas que consiga gerar estruturas com resolução similar as obtidas experimentalmente.

Referências

[1] BENSON, D. A.; KARSCH-MIZRACHI, I.; LIPMAN, D. J.; OSTELL, J.; RAPP, B. A.; WHEELER, D. L., GenBank, Nucleic Acids Research, v. 28, pp. 15-18, 2000.

[2] PROSDOCIMI, F.; CERQUEIRA, G. C.; BINNECK, E.; SILVA, A. F.; REIS, A. N.; JUNQUEIRA, A. C. M.; SANTOS, A. C. F.; NHANI JR, A.; WUST, C. I.; CAMARGO FILHO, F.; KESSEDJIAN, J. L.; PETRESKI, J. H.; CAMARGO, L. P.; FERREIRA, R. G. M.; LIMA, R. P.; PEREIRA, R. M.; JARDIM, S.; SAMPAIO, V. S.; FLATSCHART, Á. V. F., Bioinformática: Manual do Usuário, Biotecnologia Ciência e Desenvolvimento, v. 29, pp. 12-25, 2002.

[3] GIBAS, C.; JAMBECK, P., Developing Bioinformatics Computer Skills, O'Reilly, ISBN: 1565926641, 2001.

[4] WESTBROOK, J.; FENG, Z.; JAIN, S.; BHAT, T. N.; THANKI, N.; RAVICHANDRAN, V.; GITTITAND, G. L.; BLUHM, W.; WISSIG, H.; GREER, D. S.; BOURNE, P. S.; BERMAN, H. M., The protein databank: unifying the archive, Nucleic Acids Research, v. 30, pp. 245-248, 2002.

[5] VITKUP, D.; MELAMUD, E.; MOULT, J.; SANDER, C., Completeness in structural genomics, Nature Structural Biology, v.8, pp. 559-566, 2001.

[6] SCHONBRUN, J.; WEDEMEYER, W. J.; BAKER D., Protein structure prediction in 2002, In: Current Opinion in Structural Biology, v. 12, pp. 348-354, 2002.

[7] LEHNINGER, A. L.; NELSON, D. L.; COX, M. M., Princípios de bioquímica, editora Sarvier, São Paulo, ed. 2, 1995.

[8] HUNTER, L., Artificial Intelligence and Molecular Biology, L. Hunter (ed.), 1-46, AAAI Press, Menlo Park, CA, 1993.

[9] LESK, A. M., Introduction to protein architecture: the structural biology of proteins, New York, Oxford University Press, 2001.

[10] PAULING, L.; COREY, R. B., Atomic Coordinates and Structure Factors for Two Helical Configurations of Polypeptide Chains, In: Proceedings of the National Academy of Sciences, v. 37, pp. 235-240, 1951.

[11] PAULING, L.; COREY, R. B., The Structure of Synthetic Polypeptides, In: Proceedings of the National Academy of Sciences, v. 37, pp. 241-250, 1951.

[12] LESK, A. M., Introduction to Bioinformatics, Oxford University Press, 2002.

[13] ANFINSEN, C. B., Principles that govern the folding of protein chains, Science, v. 181, pp. 223-230, 1973.

[14] DEEP VIEW - SWISS-PDBVIEWER. Disponível em http://au.expasy.org/spdbv/ Acessado em abril 2005

[15] RAMACHANDRAN, G. N.; SASISEKHARAN, V., Conformation of polypeptides and proteins, In: Advances in Protein Chemistry, v. 23, pp. 283–438, 1968.

[16] VENDRUSCOLO, M.; PACI, E., Protein folding: bringing theory and experiment closer together, In: Current Opinion in Structural Biology, v.13, pp. 82-87, 2003.

[17] OSGUTHORPE, D. J., Ab initio protein folding, In: Current Opinion in Structural Biology, v. 10, pp. 146-152, 2000.

[18] MOULT, J., A decade of CASP: progress, bottlenecks and prognosis in protein structure prediction, In: Current Opinion in Structural Biology, v. 15, pp. 285-289 2005.

[19] MARTI-RENOM, M. A.; STUART, A. C.; FISER, A.; SANCHEZ, R.; MELO, F.; SALI, A., Comparative protein structure modeling of genes and genomes, In: Annual Review of Biophysics and Biomolecular Structure, v. 29, pp. 291–325, 2000.

[20] MURZIN, A. G.; BRENNER, S. E.; HUBBARD, T.; CHOTHIA, C., SCOP: a structural classification of proteins database for the investigation of sequences and structures, Journal of Molecular Biology, v. 247, pp. 536-540, 1995.

[21] BRUNETTE, T. J.; BROCK, O., Improving protein structure prediction with model-based search, Bioinformatics, v. 21, pp. 66-74, 2005.

[22] HARDIN, C.; POGORELOV, T. V.; LUTHEY-SCHULTEN, Z., Ab initio protein structure prediction, In: Current Opinion in Structural Biology, v. 12, pp. 176-181, 2002.

[23] OŁDZIEJ, S.; CZAPLEWSKI, C.; LIWO, A.; CHINCHIO, M.; NANIAS, M.; VILA, J. A.; KHALILI, M.; ARNAUTOVA, Y. A.; JAGIELSKA, A.; MAKOWSKI, M.; SCHAFROTH, H. D.; KAZ´MIERKIEWICZ, R.; RIPOLL, D. R.; PILLARDY, J.; SAUNDERS, J. A.; KANG, Y. K.; GIBSON, K. D.; SCHERAGA, H. A., Physics-based protein-structure prediction using a hierarchical protocol based on the UNRES force field: Assessment in two blind tests, In: Proceedings of the National Academy of Sciences, v. 102, pp. 7547-7552, 2005.

[24] VENCLOVAS, C.; ZEMLA, A.; FIDELIS, K.; MOULT, J., Assessment of progress over the CASP experiments, Proteins, v. 53, pp. 585-95, 2003.

[25] BERMAN, H. M.; WESTBROOK, J.; FENG, Z.; GILLILAND, G.; BHAT, T. N.; WEISSIG, H.; SHINDYALOV, I. N.; BOURNE, P. E., The Protein Data Bank, Nucleic Acids Research, 28, pp. 235-242, 2000.

[26] BLUNDELL, T. L.; MIZUGUCHI, K., Structural genomics: an overview, In: Progress in Biophysics and Molecular Biology, v. 73, pp. 289-295, 2000.

[27] GINALSKI, K.; GRISHIN, N. V.; GODZIK, A.; RYCHLEWSKI, L., Practical lessons from protein structure prediction, Nucleic Acids Research, v. 33, pp. 1874-1891, 2005.

[28] HOVMOLLER, S.; ZHOU, T.; OHLSON, T., Conformations of amino acids in proteins, Acta crystallographica Section D Biological crystallography, v. 58, pp. 768-776, 2002.

[29] ALTSCHUL, S. F.; GISH, W.; MILLER, W.; MYERS, E. W.; LIPMAN, D. J., Basic local alignment search tool, Journal of Molecular Biology, v. 215, 403-410, 1990.

[30] NEIDIGH, J. W.; FESINMEYER, R. M.; ANDERSEN, N. H., Designing a 20-residue protein, Nature Structural Biology, v. 9, pp. 425-430, 2002

[31] BROWN, M. C., Python: The Complete Reference, McGraw-Hill, ISBN: 007212718X, 2001.

[32] CHAPMAN, B.; CHANG, J., Biopython: Python tools for computational biology, In: ACM SIGBIO Newsletter, v. 20, pp. 15-19, 2000.

[33] CASE, D. A.; Cheatham, T. E.; Darden, T.; Gohlke, H.; Luo, R.; Merz Jr., K. M.; Onufriev, A.; Simmerling, C.; Wang, B.; Woods, R., The Amber biomolecular simulation programs, Journal of Computational Chemistry, v. 26, pp. 1668-1688 2005.

[34] SAMUDRALA, R.; XIA, Y.; HUANG, E.; LEVITT, M., Ab initio protein structure prediction using a combined hierarchical approach, Proteins, v. 37, pp. 194–204, 1999.

[35] HAMELRYCK, T.; MANDERICK, B., PDB file parser and structure class implemented in Python, Bioinformatics, v.19, pp. 2308-2310, 2003.

[36] STAROVASNIK, M. A.; BRAISTED, A. C.; WELLS, J. A., Structural mimicry of a native protein by a minimized binding domain, In: Proceedings of the National Academy of Sciences, v. 94, pp. 10080-10085, 1997

[37] SKOLNICK, J.; ZHANG, Y.; ARAKAKI, A. K.; KOLINSKI, A.; BONIECKI, M.; SZILAGYI, A.; KIHARA, D., TOUCHSTONE: a unified approach to protein structure prediction, Proteins, v. 53, pp. 469–479, 2003.

[38] KIHARA, D.; LU, H.; KOLINSKI, A.; SKOLNICK, J., TOUCHSTONE: An ab initio protein structure prediction method that uses threading-based tertiary restraints, In: Proceedings of the National Academy of Sciences, v. 98, pp. 10125-10130, 2001.

[39] YUAN, X.; SHAO, Y.; BYSTROFF C., Ab initio protein structure prediction using pathway models, Comparative and Functional Genomics, v. 4, pp. 397-401, 2003.

[40] JONES, D. T.; MCGUFFIN, L. J., Assembling Novel Protein Folds from Super-secondary Structural Fragments, Proteins: Structure, Function and Genetics, v. 53, pp. 480-485, 2003.

[41] BRADLEY, P.; CHIVIAN, D.; MEILER, J.; MISURA, K. M.; ROHL, C. A.; SCHIEF, W. R.; WEDEMEYER, W. J.; SCHUELER-FURMAN, O.; MURPHY, P.; SCHONBRUN, J.; STRAUSS, C. E.; BAKER, D., Rosetta predictions in CASP5: successes, failures, and prospects for complete automation, Proteins, v. 53, pp. 457–468, 2003.

[42] BETANCOURT, M. R.; SKOLNICK, J., Finding the Needle in a Haystack: Educing native Folds from Ambiguous Ab Initio Protein Structure Predictions, Journal of Computational Chemistry, v. 22, pp. 339-353, 2001.

[43] JONES, D. T., Predicting novel protein folds by using FRAGFOLD, Proteins, v. 5, pp. 127-132, 2001.

[44] BRADLEY, P.; MISURA, K. M.; BAKER, D., Toward high-resolution de novo structure prediction for small proteins, Science, v. 309, pp. 1868-1871, 2005.

Apêndice A: Código-fonte referente à implementação

```
def rodablast(sequence):
    fragmentos = []
    a = len(sequence)-4
    for x in range(a):
        print sequence[x:x+5]
        fragmentos.append(sequence[x:x+5])
    print fragmentos


def qblast(program, database, sequence,
           ncbi_gi=None, descriptions="1000", alignments=None,
           expect=None, matrix=None,
           filter=None, format_type="XML", hitlist_size=None,
           entrez_query='(none)',
           ):
    import urllib

    parameters = [
        ('QUERY', sequence),
        ('PROGRAM', program),
        ('ENTREZ_QUERY', entrez_query),
        ('DATABASE', database),
        ('COMPOSITION_BASED_STATISTICS','0'),
        ('EXPECT', '20000'),
        ('WORD_SIZE' , '2'),
        ('HITLIST_SIZE', hitlist_size),
        ('MATRIX_NAME', 'PAM30'),
        ('GAPCOSTS', '9+1'),
        ('I_THRESH' , '0.005'),
        ('FILTER', filter),
        ('CMD', 'Put'),
        ]
    query = [x for x in parameters if x[1] is not None]
    message = urllib.urlencode(query)

    handle = _send_to_qblast(message)

    rid, rtoe = _parse_qblast_ref_page(handle)
    parameters = [
        ("RID", rid),
        ('DESCRIPTIONS', '1000'),
        ('ALIGNMENTS', '50'),
        ('NCBI_GI', ncbi_gi),
        ('FORMAT_TYPE', format_type),
        ("CMD", "Get"),
        ]
    query = [x for x in parameters if x[1] is not None]
    message = urllib.urlencode(query)

    limiter = RequestLimiter(3)
    while 1:
        limiter.wait()
```

```
        handle = _send_to_qblast(message)
        results = handle.read()

        if results.find("Status=") < 0:
            break
        i = results.index("Status=")
        j = results.index("\n", i)
        status = results[i+len("Status="):j].strip()
        if status.upper() == "READY":
            break

    i = results.index("Connection: close")
    i = results.index("\n",i)
    i = results.index("\n",i+1)
    results = results[i+1:]

    return StringIO.StringIO(results)

def _send_to_qblast(query):
    """Send a BLAST request to the QBLAST server at NCBI.
    http://www.ncbi.nlm.nih.gov/BLAST/Doc/urlapi.html

    Return a handle to the results.
    """
    import socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect(('www.ncbi.nlm.nih.gov', 80))

    sock.send('POST /blast/Blast.cgi HTTP/1.0\n')
    sock.send('User-Agent: BiopythonClient\n')
    sock.send('Connection: Keep-Alive\n')
    sock.send('Content-type: application/x-www-form-urlencoded\n')
    sock.send('Content-Length: %d\n' % len(query))
    sock.send('\n')
    sock.send(query)

    handle = StringIO.StringIO()
    while 1:
        data = sock.recv(1024)
        if not data:
            break
        handle.write(data)
    sock.close()

    handle.seek(0)
    return handle

def _parse_qblast_ref_page(handle):
    """Return tuple of RID, RTOE."""
    s = handle.read()
    i = s.find("RID =")
    j = s.find("\n", i)
    rid = s[i+len("RID ="):j].strip()

    i = s.find("RTOE =")
    j = s.find("\n", i)
    rtoe = s[i+len("RTOE ="):j].strip()
    return rid, int(rtoe)

def fragmenta(sequencia):
    seq = []
    for x in range(0,len(sequencia)-4):
        seq.append(sequencia[x:x+5])
    return seq

def node_text(node):
    text = ''
    for child in node.childNodes:
```

```
        if child.nodeType is child.TEXT_NODE:
            text += child.data
        return text

def analyseblast(fileblast):
    """ Funcao que analisa todo o arquivo blast e retorna todos os
     arquivos pdbs que possuem relacao entre o fragmento procurado
     e a busca realisada segundo o Blast."""
    x = xml.dom.minidom.parse(fileblast)
    nos = x.documentElement
    elementos = []
    tupla = []
    filhos1 = [no for no in nos.childNodes if no.nodeType == \
                  x.ELEMENT_NODE]
    for pai in filhos1:
        filhos2 = [no for no in pai.childNodes if no.nodeType == \
                      x.ELEMENT_NODE]
        for filho in filhos2:
            filhos3 = [no for no in filho.childNodes if no.nodeType == \
                          x.ELEMENT_NODE]
            for neto in filhos3:
                filhos4 = [no for no in neto.childNodes if no.nodeType == \
                          x.ELEMENT_NODE]
                for gera5 in filhos4:
                    filhos5 = [no for no in gera5.childNodes if no.nodeType ==
\
                              x.ELEMENT_NODE]
                    for gera6 in filhos5:
                        if gera6.nodeName == "Hit_num":
                            hitnum = int(node_text(gera6))
                        elif gera6.nodeName == "Hit_id":
                            hit_id = str(node_text(gera6))
                        elif gera6.nodeName == "Hit_def":
                            hitdef = str(node_text(gera6))
                        filhos6   =   [no   for   no   in   gera6.childNodes   if
no.nodeType == \
                                  x.ELEMENT_NODE]
                        for gera7 in filhos6:
                            filhos7  =  [no  for  no  in  gera7.childNodes  if
no.nodeType == \
                                      x.ELEMENT_NODE]
                            for gera8 in filhos7:
                                if gera8.nodeName == "Hsp_qseq":
                                    hspqseq = str(node_text(gera8))
                                elif gera8.nodeName == "Hsp_hseq":
                                    hsphseq = str(node_text(gera8))
                                elif gera8.nodeName == "Hsp_midline":
                                    hspmidline = str(node_text(gera8))
                                    elementos.append([hitnum, hit_id, hspqseq,
hsphseq, hspmidline])
                                    xxd = hit_id.find("|pdb|")
                                    tupla.append([hit_id[xxd+5:xxd+9],
hsphseq, hspqseq, hspmidline])
                                    xxd = 0
                                    while xxd != (-1):
                                        if (hitdef.find("|pdb|", xxd+9)) != (-
1):

                                            xxd = hitdef.find("|pdb|", xxd+9)
                                            tupla.append([hitdef[xxd+5:xxd+9],
hsphseq, hspqseq, hspmidline])
                                        else:
                                            xxd = (-1)
    return tupla

def removesimilar(sequence):
    seq=len(sequence)
    x=0
    while x < seq:
```

```
        y=x+1
        while y < seq:
            if sequence[x][0]==sequence[y][0]:
                if sequence[x][1]==sequence[y][1]:
                    sequence.pop(y)
                    seq=seq-1
                    y=y-1
            y=y+1
        x=x+1

def listadePDBs(sequence):
    # gera a lista de todos os PDBs necessarios para o Download
    # o download da lista pode ser feito utilizando a função "downloadDosPDBs"
    seq=len(sequence)
    lista=[]
    x=0
    while x < seq:
        x1=0
        while x1 < len(sequence[x]):
            lista.append(sequence[x][x1][0])
            x1=x1+1
        x=x+1
        removeigual(lista)
    return lista

def removeigual(sequence):
    # utilizado no listadePDBs para remover as sequencias iguais
    seq=len(sequence)
    x=0
    while x<seq:
        y=x+1
        while y<seq:
            if sequence[x]==sequence[y]:
                sequence.pop(y)
                seq=seq-1
            y=y+1
        x=x+1

def downloadDosPDBs(lista):
    for x in range(0,len(lista)):
        try:
            downloadPDB(lista[x])
        except:
            print "Problema ao baixar o arquivo: "+lista[x]
        print "Ja baixou "+str(x+1)+" de "+str(len(lista))

def downPDBs(sequence):
    seq=len(sequence)
    for x in range(0,seq):
        downloadPDB(sequence[x][0])

def downPDBs2(sequence):
    seq=len(sequence)
    for x in range(0,seq):
        seq1=len(sequence[x])
        for y in range(0,seq1):
            downloadPDB(sequence[x][y][0])

def downloadPDB(protein):
    save_file = open('PDB/'+protein+'.pdb', 'w')
    arquivo                                                                 =
urllib2.urlopen('http://www.rcsb.org/pdb/downloadFile.do?fileFormat=pdb&compre
ssion=NO&structureId='+protein).read()
    save_file.write(arquivo)
    save_file.close()


def generatePhiPsi(filename):
```

```
    tam = len(filename)
    filename2 = filename[:(tam-4)]
    os.system('./torsions PDB/'+filename+' > PDB/'+filename2+'.txt')

def analyseSeq(sequence, frag):
    t=[]
    seq=len(sequence)
    fraglen = int(len(frag))
    center=frag[int(fraglen/2)]
    for x in range(0,seq):
        if sequence[x][3]==frag:
            generatePhiPsi(sequence[x][0]+'.pdb')
            t.extend(analysePhiPsi('PDB/'+sequence[x][0]+'.txt',          frag,
(int(len(frag)/2)+1)))
        elif len(frag)==len(sequence[x][1]):
            generatePhiPsi(sequence[x][0]+'.pdb')
            t.extend(analysePhiPsi('PDB/'+sequence[x][0]+'.txt',
sequence[x][1], (int(len(frag)/2)+1)))
        elif sequence[x][2]==frag[:4]:
            generatePhiPsi(sequence[x][0]+'.pdb')
            t.extend(analysePhiPsi('PDB/'+sequence[x][0]+'.txt',
sequence[x][1], (int(len(frag)/2)+1)))
        elif sequence[x][2]==frag[1:]:
            generatePhiPsi(sequence[x][0]+'.pdb')
            t.extend(analysePhiPsi('PDB/'+sequence[x][0]+'.txt',
sequence[x][1], (int(len(frag)/2)+1)))
    return t

def analyseSeqNew(sequence, amino):
    t=[]
    y=[]
    d=0
    frag = fragmenta(amino)
#    print frag
    seq=len(sequence)
    print seq
    for l in range(0,seq):
        y=[]
        seq1=len(sequence[l])
        for x in range(0,seq1):
            if sequence[l][x][3]==frag[l]:
                generatePhiPsi(sequence[l][x][0]+'.pdb')
                y.extend(analysePhiPsi('PDB/'+sequence[l][x][0]+'.txt',
frag[l], (int(len(frag[l])/2)+1)))
#                print str(center)+' - '+str(sequence[x][0])+' - '+frag+' -
'+str(int(len(frag)/2)+1)+' - '+str(t)
            elif len(frag[l])==len(sequence[l][x][1]):
                generatePhiPsi(sequence[l][x][0]+'.pdb')
                y.extend(analysePhiPsi('PDB/'+sequence[l][x][0]+'.txt',
sequence[l][x][1], (int(len(frag[l])/2)+1)))
#                 print str(center)+' - '+str(sequence[x][0])+' - '+frag+' -
'+str(int(len(frag)/2)+1)+' - '+str(t)
            elif sequence[l][x][2]==frag[l][:4]:
                generatePhiPsi(sequence[l][x][0]+'.pdb')
                y.extend(analysePhiPsi('PDB/'+sequence[l][x][0]+'.txt',
sequence[l][x][1], (int(len(frag[l])/2)+1)))
            elif sequence[l][x][2]==frag[l][1:]:
                generatePhiPsi(sequence[l][x][0]+'.pdb')
                d=d+1
#                 print d
                y.extend(analysePhiPsi('PDB/'+sequence[l][x][0]+'.txt',
sequence[l][x][1], (int(len(frag[l])/2)+1)))
        if t!=y:
            t.append(y)
    return t


def analyseSeqAmino(sequence, amino):
```

```
# Mesmo "analyseSeqNew" so que este apresenta também os arq pdbs
    t=[]
    y=[]
    d=0
    frag = fragmenta(amino)
#    print frag
    seq=len(sequence)
    print seq
    part=0
    for l in range(0,seq):
        y=[]
        seq1=len(sequence[l])
        k=1
        part=part+1
        for x in range(0,seq1):
            if sequence[l][x][3]==frag[l]:
                generatePhiPsi(sequence[l][x][0]+'.pdb')
                hehe = analysePhiPsi('PDB/'+sequence[l][x][0]+'.txt', frag[l],
(int(len(frag[l])/2)+1))
                print hehe
                for x in range(len(hehe)):
                    hehe[x].extend([part,k])
                print hehe
                k=k+1
                y.extend(hehe)
#                  print str(center)+' - '+str(sequence[x][0])+' - '+frag+' -
'+str(int(len(frag)/2)+1)+' - '+str(t)
            elif len(frag[l])==len(sequence[l][x][1]):
                generatePhiPsi(sequence[l][x][0]+'.pdb')
                hehe      =      analysePhiPsi('PDB/'+sequence[l][x][0]+'.txt',
sequence[l][x][1], (int(len(frag[l])/2)+1))
                print hehe
                for x in range(len(hehe)):
                    hehe[x].extend([part,k])
                print hehe
                k=k+1
                y.extend(hehe)
#                   print str(center)+' - '+str(sequence[x][0])+' - '+frag+' -
'+str(int(len(frag)/2)+1)+' - '+str(t)
            elif sequence[l][x][2]==frag[l][:4]:
                generatePhiPsi(sequence[l][x][0]+'.pdb')
                hehe      =      analysePhiPsi('PDB/'+sequence[l][x][0]+'.txt',
sequence[l][x][1], (int(len(frag[l])/2)+1))
                print hehe
                for x in range(len(hehe)):
                    hehe[x].extend([part,k])
                print hehe
                k=k+1
                y.extend(hehe)
            elif sequence[l][x][2]==frag[l][1:]:
                generatePhiPsi(sequence[l][x][0]+'.pdb')
                d=d+1
#                 print d
                hehe      =      analysePhiPsi('PDB/'+sequence[l][x][0]+'.txt',
sequence[l][x][1], (int(len(frag[l])/2)+1))
                print hehe
                for x in range(len(hehe)):
                    hehe[x].extend([part,k])
                print hehe
                k=k+1
                y.extend(hehe)
        if t!=y:
            t.append(y)
    return t


def generateStructures(sequence, aminoacids):
```

```
#  Faz a busca no arquivo gerado pelo analyseSeqNew (phi's e psi's) e retorna
as
#  estruturas PDB referente a combinação dos mesmos.
#
    u=2
    aminos = []
    size = len(aminoacids)
    for x in range(size):
        amino = aminoacids[x]
        if amino == 'A':
            aminos.append('ALA')
        elif amino == 'C':
            aminos.append('CYS')
        elif amino == 'D':
            aminos.append('ASP')
        elif amino == 'E':
            aminos.append('GLU')
        elif amino == 'F':
            aminos.append('PHE')
        elif amino == 'G':
            aminos.append('GLY')
        elif amino == 'H':
            aminos.append('HIS')
        elif amino == 'I':
            aminos.append('ILE')
        elif amino == 'K':
            aminos.append('LYS')
        elif amino == 'L':
            aminos.append('LEU')
        elif amino == 'M':
            aminos.append('MET')
        elif amino == 'N':
            aminos.append('ASN')
        elif amino == 'P':
            aminos.append('PRO')
        elif amino == 'Q':
            aminos.append('GLN')
        elif amino == 'R':
            aminos.append('ARG')
        elif amino == 'S':
            aminos.append('SER')
        elif amino == 'T':
            aminos.append('THR')
        elif amino == 'V':
            aminos.append('VAL')
        elif amino == 'W':
            aminos.append('TRP')
        elif amino == 'Y':
            aminos.append('TYR')
        elif amino == 'E':
            aminos.append('PCA')
        elif amino == 'B':
            aminos.append('ASX')
        elif amino == 'Z':
            aminos.append('GLX')
        elif amino == 'X':
            aminos.append('UNK')
    aminos[0]='N'+aminos[0]
    aminos[size-1]='C'+aminos[size-1]
#    print aminos
    arq=[]
#    print sequence
#    recursiveSearch(0, sequence, arq, aminos)
    recursiveSearch2(len(sequence)-1, sequence, arq, aminos)


kou=0
```

```
def recursiveSearch(number, sequence, arq, aminos):
#  Chamada do generateStructures que faz a recursão responsável pela
#  geração da ordem em que os phi e psi serão gerados.
#
#    len(sequence)):
#        print x
        global kou
#        for y in range(1):
        for y in range(len(sequence[number])):
            if number+1==len(sequence):
                if len(arq)>number:
                   arq=arq[:number]
#                print str(len(arq))+'  '+str(number)
                arq.append([sequence[number][y][2],     sequence[number][y][3],
sequence[number][y][1]])
#                print len(arq)
#     print arq
                generateFilePDB(arq, kou, aminos)
                kou=kou+1
            else:
             if len(arq)>number:
#                  print len(arq)
#                  print 'foi '+str(kou)
#                if len(arq)>number:
                   arq=arq[:number]
#                print str(len(arq))+'  '+str(number)
                arq.append([sequence[number][y][2],     sequence[number][y][3],
sequence[number][y][1]])
             recursiveSearch(number+1, sequence, arq, aminos)
#        print y

def generateStructuresLines(sequence, aminoacids):
#  Faz a busca no arquivo gerado pelo analyseSeqNew (phi's e psi's) e retorna
as
#  estruturas PDB referente a combinação dos mesmos.
#
    u=2
    aminos = []
    size = len(aminoacids)
    for x in range(size):
        amino = aminoacids[x]
        if amino == 'A':
            aminos.append('ALA')
        elif amino == 'C':
            aminos.append('CYS')
        elif amino == 'D':
            aminos.append('ASP')
        elif amino == 'E':
            aminos.append('GLU')
        elif amino == 'F':
            aminos.append('PHE')
        elif amino == 'G':
            aminos.append('GLY')
        elif amino == 'H':
            aminos.append('HIS')
        elif amino == 'I':
            aminos.append('ILE')
        elif amino == 'K':
            aminos.append('LYS')
        elif amino == 'L':
            aminos.append('LEU')
        elif amino == 'M':
            aminos.append('MET')
        elif amino == 'N':
            aminos.append('ASN')
        elif amino == 'P':
            aminos.append('PRO')
        elif amino == 'Q':
```

```
            aminos.append('GLN')
        elif amino == 'R':
            aminos.append('ARG')
        elif amino == 'S':
            aminos.append('SER')
        elif amino == 'T':
            aminos.append('THR')
        elif amino == 'V':
            aminos.append('VAL')
        elif amino == 'W':
            aminos.append('TRP')
        elif amino == 'Y':
            aminos.append('TYR')
        elif amino == 'E':
            aminos.append('PCA')
        elif amino == 'B':
            aminos.append('ASX')
        elif amino == 'Z':
            aminos.append('GLX')
        elif amino == 'X':
            aminos.append('UNK')
    aminos[0]='N'+aminos[0]
    aminos[size-1]='C'+aminos[size-1]
#    print aminos
    arq=[]
#    print sequence
#    recursiveSearch(0, sequence, arq, aminos)
#    recursiveSearch3(0, sequence, arq, aminos)
    kou=0
    for y in range(1,7):
        for u in range(len(sequence)):
            g=0
#            if sequence[u][g][5]==y:
#                            arq.append([sequence[u][g][2],  sequence[u][g][3],
sequence[u][g][1]])
#            elif sequence[u][g][5]>y:
#                            arq.append([sequence[u][g][2],  sequence[u][g][3],
sequence[u][g][1]])
#            else:
            while (sequence[u][g][5]<y)and((g+1)<len(sequence[u])):
                print "tinha "+str(sequence[u][g][5])+" < "+str(y)
                g=g+1
                print "gerou "+str(sequence[u][g][5])
            arq.append([sequence[u][g][2],                  sequence[u][g][3],
sequence[u][g][1]])
        generateFilePDBeach(arq, kou, aminos)
        kou=kou+1
        arq=[]


def generateFilePDBeach(arq, kou, aminos):
#    global genFileGlob
#    if kou==0 or (kou==(1000*genFileGlob)):
#        save_file = open('structs/leaps'+str(genFileGlob-1)+'.in', 'a')
#        arquivo = '''quit'''
#        save_file.write(arquivo)
#        save_file.close()
#        if genFileGlob>0:
#            os.system('tleap -f structs/leaps'+str(genFileGlob-1)+'.in')
#        genFileGlob=genFileGlob+1
#        save_file = open('structs/leaps'+str(genFileGlob)+'.in', 'w')
#        arquivo = '''source leaprc.ff99
#'''
 #   else:
#        save_file = open('structs/leaps'+str(genFileGlob)+'.in', 'a')
#        arquivo ='''  '''
#    print arq
    save_file = open('structs/leaps'+str(kou)+'.in', 'w')
```

```
    arquivo = '''source leaprc.ff99
'''
    arquivo = arquivo+'''P_trp'''+str(kou)+''' = sequence {   NASN  LEU  TYR
ILE  GLN  TRP  LEU  LYS  ASP  GLY
                    GLY  PRO  SER  SER  GLY  ARG  PRO  PRO  PRO  CSER
                    }'''
#impose  P_trp  {  1  2  }  {  {  N  CA  C  N  '''+arq[0][0]+'''  }  {  CA  C  N  CA
'''+arq[0][1]+''' } { C N CA C '''+arq[0][2]+''' } }'''
    arquivo = arquivo+'''
impose P_trp'''+str(kou)+''' { 1 2 } { { N CA C N -56.140 } { CA C N CA
176.734 } { C N CA C -43.980 } }
impose P_trp'''+str(kou)+''' { 2 3 } { { N CA C N -51.309 } { CA C N CA
178.886 } '''
    for x in range(len(arq)):
        arquivo = arquivo+'''{ C N CA C '''+arq[x][2]+''' } }
impose P_trp'''+str(kou)+''' { '''+str(x+3)+''' '''+str(x+4)+''' } { { N CA C
N '''+arq[x][0]+''' } { CA C N CA '''+arq[x][1]+''' } '''
    arquivo = arquivo+'''{ C N CA C -77.264 } }
impose P_trp'''+str(kou)+''' { 19 20 } { { N CA C N 124.223 } { CA C N CA
171.650 } { C N CA C -78.100 } }
SavePdb P_trp'''+str(kou)+''' structs/t1L2Y'''+str(kou)+'''.pdb
Saveamberparm             P_trp'''+str(kou)+'''             structs/t1L2Y.top
structs/t1L2Y'''+str(kou)+'''.crd
quit
'''
    save_file.write(arquivo)
    save_file.close()
    os.system('tleap -f structs/leaps'+str(kou)+'.in')
#    os.system('quit')


def removeSeq(sequence, seq):
#  remove os arquivos PDB's que devem ser desconsiderados:
#  NCBIWWW3f.removeSeq(estrut, '1L2Y'):
    r=0
    for x in range(len(sequence)):
        for y in range(len(sequence[x])):
            try:
                while sequence[x][y][0] == seq:
                    sequence[x].pop(y)
            except:
                r=r+1


genFileGlob=0
def generateFilePDB(arq, kou, aminos):
    global genFileGlob
    if kou==0 or (kou==(1000*genFileGlob)):
        save_trajin = open('RMSD/tptraj'+str(genFileGlob)+'.in', 'a')
        arquivo = '''trajin structs/t1L2Y'''+str(kou)+'''.pdb
reference RMSD/1L2Y_01_a.pdb
rms reference out RMSD/rmsd_new'''+str(genFileGlob)+'''.out @CA,C,N
go
'''
        save_trajin.write(arquivo)
        save_trajin.close()
        save_trajin = open('RMSD/tptraj'+str(genFileGlob+1)+'.in', 'w')
        arquivo = '''
'''
        save_trajin.write(arquivo)
        save_trajin.close()
        save_file = open('structs/leaps'+str(genFileGlob)+'.in', 'a')
        arquivo = '''quit'''
        save_file.write(arquivo)
        save_file.close()
        os.system('tleap -f structs/leaps'+str(genFileGlob)+'.in')
        os.system('ptraj 1L2Y/1L2Y.top RMSD/tptraj'+str(genFileGlob)+'.in')
        genFileGlob=genFileGlob+1
```

```
        save_file = open('structs/leaps'+str(genFileGlob)+'.in', 'w')
        arquivo = '''source leaprc.ff99
'''
    else:
        save_trajin = open('RMSD/tptraj'+str(genFileGlob)+'.in', 'a')
        arquivo = '''trajin structs/t1L2Y'''+str(kou)+'''.pdb
'''
        save_trajin.write(arquivo)
        save_trajin.close()
        save_file = open('structs/leaps'+str(genFileGlob)+'.in', 'a')
        arquivo =''' '''
#    print arq
    arquivo = arquivo+'''P_trp'''+str(kou)+''' = sequence {   NASN  LEU  TYR
ILE  GLN  TRP  LEU  LYS  ASP  GLY
                    GLY  PRO  SER  SER  GLY  ARG  PRO  PRO  PRO  CSER
                    }'''
#impose P_trp { 1 2 } { { N CA C N '''+arq[0][0]+''' } { CA C N CA
'''+arq[0][1]+''' } { C N CA C '''+arq[0][2]+''' } }'''
    arquivo = arquivo+'''
impose P_trp'''+str(kou)+''' { 1 2 } { { N CA C N -56.140 } { CA C N CA
176.734 } { C N CA C -43.980 } }
impose P_trp'''+str(kou)+''' { 2 3 } { { N CA C N -51.309 } { CA C N CA
178.886 } '''
    for x in range(len(arq)):
        arquivo = arquivo+'''{ C N CA C '''+arq[x][2]+''' } }
impose P_trp'''+str(kou)+''' { '''+str(x+3)+''' '''+str(x+4)+''' } { { N CA C
N '''+arq[x][0]+''' } { CA C N CA '''+arq[x][1]+''' } '''
    arquivo = arquivo+'''{ C N CA C -77.264 } }
impose P_trp'''+str(kou)+''' { 19 20 } { { N CA C N 124.223 } { CA C N CA
171.650 } { C N CA C -78.100 } }
SavePdb P_trp'''+str(kou)+''' structs/t1L2Y'''+str(kou)+'''.pdb

'''
    save_file.write(arquivo)
    save_file.close()
#    os.system('quit')


def recursiveSearch2(number, sequence, arq, aminos):
#  Chamada do generateStructures que faz a recursão responsável pela
#  geração da ordem em que os phi e psi serão gerados.
#
#    len(sequence)):
#        print x
        global kou
#        for y in range(1):
        for y in range(len(sequence[number])):
            if number==0:
                if len(arq)>len(sequence)-1:
                   arq=arq[len(arq)-len(sequence)+1:]
#                print str(len(arq))+'  '+str(number)
                arq.insert(0,[sequence[number][y][2],   sequence[number][y][3],
sequence[number][y][1]])
#                print len(arq)
#     print arq
                generateFilePDB(arq, kou, aminos)
                kou=kou+1
            else:
             if len(arq)>len(sequence)-number-1:
                    arq=arq[len(arq)-(len(sequence)-number)+1:]
#                    print len(arq)
#                print 'foi '+str(kou)
#                if len(arq)>number:
#                 print str(len(arq))+'  '+str(number)
                arq.insert(0,[sequence[number][y][2],   sequence[number][y][3],
sequence[number][y][1]])
             recursiveSearch2(number-1, sequence, arq, aminos)
```

```
#        print y


def analysePhiPsi(filename, sequencia, pos):
    aminos = []
    size = len(sequencia)
    for x in range(size):
        amino = sequencia[x]
        if amino == 'A':
            aminos.append('ALA')
        elif amino == 'C':
            aminos.append('CYS')
        elif amino == 'D':
            aminos.append('ASP')
        elif amino == 'E':
            aminos.append('GLU')
        elif amino == 'F':
            aminos.append('PHE')
        elif amino == 'G':
            aminos.append('GLY')
        elif amino == 'H':
            aminos.append('HIS')
        elif amino == 'I':
            aminos.append('ILE')
        elif amino == 'K':
            aminos.append('LYS')
        elif amino == 'L':
            aminos.append('LEU')
        elif amino == 'M':
            aminos.append('MET')
        elif amino == 'N':
            aminos.append('ASN')
        elif amino == 'P':
            aminos.append('PRO')
        elif amino == 'Q':
            aminos.append('GLN')
        elif amino == 'R':
            aminos.append('ARG')
        elif amino == 'S':
            aminos.append('SER')
        elif amino == 'T':
            aminos.append('THR')
        elif amino == 'V':
            aminos.append('VAL')
        elif amino == 'W':
            aminos.append('TRP')
        elif amino == 'Y':
            aminos.append('TYR')
        elif amino == 'E':
            aminos.append('PCA')
        elif amino == 'B':
            aminos.append('ASX')
        elif amino == 'Z':
            aminos.append('GLX')
        elif amino == 'X':
            aminos.append('UNK')
#    print aminos

    file = open(filename,'r')
    lines = file.readlines();
    file.close();
    sizelines = len(lines)

    gerado = []


    for x in range(2,sizelines):
        if lines[x][7:10]==aminos[0]:
```

```
            l=1
            namino=0
            lastx=x
            while l==1:
                namino=namino+1
                lastx=lastx+1
                try:
                    if lines[lastx][7:10]==aminos[namino]:
                        if (namino+1)==size:
#                                print 'Esta entre as posicoes '+str(x-1)+' e
'+str(lastx-1)
                            gerado.append([lines[lastx+pos-size][7:10],
lines[lastx+pos-size][12:20],  lines[lastx+pos-size][21:29],  lines[lastx+pos-
size][30:38]])
#                            print gerado
                            l=0
#                       elif (namino+1)==pos:
#                                           gerado.append([lines[lastx][7:10],
lines[lastx][12:20], lines[lastx][21:29], lines[lastx][30:38]])

                    else:
                        l=0
                except:
                    l=0
    return gerado
#    print lines[2][7:10]

def runprog(sequencia):
    seq = fragmenta(sequencia)
    l = []
    for x in range(0,len(seq)):
        print x
        busca = qblast("blastp", "pdb", seq[x])
        busca2 = analyseblast(busca)
#        len(busca2)
        removesimilar(busca2)
        f = open(seq[x]+".pck", "w")
        pickle.dump(busca2, f)
        f.close()
#        len(busca2)
        l.append(busca2)
    f = open(sequencia+".pck", "w")
    pickle.dump(l, f)
    f.close()
    return l

def runprog5(sequencia):
    seq = fragmenta(sequencia)
    l = []
    for x in range(0,len(seq)):
        print x
        busca = qblast("blastp", "pdb", seq[x])
        busca2 = analyseblast(busca)
#        len(busca2)
        removesimilar(busca2)
        f = open(seq[x]+"5.pck", "w")
        pickle.dump(busca2[:6], f)
        f.close()
#        len(busca2)
        l.append(busca2[:6])
    f = open(sequencia+"5.pck", "w")
    pickle.dump(l, f)
    f.close()
    return l

def runprog3(sequencia):
    seq = fragmenta(sequencia)
    l = []
```

```
    for x in range(0,len(seq)):
        print x
        busca = qblast("blastp", "pdb", seq[x])
        busca2 = analyseblast(busca)
#        len(busca2)
        removesimilar(busca2)
        f = open(seq[x]+"5.pck", "w")
        pickle.dump(busca2[:4], f)
        f.close()
#        len(busca2)
        l.append(busca2[:4])
    f = open(sequencia+"5.pck", "w")
    pickle.dump(l, f)
    f.close()
    return l

def runprog4(sequencia):
    seq = fragmenta(sequencia)
    l = []
    for x in range(0,len(seq)):
        print x
        busca = qblast("blastp", "pdb", seq[x])
        busca2 = analyseblast(busca)
#        len(busca2)
        removesimilar(busca2)
        f = open(seq[x]+"5.pck", "w")
        pickle.dump(busca2[:5], f)
        f.close()
#        len(busca2)
        l.append(busca2[:5])
    f = open(sequencia+"5.pck", "w")
    pickle.dump(l, f)
    f.close()
    return l

def runprog2(sequencia):
    seq = fragmenta(sequencia)
    l = []
    for x in range(0,len(seq)):
        print x
        busca = qblast("blastp", "pdb", seq[x])
        busca2 = analyseblast(busca)
#        len(busca2)
        removesimilar(busca2)
        f = open(seq[x]+"5.pck", "w")
        pickle.dump(busca2[:3], f)
        f.close()
#        len(busca2)
        l.append(busca2[:3])
    f = open(sequencia+"5.pck", "w")
    pickle.dump(l, f)
    f.close()
    return l

def calcula(seq):
    g=1
    for x in range(len(seq)):
        if int(len(seq[x])) != int(0):
            g=g*(len(seq[x]))
            print int(len(seq[x]))
    return g
```