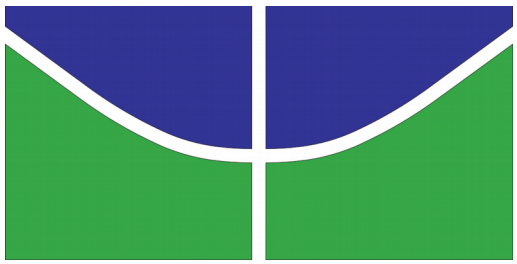

Universidade de Brasília

Instituto de Ciências Biológicas

Departamento de Biologia Celular

Laboratório de Biologia Teórica e Computacional

Análises de Coevolução Proteica

Aplicadas ao *Docking* de Proteínas

Dissertação apresentada ao Departamento de Biologia Celular do Instituto de Ciências Biológicas da Universidade de Brasília para obtenção do grau mestre em Biologia Molecular.

José Antonio Fiorote Santos

Orientador: Prof. Dr. Werner Treptow

Brasília, Fevereiro de 2019

Agradecimentos

Agradeço primeiramente à minha família, que me deu o apoio que tornou este caminho possível. Um enorme agradecimento à minha mãe, Zélia, pelo seu grande carinho e sabedoria, e também à minha irmã, Gisele, pelas noites de afetuoso carteado.

Meus amigos têm grande parte desta conquista. Quero agradecê-los imensamente. Em particular, agradeço à Mariza, que conseguiu me ensinar tanto com tanta ternura, e ao Caio, que contribuiu diretamente para esse trabalho, não só num ponto de vista técnico, mas também humano.

Um agradecimento especial aos companheiros do LBTC Alessandra, Leonardo, Letícia, Mônica e Vinícius, que sempre dispuseram seu tempo para me ajudar, e um agradecimento mais especial ainda à Camila, Diego e Miguel, a quem devo uma grande parte deste trabalho.

Agradeço ao meu orientador Werner Treptow pela suporte e cuidado, e por me mostrar um caminho científico novo e gratificante.

Agradeço também aos professores da banca, que reservaram um tempo para contribuir com este trabalho.

Por fim, agradeço à CAPES e ao CNPq pelo fomento.

Resumo

Interações entre proteínas são críticas para diversos processos celulares. As restrições dessas interações dependem do processo fisiológico que é intermediado. Uma das abordagens mais utilizadas para resolver essas interações entre proteínas em um nível atômico é o *docking* de proteínas. No entanto, a geração de falsos-positivos é uma desvantagem bem documentada dos algoritmos de *docking,* surgindo como resultado das limitações das funções de classificação, que são baseadas na descrição das energias de solvatação e da flexibilidade das proteínas. A análise de coevolução de proteínas pode ser útil para contornar este problema.

A coevolução de proteínas surge do conceito de coevolução macromolecular e refere-se a mutações correlacionadas de aminoácidos que derivam da seleção natural. A trilha de coevolução pode ser encontrada em um alinhamento múltiplo de sequências (MSA), por meio de análises que medem a entropia de suas colunas ou as correlações das sequências homólogas da espécie. Assim, combinamos essas análises e cálculos de *docking* para melhorar a modelagem de interações proteicas.

Buscando novos desenvolvimentos na área, criamos um software, o *Docking Score Module* (DSM). Especificamente, o DSM recebe como *input* os modelos tridimensionais que são gerados pelos programas de *docking* e um MSA baseado nas proteínas que compõem esses modelos. No coração do algoritmo, novos sub-alinhamentos são gerados para cada modelo, compostos exclusivamente por posições de aminoácidos que estão na interface entre as proteínas. Para cada novo alinhamento, o DSM calcula a informação mútua (MI), definida a partir das entropias de informações de Shannon, e o coeficiente de correlação linear (r), que relata a similaridade filogenética das sequências de proteínas no MSA. O *output* consiste em um arquivo de texto com informações do trabalho submetido e um gráfico, onde os modelos são ponderados por MI ou r.

Para os estudos de caso, nós submetemos onze complexos proteicos a diferentes servidores *web* de *docking*. Em um espaço de modelos de *docking* e levando em consideração os valores de MI e r, os modelos mais bem classificados DSM contêm um maior número de contatos nativos e estão mais próximos do complexo alvo.

Abstract

Protein-protein interactions are critical for diverse cell processes. The constraints of these interactions are dependent on the physiological process which are intermediated. One of the most used approaches to solve these protein-protein interactions at the atomic level is protein docking. The generation of false positive hits is however well documented drawback of docking algorithms as a result of the limitations of the scoring function in describing solvation energies and protein flexibility. Protein coevolution analysis can be usuful to contour this problem.

Protein coevolution arises from the concept of macromolecular coevolution and refers to amino acids correlated mutations that stems from natural selection. The trail of coevolution can be find in a multiple sequence alignment (MSA), by analysis wich measure the entropy of its columns or the correlations of the species homologous sequences. Thus, we combined these analyses and docking calculations to improve docking modeling.

Looking for new developments in the field, we have created a software, the Docking Score Module (DSM). Specifically, DSM takes as input 3D models generated from docking programs and a MSA based on proteins models. At the heart of the algorithm, new sub-alignments exclusively containing amino acid positions wich are in protein-protein interface are generated for each docking model. For each new sub-alignment, DSM calculates mutual information (MI), as defined from Shannon's information entropies, and correlation index (r), which reports phylogenetic similarity of protein sequences in MSA. The output consists of a text file with job information and a plot, in wich models are weighted by MI or r.

For the case studies, we docked eleven complexes using different docking web servers. Across a space of docking model and taking into consideration the value of MI an r, best ranked models in DSM contain the biggest number of native contacts and are closer to the target complex.

Sumário

Lista de Figuras

Lista de Tabelas

Siglas e Abreviações

FFT – Transformada Rápida de Fourier, do inglês, *Fast Fourier transform;*

$rmsd$ – Raiz do Desvio Médio Quadrático, do inglês, *Root Mean Square Deviation;*

MSA – Alinhamento Múltiplo de Sequências, do inglês, *Multiple Sequence Alignment;*

PDB – Banco de Dados de Proteínas, do inglês, *Protein Data Bank;*

MI – Informação Mútua, do inglês, *Mutual Information;*

UML – Linguagem de Modelagem Unificada, do inglês, *Unified Modeling Language;*

DCA – Análise de Acoplamento Direto, do inglês, *Direct Coupling Analysis*.

I – Introdução

I.1 – Interação entre Proteínas

Incontáveis processos celulares são mediados por interações entre proteínas. Elas são imprescindíveis em eventos vitais como replicação, transcrição e tradução do DNA, sinalização celular, transporte, entre outros[1]. A revista *Science*, em um edição comemorativa[2] em homenagem aos seus 125 anos, elegeu 125 questões de grande importância a serem respondidas pela ciência no próximo século. Dentre elas está a pergunta "como as proteínas encontram as suas parceiras?". A relevância da caracterização de interações proteicas pode ainda ser vislumbrada pelos vários artigos encontrados na literatura, que associam esse tema à revelação de novas possibilidades em áreas como desenho racional de fármacos[3–8] e avanços dentro do campo da biologia sintética, em especial na engenharia de proteínas[9,10].

As proteínas reconhecem-se em um ambiente "conturbado" e densamente povoado por outras estruturas biomoleculares. Embora muitas delas possam ser altamente específicas nas interações que realizam, há proteínas que podem ligar-se a várias outras parceiras. A formação de um complexo proteico também pode acontecer pela ligação de duas proteínas idênticas. Quando isso acontece, esse complexo recebe a classificação de homo-oligômero. Por sua vez, o complexo composto por proteínas diferentes é chamado de hétero-oligômero. Em sua maioria, os complexos homo-oligômeros são simétricos e estáveis, mas a estabilidade de complexos hétero-oligômeros não se encaixa em nenhum padrão, variando de acordo com a sua função, localização e estrutura[11].

Podemos distinguir complexos proteicos também com base na obrigatoriedade de sua formação. Os complexos são obrigatórios quando são formados por proteínas que não apresentam estabilidade por si mesmas. Esse tipo de complexo também pode ser chamado de dobramento em duas etapas, uma vez que a formação do complexo é concomitante ao dobramento das proteínas que dele participam. Por outro lado, os complexos não-obrigatórios apresentam proteínas que têm estabilidade para existir independentemente. De maneira geral, as proteínas de um complexo obrigatório são frequentemente coexpressas e colocalizadas durante a síntese e apresentam alta

afinidade, enquanto as de um complexo não-obrigatório podem ou não migrar de compartimento antes da sua interação **(Figura 1)**[12–14].

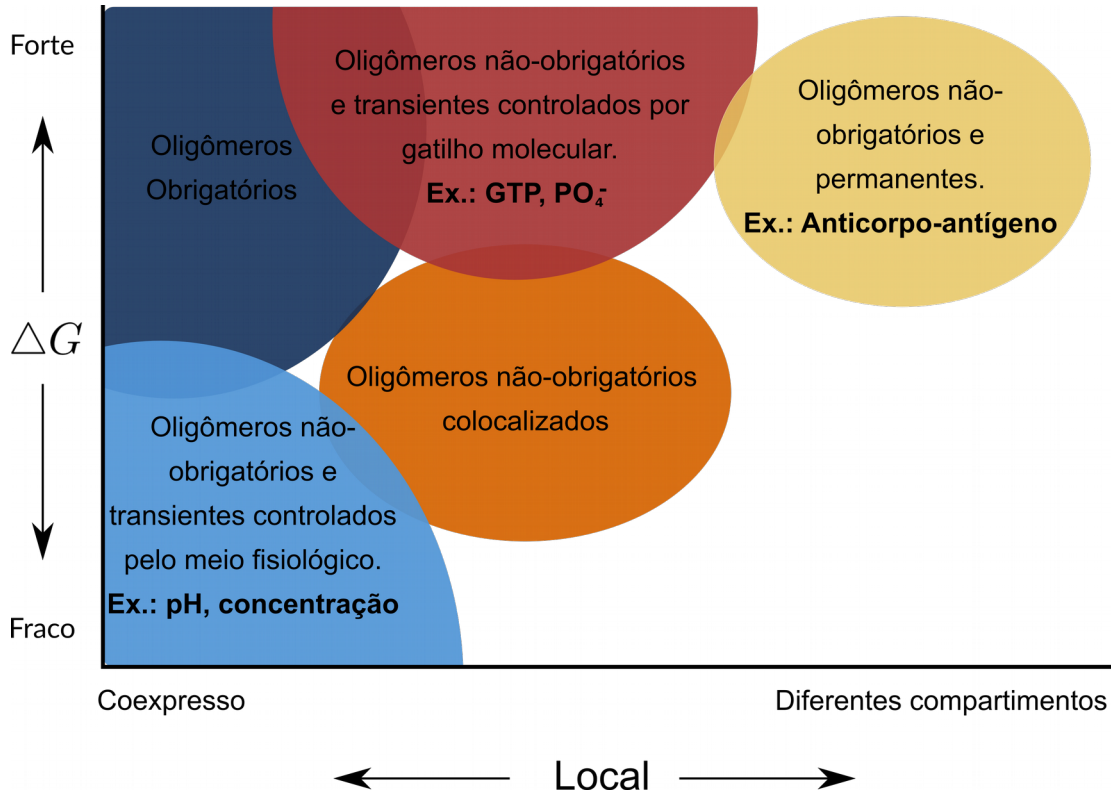

Figura 1: Relação entre diferentes tipos de interações proteicas, sua afinidade de ligação e a localização das proteínas envolvidas. Os gatilhos que controlam a oligomerização transiente estão grifados em negrito. Maiores mudanças conformacionais estão geralmente associadas a estas interações transientes. Adaptado de Nooren e colaboradores, 2003.

A interação entre proteínas pode ainda ser classificada em função da sua duração. Quando encontrada *in vivo,* uma interação permanente apresenta grande estabilidade, enquanto uma interação transiente associa-se e dissocia-se continuamente. Por consequência, as interações estrutural e funcionalmente obrigatórias são geralmente também interações permanentes. As interações não-obrigatórias, por outro lado, não apresentam uma relação direta quanto a obrigatoriedade de sua formação, podendo ser tanto permanentes quanto transitórias. Complexos transientes, em sua maioria, estão ligados a processos celulares regulatórios. Eles podem formar-se e romper-se por influência das condições do meio onde as proteínas se encontram, como pH, temperatura ou força iônica, ou ainda por um gatilho molecular específico, como os complexos dissociados por fosforilação, formados por proteínas de grande afinidade **(Figura 1)**[12,15,16]. Podemos

associar a duração de um complexo proteico ao valor de sua constante de dissociação (K_d). Complexos permanentes de grande estabilidade apresentam valores de K_d na ordem de nM (1×10^{-9}), enquanto complexos transientes estão na ordem de μM (1×10^{-6})[13].

Em última instância, a formação de um complexo proteico depende da concentração das proteínas envolvidas e da energia de ligação relativa aos estados alternativos desse complexo, sendo um reflexo das condições fisiológicas na qual ele é formado. O controle fisiológico da concentração desloca o equilíbrio da reação e é feito através de mudanças na taxa de expressão das proteínas do complexo ou de seus degradantes, ou por meio da taxa de difusão celular para proteínas não colocalizadas. Já o controle da afinidade de ligação é feito por meio da alteração do meio fisiológico, que provoca uma mudança nas propriedades físico-químicas e geométricas das interfaces das proteínas[12].

I.2 –*Docking* de Proteínas

Um método amplamente usado para modelagem computacional de interação entre proteínas é o *docking* de proteínas. Sugerido primeiramente em 1978, o termo "*docking"* é usado para descrever um conjunto de algoritmos computacionais que têm por objetivo encontrar o modelo mais próximo ao nativo da interação entre duas proteínas, ou seja, o complexo encontrado na natureza. Em uma visão geral, um programa de *docking* recebe os arquivos de coordenadas de uma proteína receptora A e uma proteína ligante B, e retorna um conjunto de arquivos de coordenadas tridimensionais dos modelos AB mais bem ranqueados de acordo com vários critérios, que geralmente incluem complementos esteroquímicos, eletrostáticos, formação de ligações de hidrogênio e solvatação **(Figura 2)**[17–19].

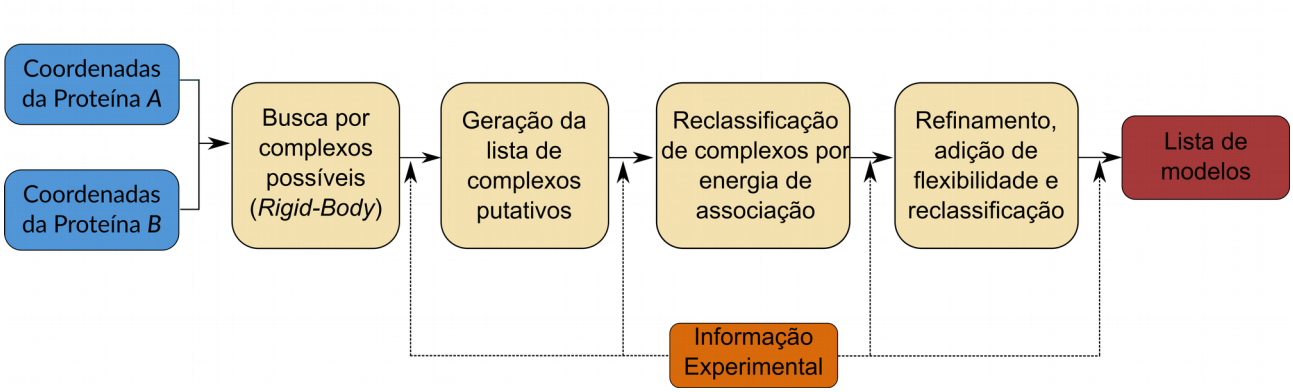

Figura 2: Estágios de um *docking* de proteínas. Adaptado de Smith e colaboradores, 2002.

Existem três elementos chaves para a realização de um *docking* de proteínas: a representação do sistema, a busca no espaço conformacional e a classificação de potenciais modelos de solução. A representação do sistema é feita com base na função de busca utilizada[20]. Dentre as técnicas usadas no estágio de busca estão as baseadas na transformada rápida de Fourier (FFT, do inglês "*Fast Fourier Transformation*")[21], a utilizada no programa BiGGER[22] e a técnica de *Geometrical Hashing*[23].

Nos programas de *docking* baseados em FFT[24–26] e também no BiGGER, que utiliza uma função desenvolvida por Palma e colaboradores[22], o sistema é representado em um *grid* cartesiano tridimensional e as proteínas são discretizadas em *voxels* (redução dos termos em inglês "*volume*" + "*pixels*"). Assim, modelos que apresentam sobreposições de *voxels* que estão no núcleo da proteína são penalizadas, enquanto os que apresentam *voxels* de interface que se posicionam lado a lado são favorecidos. Cada *voxel* pode assumir o valor 1 ou 0: 1 se a célula estiver dentro do raio de van der Waals de um átomo da proteína e 0, caso contrário[27]. Por sua vez, a técnica *geometric hashing* utiliza um algoritmo com representação explícita da forma das proteínas. Pontos críticos na superfície do ligante e do receptor são usados para definir quadros de coordenadas locais. As correspondências desses sistemas de coordenadas locais são classificadas pelo número de pontos críticos do ligante e do receptor que estão próximos. A busca espacial baseada na iteração da tabela faz desse o algoritmo de busca mais rápido dentre os três apresentados[17,20].

A busca no espaço conformacional consiste em amostrar quais são as possíveis conformações do complexo proteico. Para isso, é preciso realizar uma exaustiva busca global das

orientações de ligação em um espaço de 6 dimensões ($3D$ translacional + $3D$ rotacional). No entanto, essa busca tem um alto custo computacional devido aos inúmeros graus de liberdade de um sistema formado por duas proteínas, com uma complexidade na ordem de $O(N^6)$. A solução implementada pela maioria dos algoritmos de *docking* é desconsiderar inicialmente a flexibilidade das proteínas, por meio de uma técnica chamada *rigid-body docking:* uma das moléculas permanece estática, enquanto a outra move-se em torno dessa molécula fixa. Esta técnica diminui consideravelmente o problema da complexidade. A amostragem se dá por uma análise geométrica, por meio da localização de características complementares entre as superfícies das proteínas, tal como cavidades e protuberâncias locais. Como resultado dessa etapa, tem-se uma lista de complexos putativos da estrutura nativa a serem ranqueados por funções de energia[17,19,20].

A etapa anterior pode produzir umas poucas centenas de modelos estericamente possíveis. A etapa de classificação, por sua vez, é responsável por afunilar esta lista por meio da discriminação de associações proteicas energeticamente favoráveis. Ao estimar a energia de ligação entre duas proteínas, o algoritmo deve levar em conta as contribuições entálpicas e entrópicas desse processo. Para isso, é comum a aplicação de técnicas heurísticas na estimativa do componente entrópico da energia livre, assim como o uso de funções potenciais simplificadas para o cálculo do componente entálpico. Outras funções de ranqueamento incluem ainda a minimização de energia conformacional e de solvatação, o termo de van der Waals na expressão de energia livre ou a contagem de ligações de hidrogênio.[17,19,28–33].

Por fim, com uma quantidade reduzida de possíveis modelos, alguns algoritmos de *docking* ainda podem considerar um limitado grau de flexibilidade e introduzir, explicitamente, pequenas mudanças na superfície das cadeias laterais dos aminoácidos. Essa abordagem não diminui o desvio padrão quadrático médio $(rmsd)$ em relação à estrutura alvo, que é uma medida da distância entre duas estruturas, mas permite a remoção de sobreposições, além de diminuir a energia livre de ligação, impactando significativamente a performance do algoritmo[34–37]. Por fim, os modelos são novamente ranqueados e apresentados ao usuário na forma de arquivos de coordenadas.

O *docking* feito a partir de proteínas que foram cristalizadas individualmente, o chamado *unbound docking,* é de longe mais complexo que o *bound docking,* feito a partir de proteínas que

foram cristalizadas conjuntamente. Como explicado anteriormente, algumas proteínas encontram-se desordenadas antes da formação do complexo, de modo que o processo final de dobramento das proteínas acontece juntamente com a sua interação. Os resultados de *docking* obtidos a partir de estruturas conhecidas são geralmente satisfatórios. No entanto, quando aplicamos os mesmos algoritmos para o problema de *unbound docking*, os resultados dependem da extensão da mudança conformacional das proteínas na formação do complexo[20,38].

Outro problema bem documentado do *docking* de proteínas é a geração de falsos-positivos. Esse problema é um reflexo de deficiências nas funções que utilizam critérios energéticos para a classificação dos complexos. Assim, apesar de os algoritmos de *docking* encontrarem resultados muito próximos à estrutura nativa, eles não conseguem discriminá-los das falsas soluções, que são modelos bem classificados, mas estruturalmente distantes do complexo alvo. Encontrar o mínimo global de energia em uma paisagem energética tão complexa quanto a de um sistema formado por duas proteínas é uma tarefa árdua e ainda permanece como objeto de grande discussão. Grandes esforços têm sido feitos no desenvolvimento de funções mais eficientes, entretanto, esse objetivo ainda não foi plenamente alcançado[17,19,20,39]. O estudo de coevolução de proteínas pode lançar uma luz sobre essa questão.

I.3 – Coevolução

I.3.1 – Coevolução Macroscópica

Um dos fatos mais importantes elucidados pela teoria darwiniana é a auto-organização dos sistemas de organismos vivos. Os organismos se adaptam ao meio em que vivem por processos de multiplicação, herança e variação, por meio da seleção natural. A mudança dos organismos em face dessas pressões ambientais recebe o nome de evolução[40]. No entanto, um processo evolutivo não pode ser considerado isoladamente, uma vez que o meio em que um grupo de organismos vive é ativamente influenciado por mudanças em outros grupos. De forma que essas mudanças estão acopladas, gerando um equilíbrio dinâmico entre essas populações[41], [42].

O termo "coevolução" foi primeiramente explorado de forma explícita no clássico artigo de 1964 "*Butterflies and plants: a study in coevolution*", escrito por Ehrlich e Raven, e se refere a mudanças evolutivas recíprocas entre indivíduos ou grupos que interagem, dirigidas por seleção natural. Tal processo não pode ser diretamente observado, visto a extensão de sua escala temporal. Sendo assim, como podemos aprender sobre o processo de coevolução de organismos sem conhecer sua história evolutiva?[43] A resposta para esse questionamento, anteriormente feito por Ehrlich e Raven, está nas marcas deixadas nos sistemas dos organismos durante os processos de coevolução.

Processos de coevolução podem ser vislumbrados em grupos de organismos que estabelecem relações simbióticas, como as do tipo parasita-hospedeiro ou predador-presa. Nesses casos, as alterações morfológicas em um indivíduo são provocadas pela adaptação do outro indivíduo ao meio, de forma a manter uma vantagem evolutiva, como acontece em uma corrida armamentista[40]. São essas mudanças acopladas que nos permitem reconstituir a história coevolutiva desses organismos.

I.3.2 – Coevolução de Proteínas

As bases da teoria evolutiva foram desenvolvidas a partir de uma observação macroscópica da relação entre organismos. Em última instância, a mudança das frequências alélicas em um grupo é o produto de mutações em genes que codificam proteínas. De forma que o rastro da evolução está implícito no padrão de substituição dos aminoácidos das proteínas que, em um nível microscópico, são responsáveis pelas características macroscópicas observadas.

A interação entre dois aminoácidos, sejam da mesma proteína ou ainda de proteínas distintas, pode determinar a ocorrência de mudanças evolutivas recíprocas, pois as mudanças no primeiro aminácido podem modificar pressões seletivas que agem sobre o segundo aminoácido. Suponhamos que A e B sejam dois aminoácidos que interagem. Caso uma mudança aconteça no aminoácido A, de forma que haja uma diminuição do valor adaptativo dessa interação, essa mutação tende a não ser selecionada e, por consequência, não é observada. No entanto, se houver uma

mutação recíproca no aminácido B que mantenha ou aumente o valor adaptativo dessa interação, essas mutações tendem a ser selecionadas e observadas[44], como ilustrado na **Figura 3**.

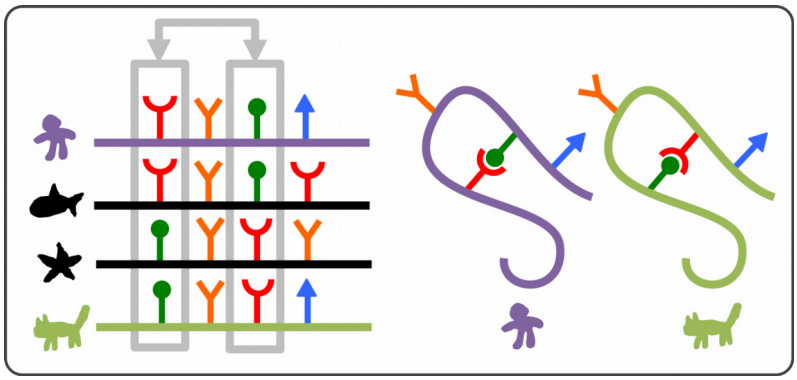

Figura 3: Ilustração de Sergey Ovchinnikov representando uma mutação correlata de aminoácidos em duas colunas de um alinhamento.

O processo de coevolução entre duas proteínas ocorre a partir da manutenção da interação entre elas, produzindo, ao longo do tempo, um conjunto de variações de suas sequências primárias. Com o crescimento exponencial de genomas sequenciados, a história coevolutiva das proteínas, implícita na sequência de suas proteínas homólogas, está mais perto de ser desvendada. A reconstituição dessas informações pode ser feita por meio da análise de um alinhamento múltiplo de sequências (do inglês *"multiple sequence alignment"* (MSA)) de uma proteína. O MSA consiste em um empilhamento de sequências primárias de proteínas homólogas alinhadas em relação às posições de seus aminoácidos. Essa disposição das sequências nos permite inferir estatisticamente quais são os resíduos de aminoácidos que são conservados e que podem ter importância funcional[45], ou também para deduzir contatos tridimensionais entre proteínas[46–48], revelando novas perspectivas para o *docking* de proteínas

Genes que codificam proteínas em um mesmo genoma podem possuir diferentes taxas evolutivas, decorrentes de restrições estruturais ou funcionais de seus produtos. Essa diferença da taxa de evolução também é observada ao longo de um mesmo gene, de forma que algumas posições deste gene evoluem mais rápido que outras. O trabalho de Echave e colaboradores, 2016[49], mostra

que a interface é a região da proteína que apresenta uma menor taxa de mudança **(Figura 4)**. Isto se deve às restrições impostas por sua interação. Logo, a interface entre duas proteínas pode nos fornecer maiores informações sobre seu processo coevolutivo.

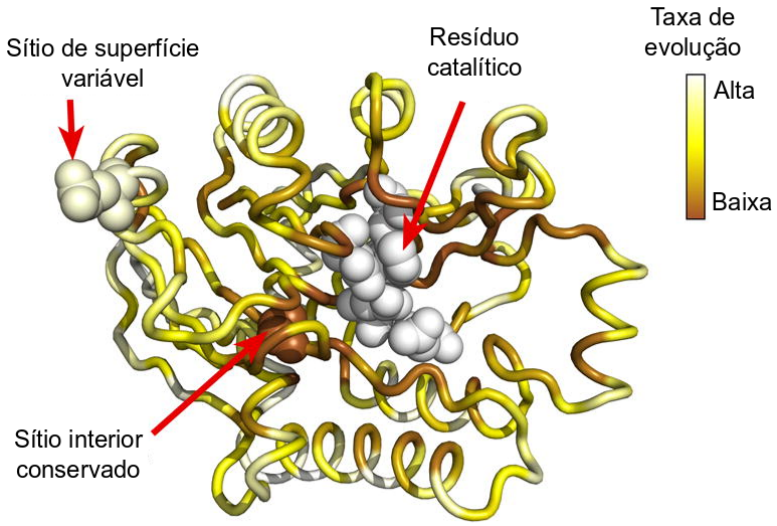

Figura 4: Taxa evolutiva de uma proteína globular. Representação esquemática da estrutura de Exonuclease III de ***Escherichia coli*** (código do PDB: 1AKO) colorida por taxa de evolução. As setas vermelhas destacam as diferentes restrições estruturais e funcionais sobre a taxa evolutiva. Adaptado de Echave e colaboradores, 2016.

Variadas técnicas são usadas para medir a quantidade de informação coevolutiva presente nas proteínas. Elas são baseadas em modelos estatísticos e algumas abordagens levam em conta o perfil filogenético das sequências homólogas ou também a incerteza relacionada aos aminoácidos nas colunas de um MSA.[50] Dentre elas estão o cálculo do coeficiente linear de correção[51], de dimensão filogenética, e a informação mútua[47], que leva em conta a entropia de uma posição de aminoácidos. Como a diferença entre os modelos gerados pelo *docking* está na interface entre as proteínas do complexo, nós podemos usar essas metodologias, descritas no Capítulo III, para medir a quantidade de informação armazenada nessa interface para discriminar modelos em função da quantidade dos seus contatos nativos e, assim, melhorar as predições deste método.

II – Objetivos

II.1 – Objetivo Geral

O presente trabalho tem como objetivo geral o desenvolver uma ferramenta computacional que combine análises de coevolução de proteínas para melhorar as predições realizadas por programas de *docking*.

II.2 – Objetivos Específicos

i - Implementar o algoritmo para o cálculo de informação mútua entre posições de alinhamentos.

ii – Implementar o algoritmo para o cálculo do coeficiente de correlação linear entre alinhamentos

iii – Comparar o uso isolado das métricas de coevolução com o seu uso conjunto em relação a sua capacidade de distinguir contatos nativos em uma proteína;

iv – Implementar uma ferramenta computacional que automatize as análises de coevolução de proteínas para modelos de *docking*.

III – Metodologia

III.1 – Determinação da Distância de Interface

Os aminoácidos que estão presentes na interface entre duas proteínas foram definidos com base em um valor da distância de contato $r_C(\mathring{A})$. Se a distância entre os pontos de referência de dois aminoácidos de diferentes proteínas é menor ou igual à distância r_C, esses aminoácidos estão em contato e são considerados parte da interface **(Figura 5)**. Neste trabalho, o ponto de referência adotado foi a coordenada do primeiro carbono da cadeia lateral ($C\beta$) para todos os aminoácidos, com exceção da glicina, onde a referência considerada foi o $C\alpha$. O valor de r_C foi definido como $8\mathring{A}$ com base no trabalho de Morcos e colaboradores[52], sendo este o valor usado em todos os casos deste trabalho.

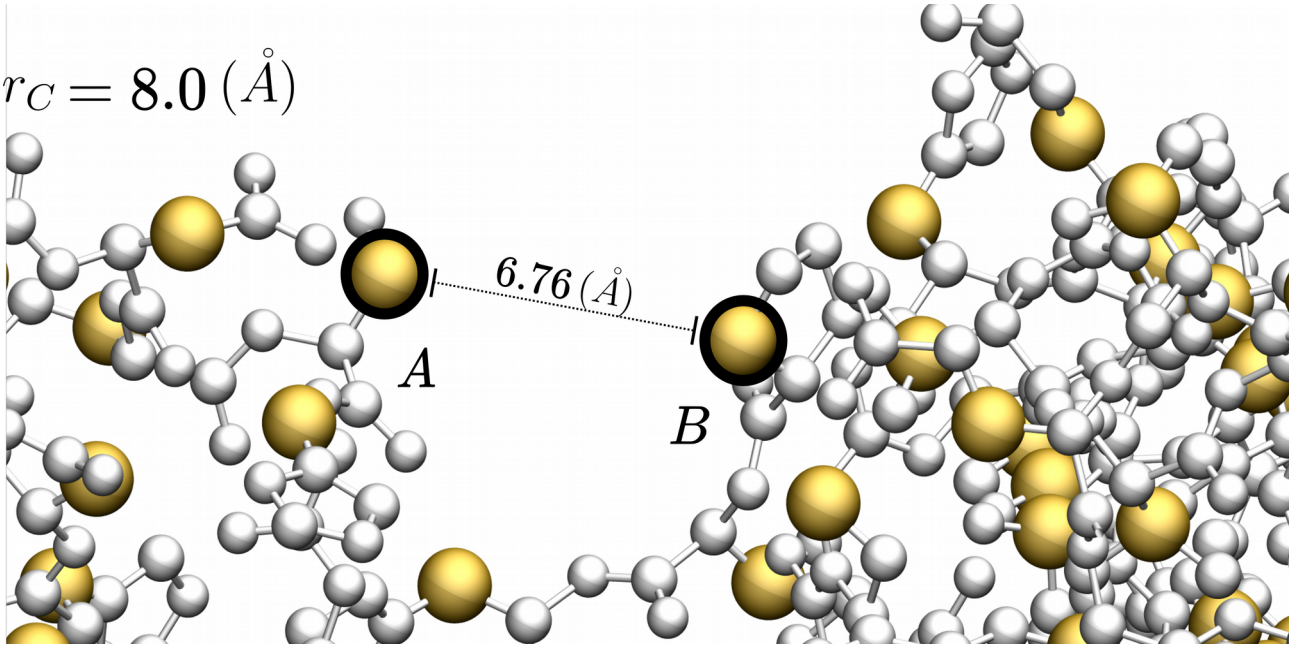

Figura 5: Representação do contato entre dois aminoácidos. Os $C\beta$ (representados em amarelo) dos aminoácidos A e B são os pontos de referência destes resíduos e estão separados por uma distância de $6.76\mathring{A}$. Como o valor de r_C é igual à $8.0\mathring{A}$, estes resíduos de aminoácidos fazem contato.

III.2 – Geração de Modelos Putativos

O primeiro passo para a validação de um algoritmo é certificar seu funcionamento para casos conhecidos. Dessa forma, os complexos de interação já conhecida descritos na **Tabela 1 (**No decorrer deste trabalho, os complexos serão sempre citados pelo seu PDB *Id* juntamente com as letras que representam a proteína A e a proteína B no arquivo de coordenadas) foram submetidos a quatro diferentes servidores de *docking* de proteínas: GRAMMX[53], HDOCK[25], ROSIE[54] e ZDOCK[55], cujos URLs estão listados na **Tabela 2** (Aqui, o termo servidor refere-se ao programa que estabelece o acesso dos usuários ao algoritmo de *docking* utilizando *Hypertext Transfer Protocol* (HTTP), também chamado de servidor *web*). Esses servidores de *docking* foram escolhidos devido à facilidade de sua utilização. Nenhum parâmetro adicional como a indicação de sítios catalíticos ou algum tipo de restrição foi previamente informado. No entanto, a documentação do servidor ROSIE estipula uma distância máxima de $6\mathring{A}$ entre as proteínas do complexo. Dessa maneira, submeteu-se para esse servidor o arquivo de coordenadas do complexo em sua forma nativa, enquanto para os outros servidores os aquivos de coordenadas de cada uma das proteínas foi submetido de forma separada. Para cada um dos servidores de *docking*, foram selecionados os dez modelos mais bem classificados de cada complexo, de modo que obtivemos um total de 40 modelos para a maioria dos complexos. Os complexos em negrito na **Tabela 1** não foram submetidas ao servidor HDOCK e o complexo 3ZET A-B também não foi submetido ao servidor ROSIE. Sendo assim, os complexos 1YOH I-L, 1VET A-B e 5F5S A-B possuem 30 modelos putativos, e o complexo 3ZET A-B, possui 20 modelos. Os valores de $rmsd$ em relação à estrutura nativa e a ordem de classificação dos modelos gerados por cada servidor de *docking* encontram-se no Capítulo IV e nas tabelas referentes a cada complexo, no Anexo I.

Tabela 1: Relação do complexos proteicos utilizados nesse trabalho. São listados o código da estrutura cristalográfica, o organismo utilizado como vetor ou do qual a proteína foi isolada, as proteínas utilizadas como *input* na geração dos modelos de *docking* e também a resolução da estrutura.

PDB *Id*	Organismo	Proteína A	Proteína B	Resolução da Estrutura
1BXR	*Escherichia coli – strain K12*	*Carbamoyl – phosphate synthetase – Chain A*	*Carbamoyl – phosphate synthetase – Chain B*	*2.1 Å*
1EP3	*Lactococcus lactis subsp. Cremoris – strain MG1363*	*Dihydroorotate dehydrogenase B. – Chain A*	*Dihydroorotate dehydrogenase B. – Chain B*	*2.1 Å*
1TYG	*Bacillus subtilis – strain 168*	*Thiazole synthase/ThiS complex – Chain A*	*Thiazole synthase/ThiS complex – Chain B*	*3.15 Å*
2VPZ	*Thermus thermophilus – strain HB27 / ATCC BAA-163 / DSM 7039*	*Polysulfide reductase – Chain A*	*Polysulfide reductase – Chain B*	*2.4 Å*
2Y69	*Bos taurus*	*Bovine heart cytochrome c oxidase re-refined – Chain A*	*Bovine heart cytochrome c oxidase re-refined – Chain B*	*1.95 Å*
2Y69	*Bos taurus*	*Bovine heart cytochrome c oxidase re-refined – Chain A*	*Bovine heart cytochrome c oxidase re-refined – Chain C*	*1.95 Å*
3OAA	*Escherichia coli – strain K12*	*F1-ATP synthase – Chain G*	*F1-ATP synthase – Chain H*	*3.26 Å*
1OYH	*Homo sapiens*	*Antithrombin-III – Chain I*	*Antithrombin-III – Chain L*	*2.62 Å*
1VET	*Mus musculus*	*Mitogen-activated protein kinase kinase 1 interacting protein 1 – Chain A*	*Late endosomal/lysos-omal Mp1 interacting protein – Chain B*	*1.9 Å*
3ZET	*Salmonella typhimurium (strain 4/74)*	*Putative M22 peptidase YEAZ – Chain A*	*Probable tRNA threonylcarbamo-yladenosine biosynthesis protein GCP – Chain B*	*2.31 Å*
5F5S	*Homo sapiens*	*Pre-mRNA – splicing factor 38A – Chain A*	*Microfibrillar – associated protein 1 – Chain B*	*2.4 Å*

Tabela 2: Lista de servidores web de *docking* utilizados na geração de modelos. São listados o nome do servidor de *docking*, o endereço eletrônico do servidor e data do último acesso.

Servidor	URL
GRAMMX	http://vakser.compbio.ku.edu/resources/gramm/grammx/ Acessado em: 20/01/2019
HDOCK	http://hdock.phys.hust.edu.cn/ Acessado em: 20/01/2019
ZDOCK	http://zdock.umassmed.edu/ Acessado em: 20/01/2019
ROSIE (Rosetta *server*)	http://rosie.rosettacommons.org/docking2 Acessado em: 20/01/2019

III.3 – Alinhamento Múltiplo de Sequências

O MSA da maioria das proteínas dos complexos citados na **Tabela 1** foram obtidos diretamente do trabalho de Ovchinnikov e colaboradores[48]. As proteínas dos complexos grafados em negrito foram construídos a partir de uma consulta na base de dados PDB Bind, que é composto por estruturas de proteínas que sabidamente interagem na natureza. Foram considerados na seleção somente complexos que apresentavam estruturas globulares, cujas proteínas constituintes tivessem um domínio único no Pfam. A busca por sequências homólogas às das proteínas desses complexos foi feita por meio da base de dados Blast-P, utilizando um número de *hits* de 100.000, sem qualquer critério de tamanho ou limite de *e-value*. Essas sequências foram filtradas, de modo que os alinhamentos das proteínas de cada complexo fossem compostos por sequências oriundas das mesmas espécies, representadas apenas uma vez em cada alinhamento. Após isso, foram descartadas as sequências que apresentavam um *e-value* $> 10^{-5}$ e que possuíam um tamanho 70% menor ou 130% maior do que a média do tamanho das sequências restantes. O alinhamento das sequências foi feito mediante o uso do programa ClustalW 2.1 e todos os parâmetros foram deixados na configuração padrão. O alinhamentos da proteína A, MSA^A, e o da proteína B, MSA^B, de cada um desses complexos, foram dispostos em formas comparáveis a árvores filogenéticas que possuem a mesma ramificação. Cada $a = \{1..., M\}$ folha na terminação do ramo representa uma sequência de uma mesma espécie em ambos os alinhamentos **(Figura 6, Letra a)**.

Sub-alinhamentos foram gerados a partir dos alinhamentos resultantes da etapa anterior. Como discutido na introdução, a interface das proteínas guarda grande quantidade de informação evolutiva. Desta modo, os sub-alinhamentos MSA^A_{sub} e MSA^B_{sub} foram gerados unicamente a partir das colunas das posições i dos aminoácidos presentes na interface entre as proteínas A e B para cada modelo putativo gerado pelos servidores de *docking* **(Figura 6, Letra b)**. Foram considerados aminoácidos de interface aqueles cujo ponto de referência encontra-se dentro de uma distância r_C de qualquer ponto de referência de um aminoácido da proteína vizinha, como discutido na Seção III.1.

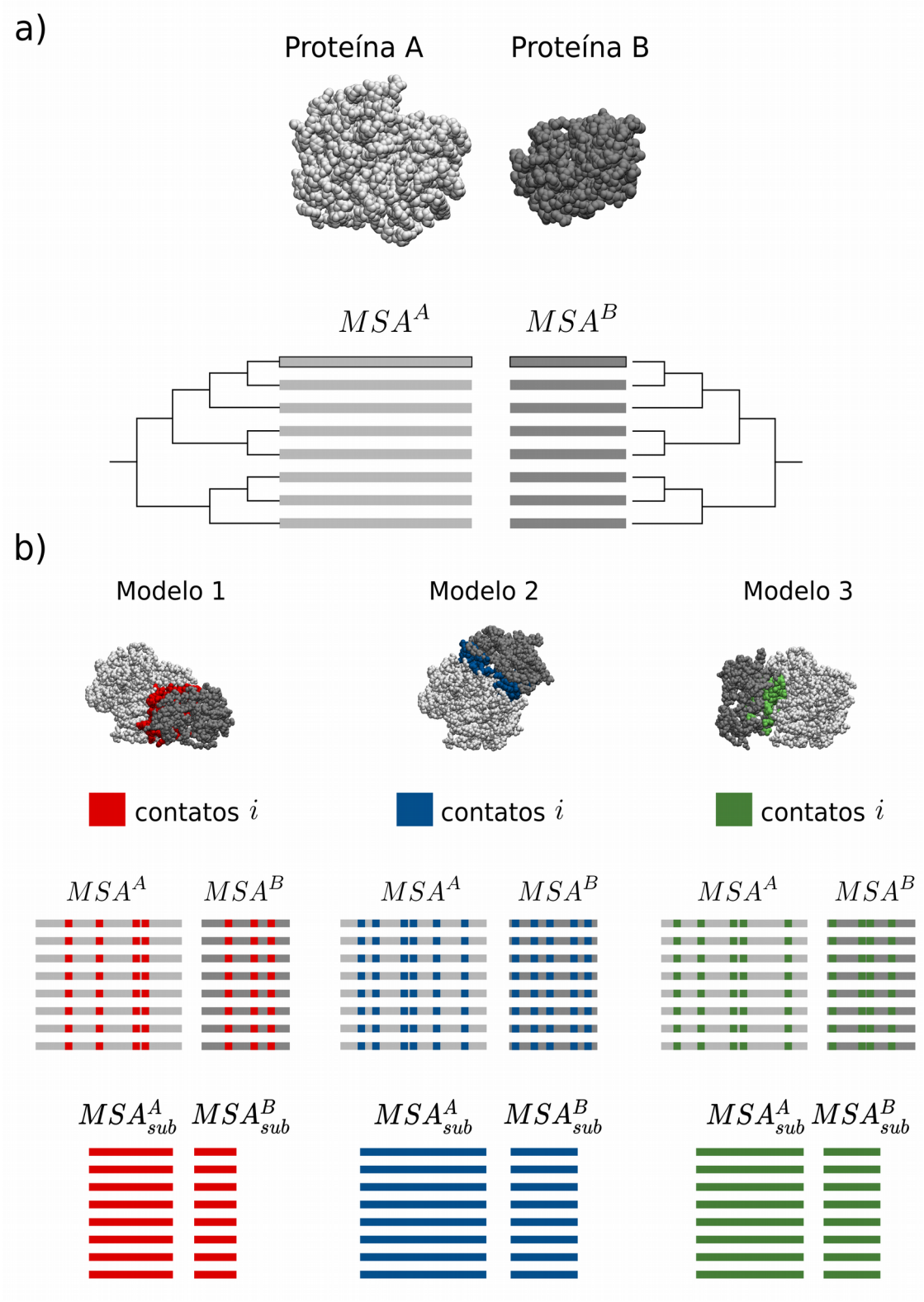

Figura 6: Etapas da construção dos sub-alinhamentos. a) representação da construção do MSA^A e MSA^B a partir das proteínas A e B do complexo proteico. **b)** representação da construção dos sub-alinhamentos MSA^A_{sub} e MSA^B_{sub} a partir das posições dos aminoácidos que estão presentes na interface dos modelos.

III.4 – Métricas de Coevolução

Utilizamos os sub-alinhamentos extraídos a partir do MSA^A e MSA^B para classificar cada um dos modelos de *docking* e indicar quais modelos realizam contatos mais favoráveis. Para isso, utilizamos conjuntamente duas métricas de coevolução: a informação mútua e o coeficiente de correlação linear, como explicado no Capítulo II.

III.4.1 – Teoria da Informação

Originalmente proposta por Claude E. Shannon em um artigo de 1948 chamado "*A Mathematical Theory of Communications*", a teoria da informação envolve a criação de modelos descritivos dos processos de comunicação. Shannon afirma que o problema fundamental das comunicações é a reprodução exata ou aproximada, em um dado ponto, de uma mensagem emitida em outro ponto em um sistema de comunicação **(Figura 7)**. O aspecto relevante dessa mensagem, entretanto, não está na sua semântica, mas no fato de que essa determinada mensagem foi selecionada em um conjunto de outras mensagens possíveis. Essa medida de liberdade na seleção de uma mensagem está ligada à sua informação[56].

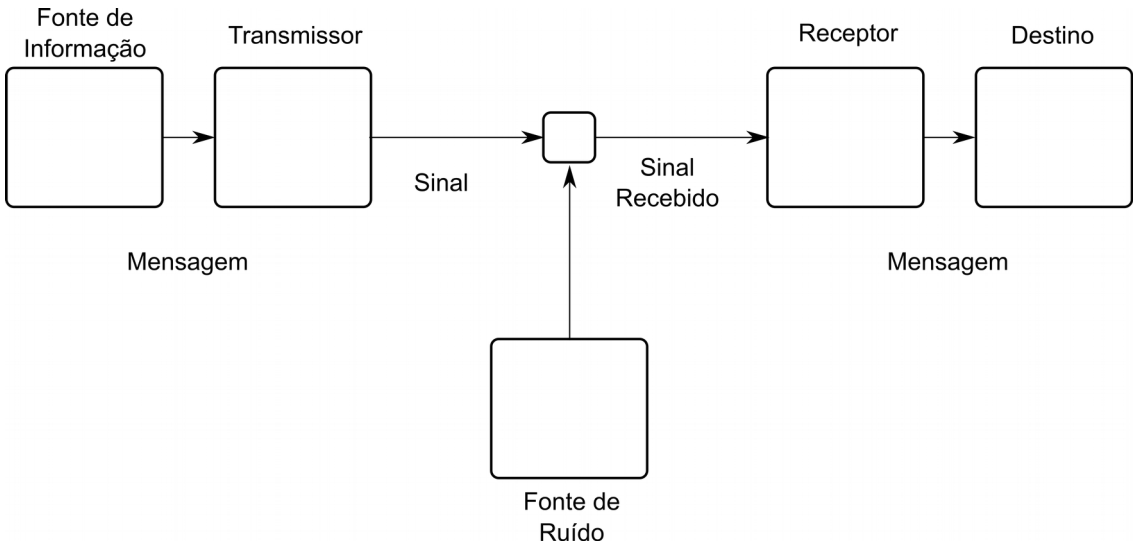

Figura 7: Esquema geral de um sistema de comunicação. A fonte de informação seleciona uma mensagem entre um conjunto de mensagens possíveis, o transmissor codifica a mensagem e a envia através de um canal de comunicação, e o receptor decodifica a mensagem a ser recebida no destino. Adaptado de Shannon, 1948.

III.4.2 – Entropia e Informação Mútua

Em razão de sua amplitude, o conceito de informação não pode ser contido em uma definição simples. Entretanto, para qualquer função de distribuição de probabilidades, é possível definir uma quantidade que contem propriedades que correspondam à noção de uma medida intuitiva de informação. Essa quantidade é denominada **entropia** (H), e pode ser conceituada como a medida de incerteza de uma variável estocástica.

A entropia H de uma variável estocástica discreta X de função de probabilidade $\rho(x) = Pr\{X = x\}$, definida no alfabeto χ de tamanho $|\chi|$, é dada por:

$$H(X) = -\sum_{x} \rho(x)\ ln\ \rho(x) \tag{1}$$

Estendendo a definição acima, a entropia conjunta de um par de variáveis estocásticas discretas X e Y com distribuição conjunta $p(x, y)$ é dada por:

$$H(X,Y) = -\sum_{x,y} \rho(x,y)\ ln(\rho(x,y)) \tag{2}$$

e é equivalente à entropia de uma variável mais a entropia condicional da outra variável:

$$H(X,Y) = H(X) + H(Y \mid X) \tag{3}$$

A entropia pode ser expressa em diferentes unidades, a depender da base do logaritmo. A entropia é expressa em *nats,* nesse caso, onde a base do logaritmo é igual a e. Para as bases 2, 3, 4, 5, 6, 7 e 10, por exemplo, ela é expressa respectivamente em bits, trits, quarts, quints, sexts, septs e dits. A troca de base pode ser feita pela relação $H_b(X) = (log_b a) \times H_a(X)$.

A partir da definição de entropia, podemos também estabelecer uma medida de dependência mútua entre duas variáveis, denominada **informação mútua** (do inglês "*mutual information*" (MI)

). Mais especificamente, a MI quantifica o quanto de informação pode-se obter sobre uma variável a partir da observação de outra variável. Consideremos duas variáveis estocásticas discretas X e Y, com funções de probabilidade $\rho(x)$ e $\rho(y)$. A informação mútua entre essas duas variáveis é:

$$MI(X;Y) = \sum_{x,y} \rho(x,y)\ ln \frac{\rho(x,y)}{\rho(x)\rho(y)} \tag{4}$$

Assim, quanto maior o valor de informação mútua, mais acopladas estão essas variáveis.

III.4.3 – Relação entre Entropia e Informação Mútua

Podemos reescrever a equação [4] da seguinte maneira:

$$MI(X;Y) = \sum_{x,y} \rho(x,y)\ ln \frac{\rho(x \mid y)}{\rho(x)} \tag{5}$$

$$= -\sum_{x,y} \rho(x,y)\ ln\ \rho(x) + \sum_{x,y} \rho(x,y)\ ln\ \rho(x \mid y) \tag{6}$$

$$= -\sum_{x} \rho(x)\ ln\ \rho(x) - \left(-\sum_{x,y} \rho(x,y)\ ln\ \rho(x \mid y) \right) \tag{7}$$

$$= H(X) - H(X \mid Y) \tag{8}$$

Inferimos, então, que a informação mútua é a redução da incerteza da variável X pelo conhecimento da variável Y. Similarmente:

$$MI(X;Y) = H(Y) - H(Y \mid X) \tag{9}$$

Desta forma, conhecemos tanto de X a partir de Y quanto de Y a partir de X. Como definido na equação [3]:

$$H(X,Y) = H(X) + H(Y \mid X)$$

concluímos que:

$$MI(X;Y) = H(X) + H(Y) - H(X \mid Y) \tag{10}$$

logo:

$$MI(X;X) = H(X) - H(X \mid X) = H(X) \tag{11}$$

a informação mútua de uma variável e si mesma é a entropia desta variável. A **Figura 8** representa a relação entre $H(X)$, $H(Y)$, $H(X,Y)$, $H(X \mid Y)$, $H(Y \mid X)$ e $MI(X;Y)$. Note que a informação mútua entre as variáveis X e Yrepresenta a intersecção entre as informações dessas variáveis[57].

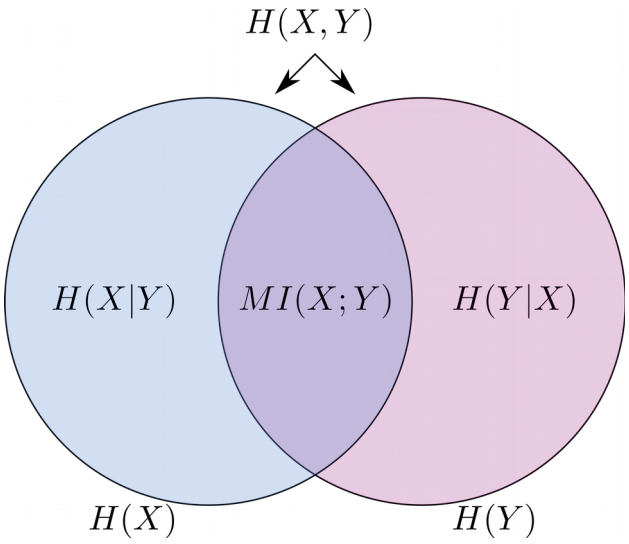

Figura 8: Relação entre informação mútua e entropia.
Adaptado de Cover e Thomas, 2006.

III.4.4 – Calculando a Informação Mútua entre Proteínas

Apesar do contexto em que foi desenvolvida, a teoria da informação encontra aplicações em diversas áreas, como economia, estatística, física e também na biologia. Aplicando seus conceitos à análise de coevolução de proteínas, podemos entender cada uma das posições dos aminoácidos da sequência primária de uma proteína como um canal por onde é transmitida a informação evolutiva. Esta informação pode ser inferida a partir da distribuição das frequências sítio-específicas para cada coluna de um MSA: uma alta frequência de um aminoácido em uma determinada posição pode indicar que há uma pressão evolutiva que favorece a sua conservação. A informação transmitida por esse canal está relacionada ao grau de incerteza médio do aminoácido nessa posição, ou seja, a medida da conservação desse aminoácido é a medida da entropia desse canal.

Em uma estrutura tridimensional nativa, os aminoácidos realizam contatos dependendo da sua natureza química. Assim, quando acontece uma mutação deletéria em algum aminoácido e o equilíbrio químico dessa interação é desfeito, é preciso que haja uma mutação compensatória nos aminoácidos vizinhos, de modo a restabelecer a química dessa interação. Dessa forma, as distribuições de frequências de aminoácidos em diferentes posições são dependentes umas das outras. O acoplamento entre as mudanças dos aminoácidos pode ser quantificado através da medida de informação mútua.

Consideremos agora um modelo proposto por um algoritmo de *docking*, onde duas proteínas A e B interagem através de $i = 1, ..., N$ contatos entre aminoácidos. Como mostrado anteriormente, os sub-alinhamentos MSA^A_{sub} e MSA^B_{sub} são respectivamente gerados a partir dos alinhamentos MSA^A e MSA^B. As sequências $a = \{1..., M\}$ presentes nesses sub-alinhamentos podem ser respectivamente descritas pelos blocos de N variáveis estocásticas discretas $X^N \equiv (X_1, ..., X_N)$ e $Y^N \equiv (Y_1, ..., Y_N)$, com funções de probabilidade $\left\{\rho(x^N), \rho(y^N), \rho(x^N, y^N)\right\}$ para cada sequência conjunta $\left\{X^N, Y^N\right\}_{|\chi|^{2N}}$ definida no alfabeto χ de tamanho $|\chi|$. Assim, a quantidade de informação que a proteína A guarda sobre a proteína B é dada pela informação mútua entre X^N e Y^N. A MI atinge seu limite inferior caso X^N e Y^N sejam totalmente independentes. Por outro lado, seu valor máximo é alcançado quando estas variáveis estão perfeitamente correlacionadas.

Tais valores não devem exceder o limite da entropia de qualquer um dos blocos de variáveis $H(X^N)$ e $H(Y^N)$, como pode ser visto na relação mostrada na **Figura 8**.

As probabilidades conjuntas requeridas para o cálculo de MI foram inferidas a partir das frequências observadas $f = \{f_{x_i,y_i}\}$, de modo que:

$$\rho(x_i, y_i) \equiv f_{x_i,y_i} \tag{12}$$

Os cálculos das frequências simples e duplas dos aminoácidos foram baseados na metodologia proposta por Morcos e colaboradores[52], onde as sequências homólogas apresentam um peso diferente, dependendo da sua similaridade. A escolha desse método teve como objetivo eliminar vieses na amostragem das sequências. As frequências duplas foram calculadas da seguinte forma:

$$fx_i, y_i = \frac{1}{M_{eff} + \lambda} \left[\frac{\lambda}{|\chi|^2} + \sum_{m=1}^{M} \frac{1}{n^m} \ \delta_{x_i^m, y_i^m, x_i, y_i} \right] \tag{13}$$

onde n^m representa o número de sequências similares m'que possuem com uma certa distância de Hamming = θ da sequência m

$$n^m = |m' \mid 1 \leq m' \leq M, h(m, m') \geq \theta| \tag{14}$$

e

$$M_{eff} = \sum_{m=1}^{M} \ \frac{1}{n^m} \tag{15}$$

é o número de sequências primárias distinguíveis no limiar de distância θ. O delta de Kronecker $\delta_{x_i^m, y_i^m, x_i, y_i}$, garante a contagem apenas de (x_i, y_i), e é regularizado pela pseudocontagem λ[58], que corrige possíveis vieses de amostragem adicionando um valor prévio à contagem de cada

aminoácido, de modo a não permitir que a matriz de frequências seja singular. O tamanho do alfabeto $|\chi|$ foi definido como 21, para os 20 diferentes aminoácidos e o *gap*.

De maneira análoga, as probabilidades simples $\rho(x_i) \equiv fx_i$ e $\rho(y_i) \equiv fy_i$ foram calculadas a partir das frequências simples:

$$\begin{cases} f_{x_i y_i} = \frac{1}{M_{eff}+\lambda} \left[\frac{\lambda}{|\chi|} + \sum_{m=1}^{M} \frac{1}{n^m} \ \delta_{x_i^m, x_i} \right] \\ f_{x_i y_i} = \frac{1}{M_{eff}+\lambda} \left[\frac{\lambda}{|\chi|} + \sum_{m=1}^{M} \frac{1}{n^m} \ \delta_{y_i^m, y_i} \right] \end{cases} \tag{16}$$

Ao final, a informação mútua foi normalizada de modo a considerar o tamanho da interface de cada modelo. Assim, o valor de informação mútua para cada sub-alinhamento foi dividido pela soma de sua entropia conjunta, de acordo com a relação explicitada na **Figura 8**.

III.4.5 – Coeficiente de Correlação Linear

O coeficiente de correlação linear (r) é uma quantificação da relação entre os alinhamentos da proteína A e da proteína B e foi calculado usando a fundamentação teórica proposta por Pazos e Valencia, 2001[51]. Os alinhamentos MSA_{sub}^{A} e MSA_{sub}^{B} foram usados para construir as matrizes D^A e D^B do tipo BLOSUM62[59] (***BLO**cks of Amino Acid **SU**bstitution **M**atrix*) **(Figura 9)**. A matriz foi construída utilizando funções já implementadas no Biopython 1.72[60] (ver Anexo II). A correlação entre essas matrizes D^A e D^B foi calculada usando a seguinte relação:

$$r = \frac{\sum_{i=1}^{n} (d_i^A - \overline{d^A})(d_i^B - \overline{d^B})}{\sqrt{\sum_{i=1}^{n} (d_i^A - \overline{d^A})^2} \sqrt{\sum_{i=1}^{n} (d_i^B - \overline{d^B})^2}} \tag{17}$$

onde d_i^A e d_i^B representam os elementos das matrizes de distância D^A e D^B, respectivamente, e $\overline{d^A}$ e $\overline{d^B}$ são os valores médios das matrizes D^A e D^B, respectivamente.

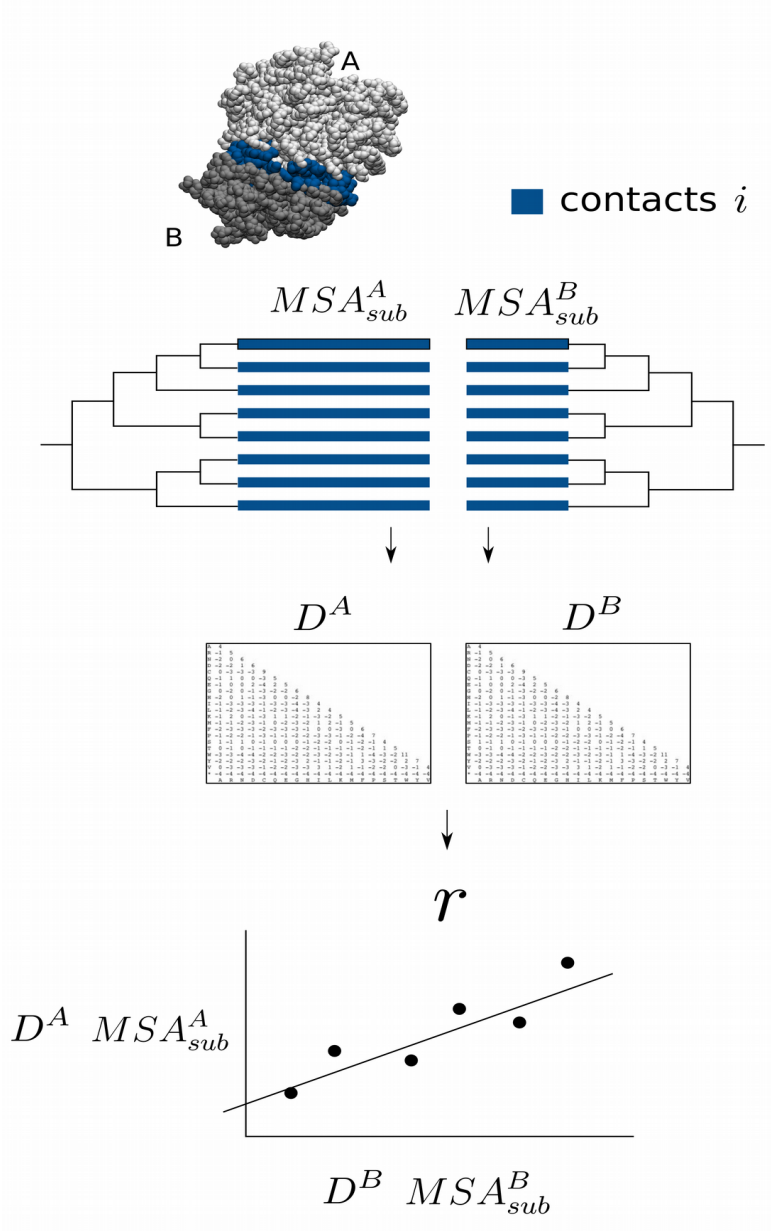

Figura 9: Esquema geral do cálculo do coeficiente de correlação linear (r). Adaptado de Pazos e Valencia, 2001.

III.5 – Desenvolvimento da Ferramenta Computacional

O esquema geral das etapas do *software* desenvolvido está representado na **Figura 10**. A entrada é composta pelo Alinhamento Múltiplo de Sequências (MSA) de cada proteína do complexo e pelos arquivos de coordenadas dos modelos putativos, que são gerados pelos algoritmos de *docking* (**Figura 10, Letra a**) . Em seguida, há uma validação desses arquivos de entrada através da verificação do tamanho dos alinhamentos MSA^A e MSA^B, da similaridade entre as sequências de aminoácidos de todos os modelos e também da similaridade entre as sequências de aminoácidos das proteínas e a primeira sequência de seus respectivos alinhamentos (**Figura 10, Letra b**). Se os os arquivos forem correspondentes, são construídos sub-alinhamentos para cada modelo a partir das posições dos aminoácidos de interface (**Figura 10, Letra c**), como descrito na Seção III.3. Estes sub-alinhamentos são usados para calcular o valor de MI e r de cada modelo (**Figura 10, Letra d**), explicitados na Seção III.4. Decorridos os cálculos, são gerados um relatório contendo os dados brutos dos resultados e um gráfico com os valores das métricas calculadas para cada modelo (**Figura 10, Letra e**), que, por sua vez, são retornados ao usuário (**Figura 10, Letra f**).

Utilizou-se a classe PDB_Parser para a leitura de arquivos de coordenadas ".pdb" e a classe AlignIO para a leitura e tratamento de arquivos fasta, ambas do pacote Biopython 1.72[60,61], uma das dependências do *software*. Após a construção dos sub-alinhamentos e a realização dos cálculos, utilizamos o pacote PyQtGraph para a plotagem dos resultados, devido à sua interação com o PyQT 5.4, utilizado para desenvolver a interface gráfica. O programa foi desenvolvido utilizando a linguagem Pyhton 3.5 devido a sua grande versatilidade no tratamento de *strings* e arquivos, além da facilidade na manipulação de matrizes que a biblioteca Scipy[62] oferece.

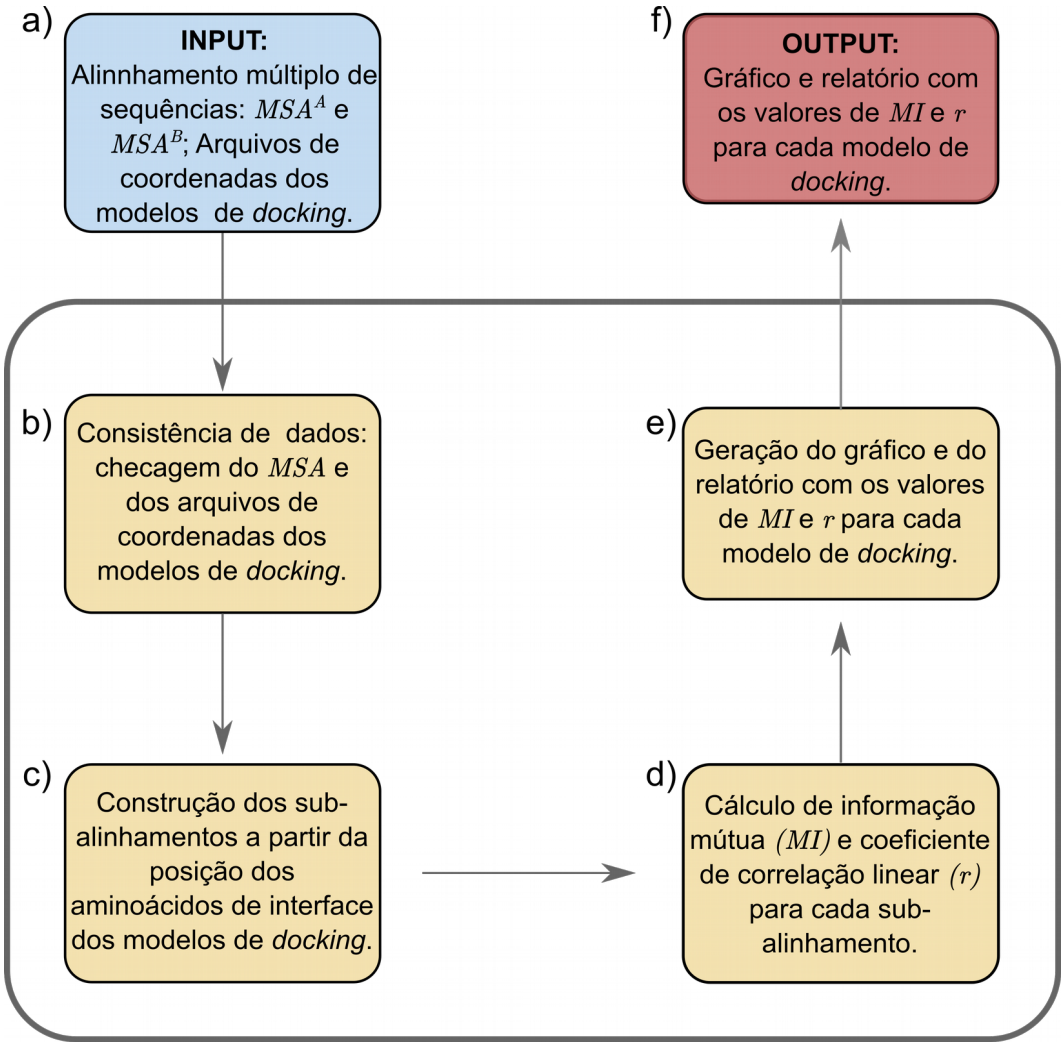

Figura 10: Esquema geral das etapas do software.

IV – Resultados e Discussão

IV.1 – Impacto do Valor de de Similaridade de Sequências e da Pseudocontagem no Cálculo de *MI*

Como referido anteriormente, as proteínas dos complexos citados na **Tabela 1** foram submetidas a quatro diferentes servidores de *docking* (GRAMMX, HDOCK, ROSIE e ZDOCK) e cada um desses servidores gerou dez modelos putativos para estes complexos. Calculou-se o valor de informação mútua (MI) dos contatos de interface de cada um dos modelos utilizando os sub-alinhamentos gerados a partir dos MSAs das proteínas do complexo. O cálculo de MI leva em conta as frequências simples e conjuntas de aminoácidos nas colunas do MSA. No entanto, como pode haver um viés de amostragem de aminoácidos nas sequências homólogas, utiliza-se uma pseudocontagem (λ) para atribuir um valor aos aminoácidos não amostrados. Além disso, o cálculo também leva em conta o número de sequências efetivas (M_{eff}) para a repesagem do valor de cada sequência homóloga. Esse número é inferido por meio de um valor de similaridade de sequências (θ). Para inferir os valores a serem usados no trabalho, calculou-se os valores de MI/H dos modelos putativos do complexo 1BXR A-B para os valores de θ = 0,8, e 1,0, e para os valores de λ = 0,5 e 0,1. Estes valores foram relacionados à quantidade de contatos nativos e estão representados na forma de gráfico na **Figura 11**.

A partir da observação dos gráficos, é possível perceber que o parâmetro θ é o mais relevante para a linearidade da relação entre os valores de MI/H e número de contatos nativos dos modelos. Quando θ = 1, o M_{eff} é igual ao número de sequências do MSA de cada proteína. As sequências homólogas podem apresentar um viés de amostragem que pode resultar da relação filogenética entre as espécies de onde provém as sequências, ou ainda devido ao sequenciamento de cepas de uma mesma espécie. Por sua vez, quando θ = 0,8, esse viés é minimizado e a informação evolutiva pode ser melhor revelada. Então, o valor de θ escolhido para todos os casos estudados neste trabalho foi de 0,8, o que é condizente com os valores utilizados na literatura[52].

O valor de pseudocontagem λ tem menor influência na linearidade dos resultados, no entanto, quando λ = 0,5, há um maior separação entre os valores de MI/H entre os modelos, quando comparado ao λ = 0,1. Esta diferença pode ser significativa no algoritmo de classificação de modelos de *docking*. Portanto, os estudos de caso desse trabalho foram feitos utilizando um valor de pseudocontagem λ igual a 0,5.

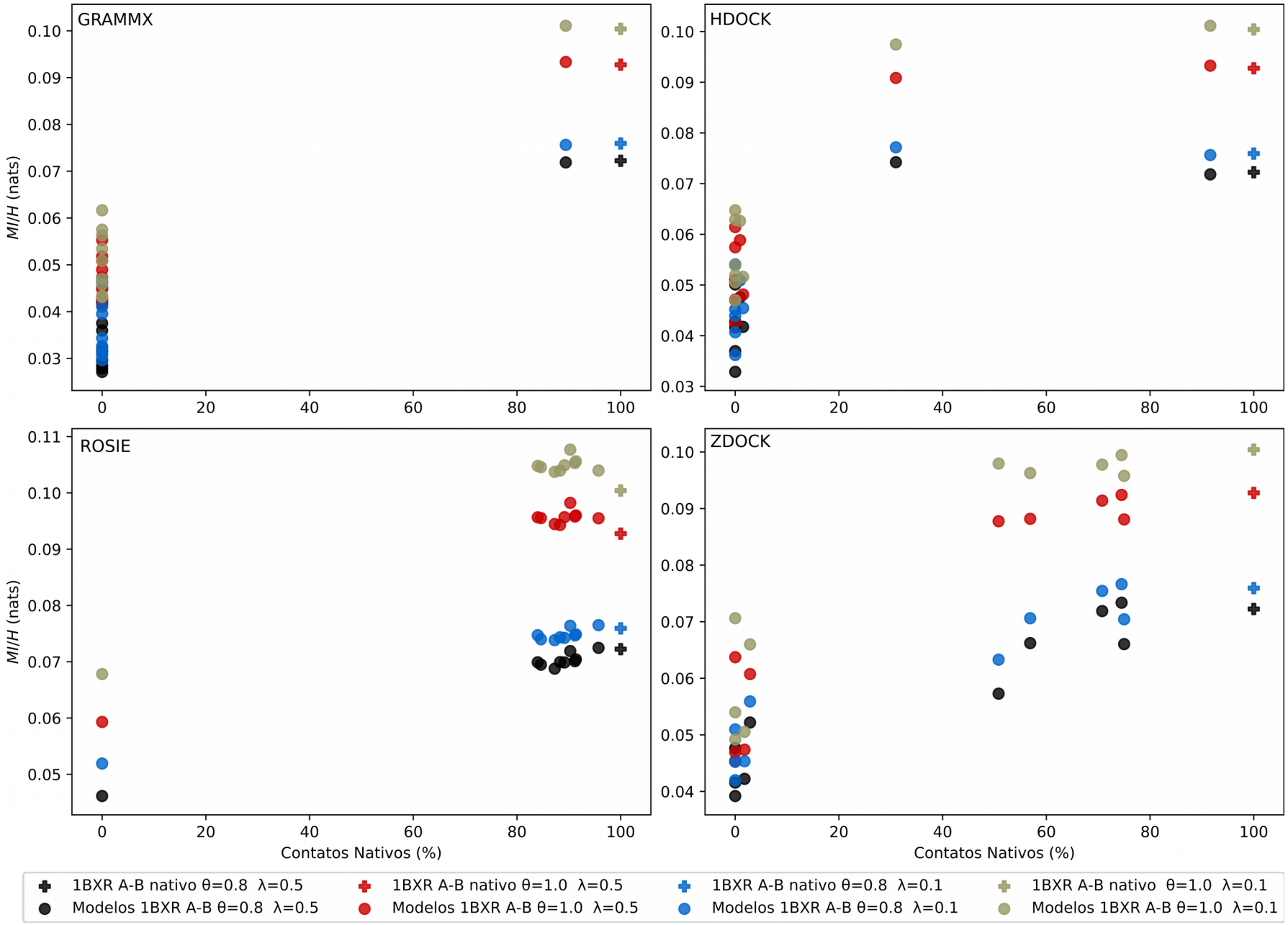

Figura 11: Relação entre valores de MI/H e porcentagem de contatos nativos do complexo 1BXR A-B e seus modelos putativos utilizando diferentes valores de θ e λ.

IV.2 – Aliando a Filogenia à Incerteza Contida nas Posições do *MSA*

Foram calculados os valores de MI/H e r para todos os sistemas descritos na **Tabela 1** e para os seus respectivos modelos gerados pelos algoritmos de *docking*. Esses valores, isolados ou associados, foram relacionados aos valores do desvio estrutural $(rmsd)$ do modelo em relação ao complexo nativo e à porcentagem de contatos nativos estabelecidos na interface do modelo, que são os contatos idênticos aos estabelecidos na interface da estrutura alvo. Os resultados obtidos estão representados graficamente nas **Figuras 12 – 20, 21 e 23** e registradas de maneira integral nas **Tabelas 4 – 14** localizadas no Anexo I.

IV.2.1 – Proteínas Globulares

A partir dos resultados referentes aos complexos 1BXR A-B **(Figura 12)**, 1EP3 A-B **(Figura 13)**, 1TYG A-B **(Figura 14)**, 2VPZ A-B **(Figura 15)**, 2Y69 A-B **(Figura 16)**, 1OYH I-L **(Figura 17)**, 1VET A-B **(Figura 18)**, 3ZET A-B **(Figura 19**) e 5F5S A-B **(Figura 20)**, é possível inferir que os valores de MI/H e r, quando tomados isoladamente, parecem não ser bons descritores da proximidade entre um modelo putativo e sua estrutura alvo. Em vários gráficos é possível ver que modelos mais próximos à estrutura nativa possuem altos valores de MI/H ou de r, mas não conseguimos diferenciá-los dos modelos com maior desvio estrutural. No entanto, quando consideramos o produto dos valores destas duas métricas $(MI/H \times r)$, vemos que há, para a maioria os casos estudados nesse trabalho, uma relação direta desse valor com a porcentagem de contatos nativos estabelecidos pelo modelo e também com o seu valor de $rmsd$ em relação à estrutura nativa.

Estes valores de coevolução estão intimamente conectados à função do complexo proteico. As funções fisiológicas impõem restrições à estrutura das proteínas, determinando a transitoriedade e obrigatoriedade de suas interações, assim, influenciando as suas taxas de evolução[55],[64]. Muitos dos complexos descritos na **Tabela 1** têm função enzimática e participam de vias de síntese de

biomoléculas importantes para a manutenção das espécies das quais elas provém[65–69]. A pressão seletiva pode, então, favorecer fortemente o equilíbrio da interação proteica nesses complexos, acoplando as mutações entre os aminoácidos da interface.

Para o caso das proteínas globulares, a combinação das métricas de coevolução se mostra efetiva por que somente a região que faz parte da interface do complexo é conservada. Assim, há uma pressão seletiva que acopla somente as mutações dos aminoácidos que estão em uma região da casca das proteínas.

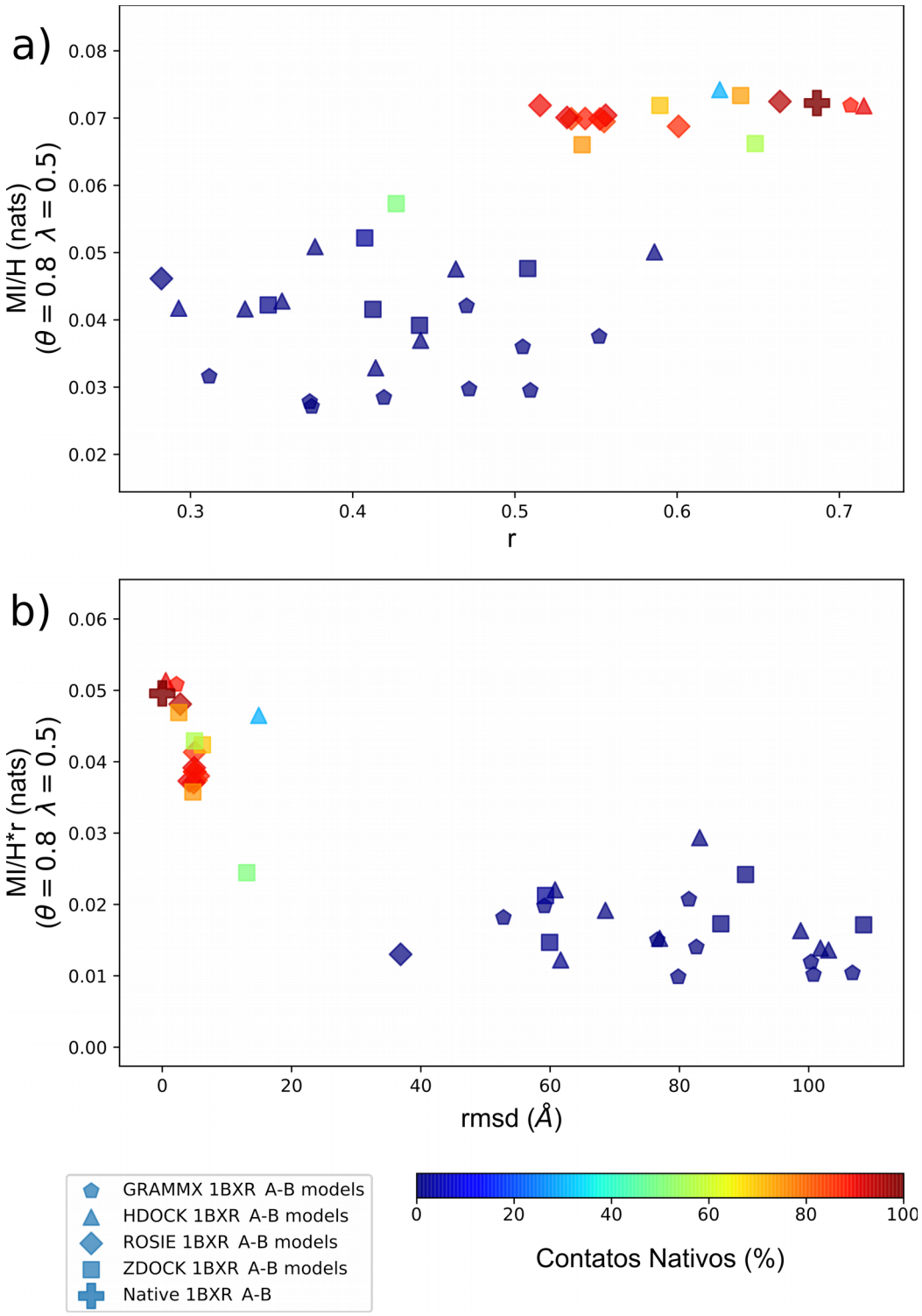

Figura 12: Resultados dos cálculos para o complexo 1BXR A-B. a) valores de MI/H, r e porcentagem de contatos nativos para o complexo 1BXR A-B e seus modelos putativos. **b)** valores do produto de MI/H e r , $rmsd$ em relação à estrutura nativa e porcentagem de contatos nativos para o complexo 1BXR A-B e seus modelos putativos. A cor de cada ponto representa a porcentagem de contatos nativos, enquanto sua forma representa o servidor de *docking* no qual ele foi gerado. Estes resultados estão integralmente registrados na **Tabela 4**, no Anexo I.

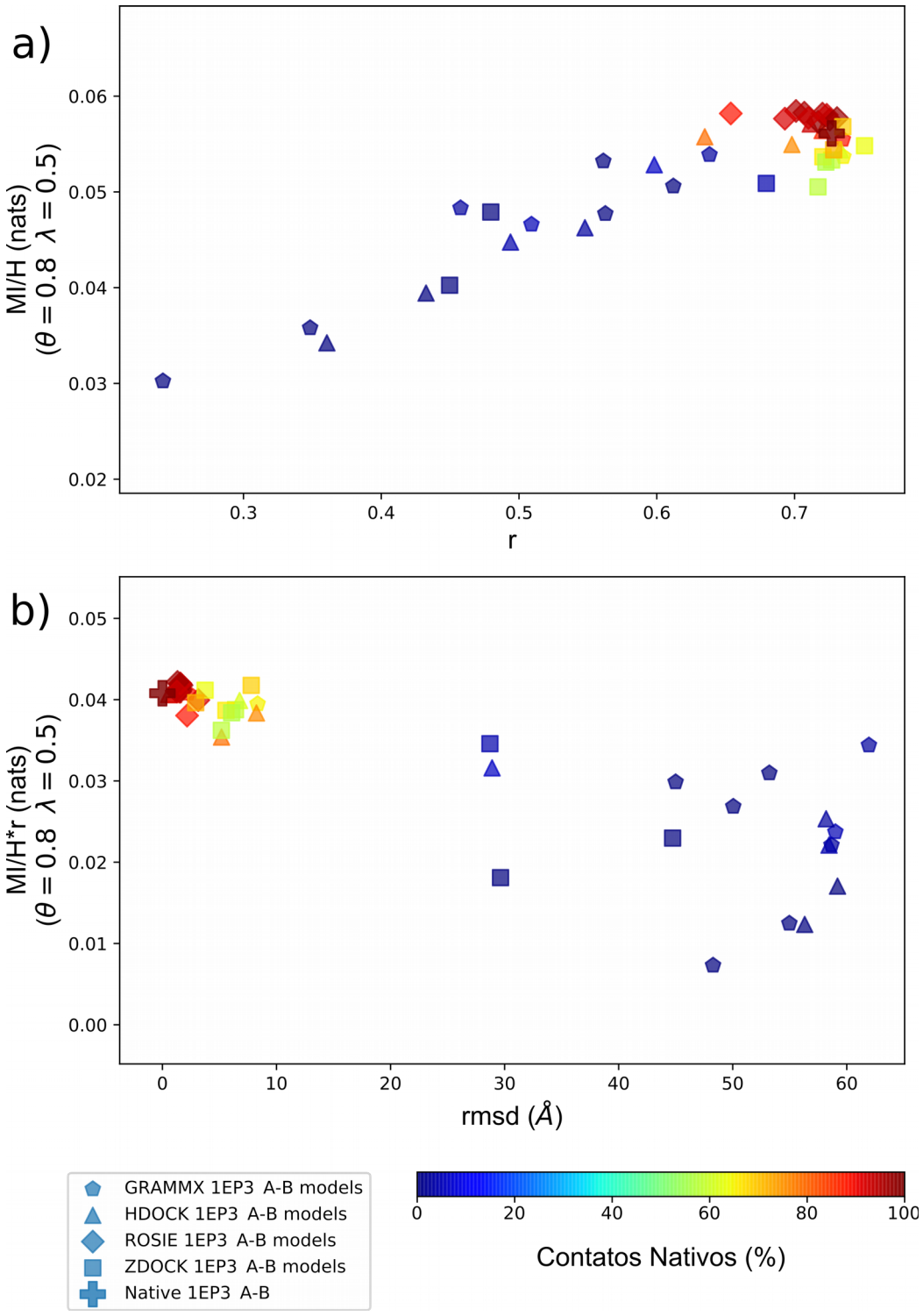

Figura 13: Resultados dos cálculos para o complexo 1EP3 A-B. a) valores de MI/H, r e porcentagem de contatos nativos para o complexo 1EP3 A-B e seus modelos putativos. **b)** valores do produto de MI/H e r , $rmsd$ em relação à estrutura nativa e porcentagem de contatos nativos para o complexo 1EP3 A-B e seus modelos putativos. A cor de cada ponto representa a porcentagem de contatos nativos, enquanto sua forma representa o servidor de *docking* no qual ele foi gerado. Estes resultados estão integralmente registrados na **Tabela 5**, no Anexo I.

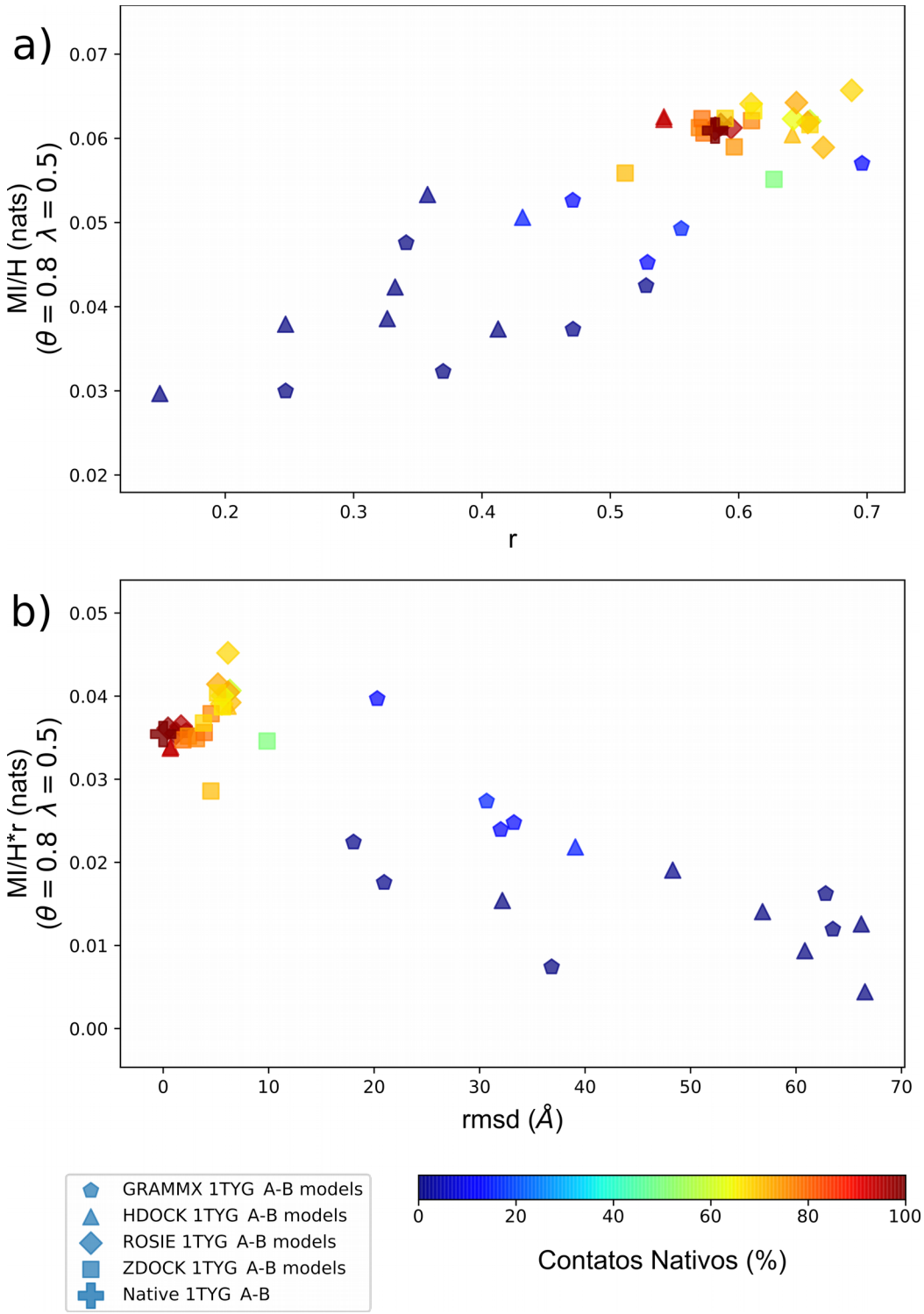

Figura 14: Resultados dos cálculos para o complexo 1TYG A-B. a) valores de MI/H, r e porcentagem de contatos nativos para o complexo 1TYG A-B e seus modelos putativos. **b)** valores do produto de MI/H e r , $rmsd$ em relação à estrutura nativa e porcentagem de contatos nativos para o complexo 1TYG A-B e seus modelos putativos. A cor de cada ponto representa a porcentagem de contatos nativos, enquanto sua forma representa o servidor de *docking* no qual ele foi gerado. Estes resultados estão integralmente registrados na **Tabela 6**, no Anexo I.

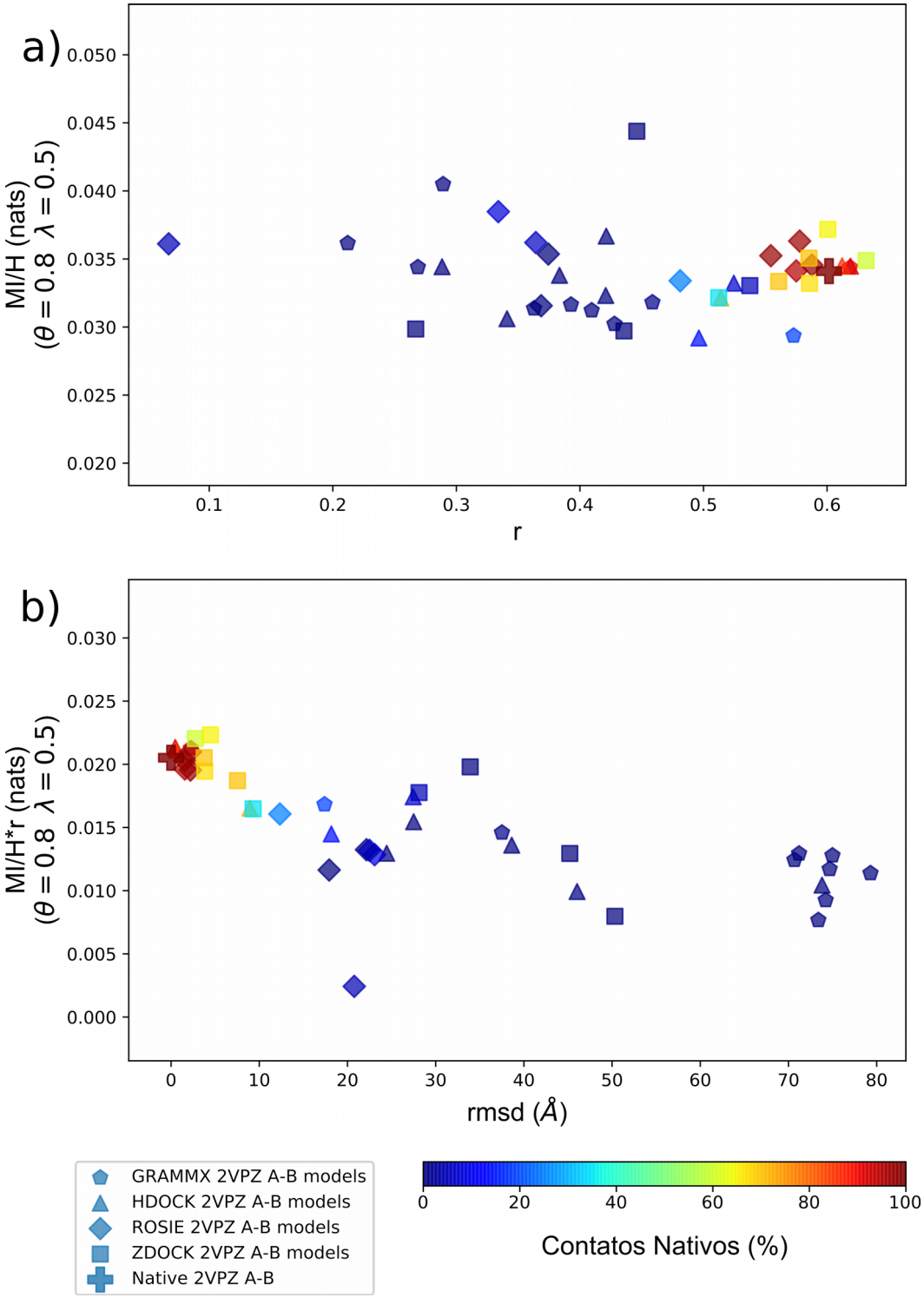

Figura 15: Resultados dos cálculos para o complexo 2VPZ A-B. a) valores de MI/H, r e porcentagem de contatos nativos para o complexo 2VPZ A-B e seus modelos putativos. **b)** valores do produto de MI/H e r , $rmsd$ em relação à estrutura nativa e porcentagem de contatos nativos para o complexo 2VPZ A-B e seus modelos putativos. A cor de cada ponto representa a porcentagem de contatos nativos, enquanto sua forma representa o servidor de *docking* no qual ele foi gerado. Estes resultados estão integralmente registrados na **Tabela 7**, no Anexo I.

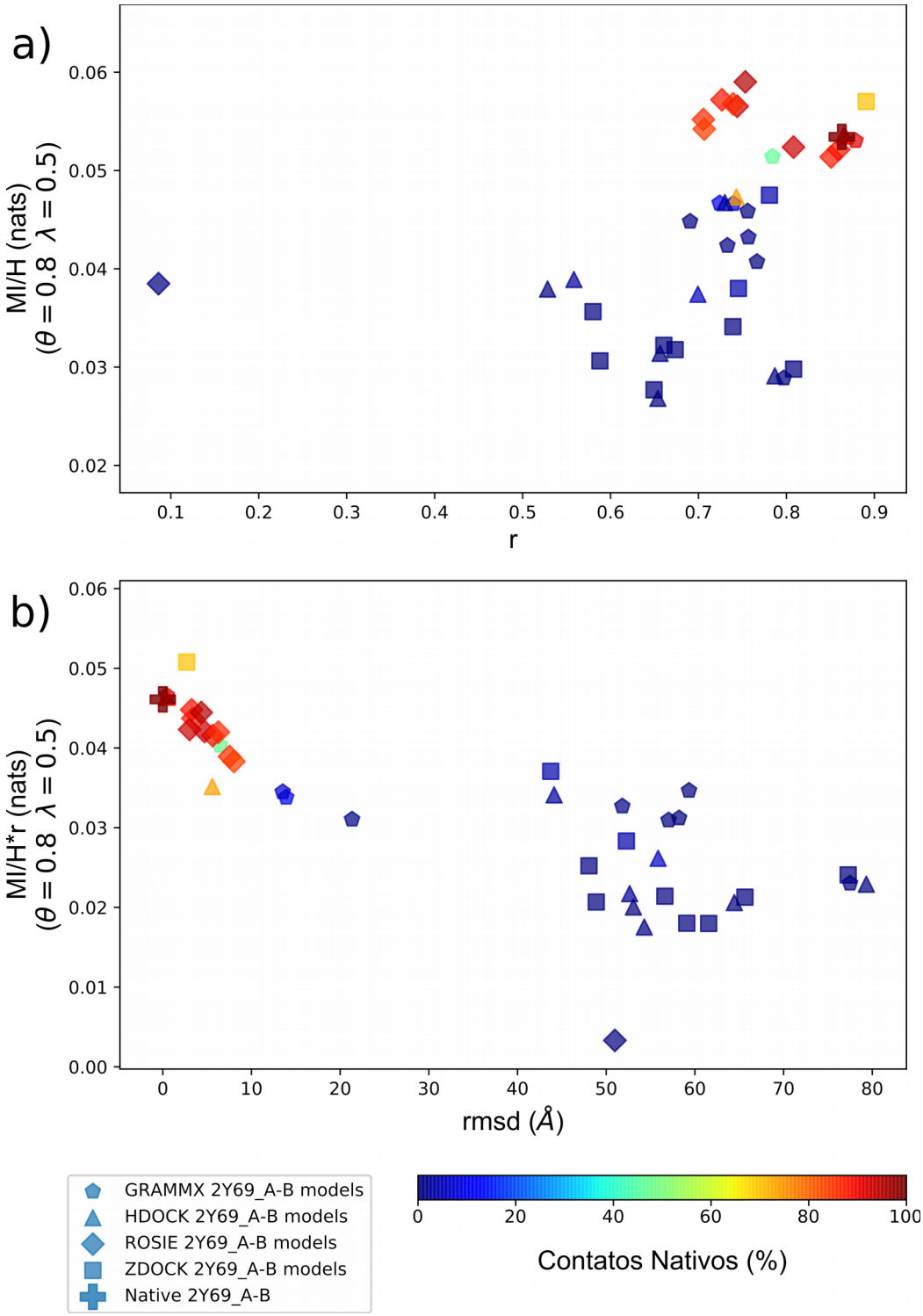

Figura 16: Resultados dos cálculos para o complexo 2Y69 A-B. a) valores de MI/H, r e porcentagem de contatos nativos para o complexo 2Y69 A-B e seus modelos putativos. **b)** valores do produto de MI/H e r , $rmsd$ em relação à estrutura nativa e porcentagem de contatos nativos para o complexo 2Y69 A-B e seus modelos putativos. A cor de cada ponto representa a porcentagem de contatos nativos, enquanto sua forma representa o servidor de *docking* no qual ele foi gerado. Estes resultados estão integralmente registrados na **Tabela 8**, no Anexo I.

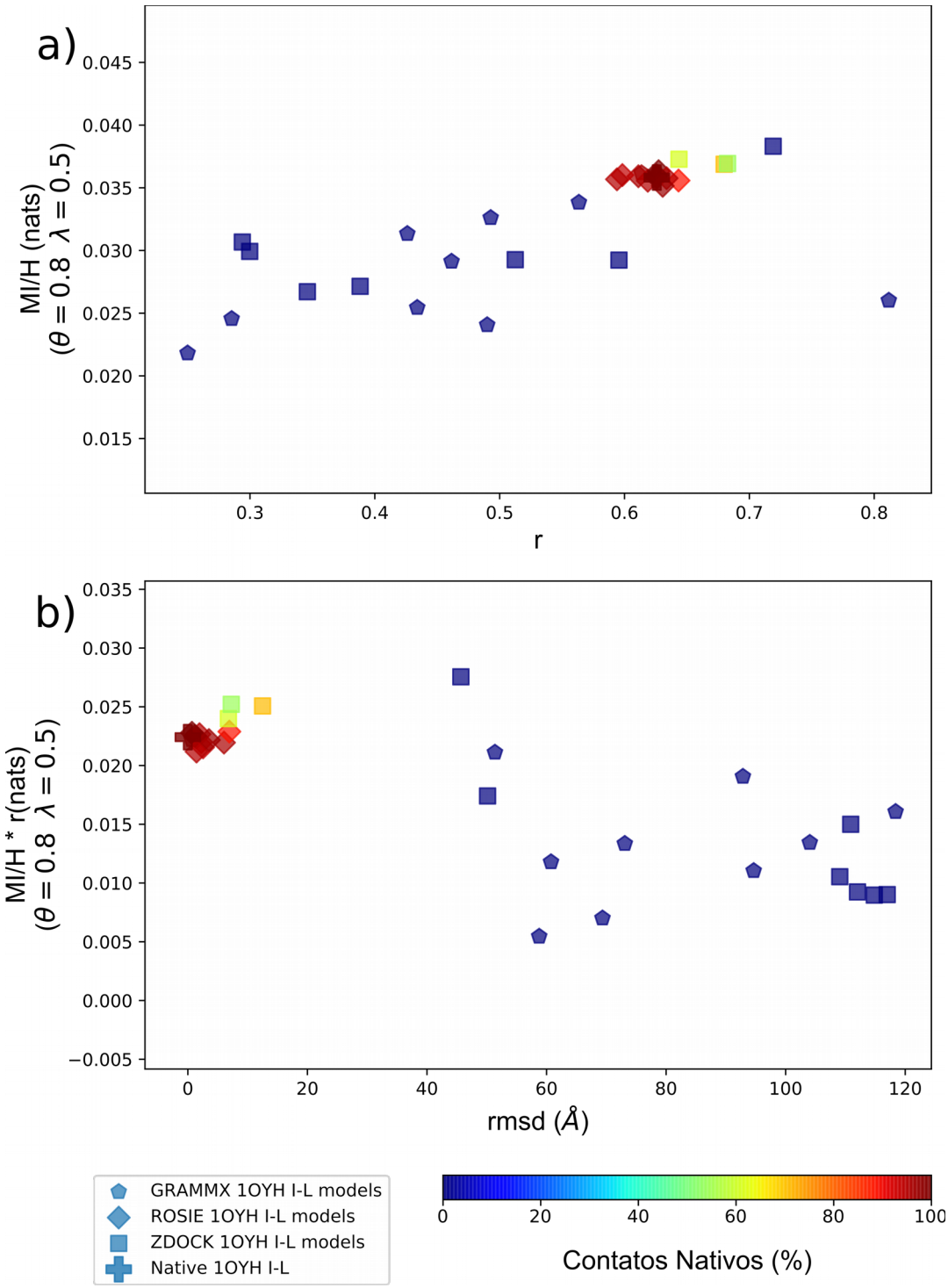

Figura 17: Resultados dos cálculos para o complexo 1OYH I-L. **a)** valores de MI/H, r e porcentagem de contatos nativos para o complexo 1OYH I-L e seus modelos putativos. **b)** valores do produto de MI/H e r , $rmsd$ em relação à estrutura nativa e porcentagem de contatos nativos para o complexo 1OYH I-L e seus modelos putativos. A cor de cada ponto representa a porcentagem de contatos nativos, enquanto sua forma representa o servidor de *docking* no qual ele foi gerado. Estes resultados estão integralmente registrados na **Tabela 9**, no Anexo I.

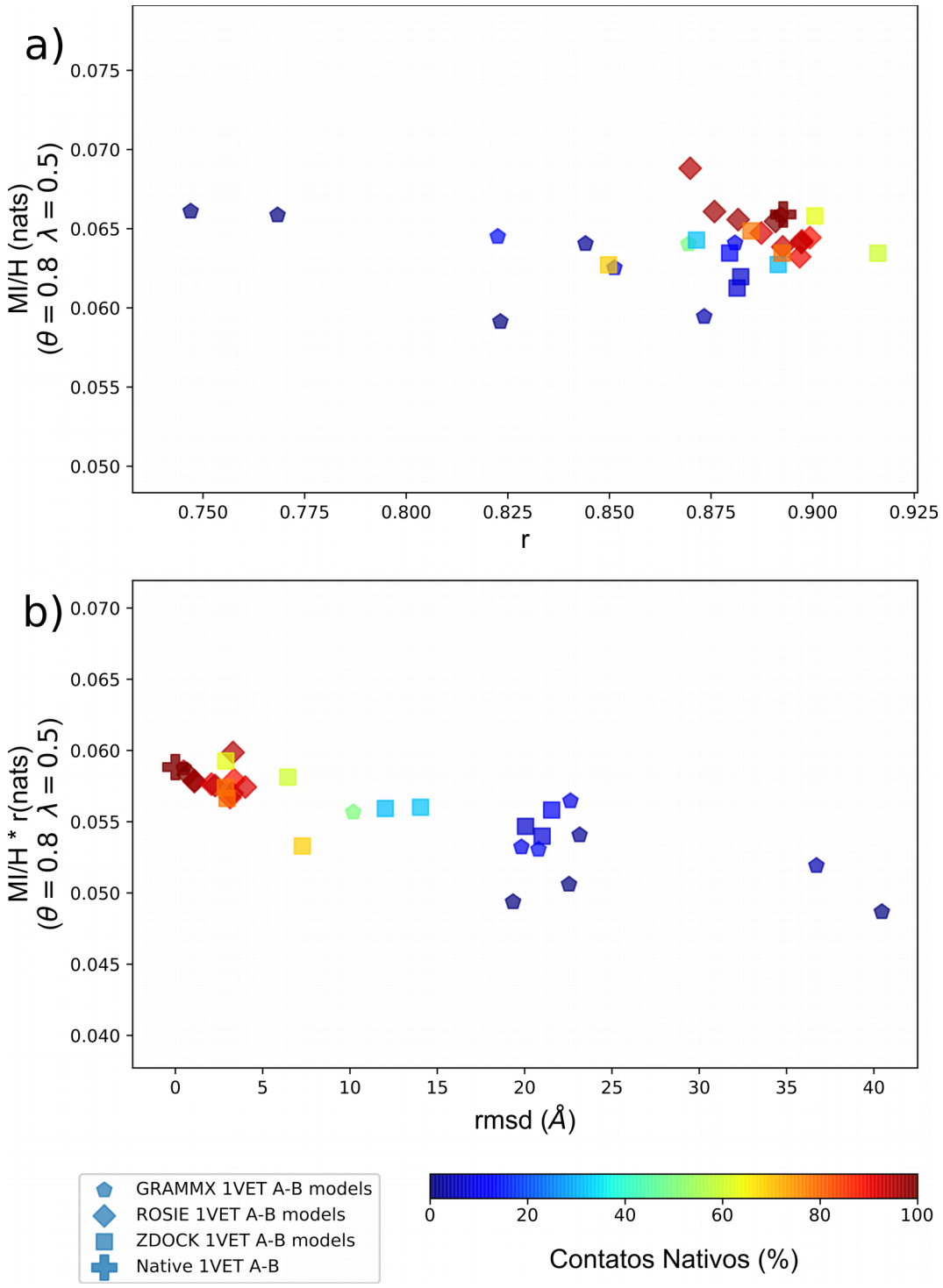

Figura 18: Resultados dos cálculos para o complexo 1VET A-B. a) valores de MI/H, r e porcentagem de contatos nativos para o complexo 1VET A-B e seus modelos putativos. **b)** valores do produto de MI/H e r , $rmsd$ em relação à estrutura nativa e porcentagem de contatos nativos para o complexo 1VET A-B e seus modelos putativos. A cor de cada ponto representa a porcentagem de contatos nativos, enquanto sua forma representa o servidor de *docking* no qual ele foi gerado. Estes resultados estão integralmente registrados na **Tabela 10**, no Anexo I.

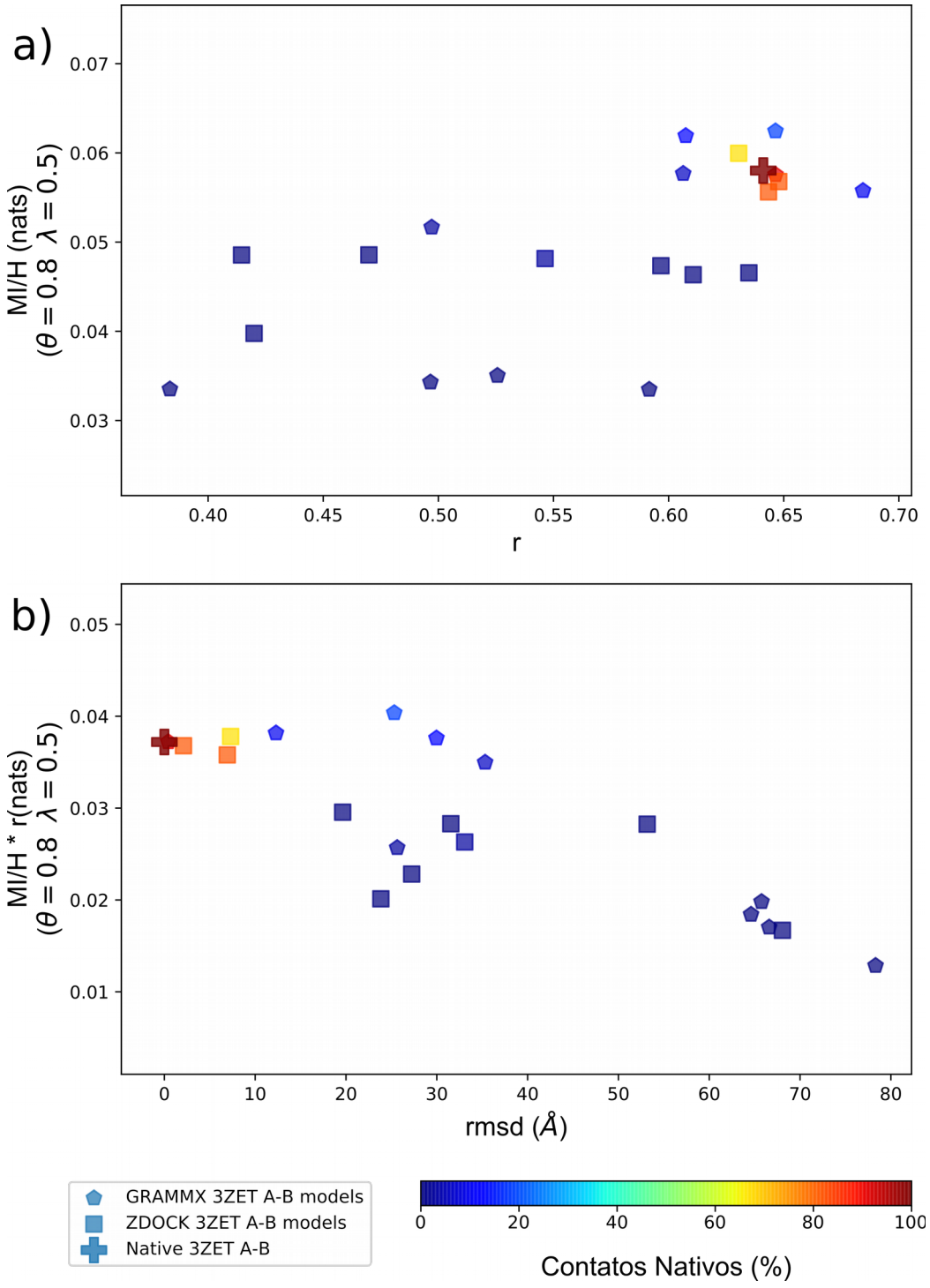

Figura 19: Resultados dos cálculos para o complexo 3ZET A-B. a) valores de MI/H, r e porcentagem de contatos nativos para o complexo 3ZET A-B e seus modelos putativos. **b)** valores do produto de MI/H e r , $rmsd$ em relação à estrutura nativa e porcentagem de contatos nativos para o complexo 3ZET A-B e seus modelos putativos. A cor de cada ponto representa a porcentagem de contatos nativos, enquanto sua forma representa o servidor de *docking* no qual ele foi gerado. Estes resultados estão integralmente registrados na **Tabela 11**, no Anexo I.

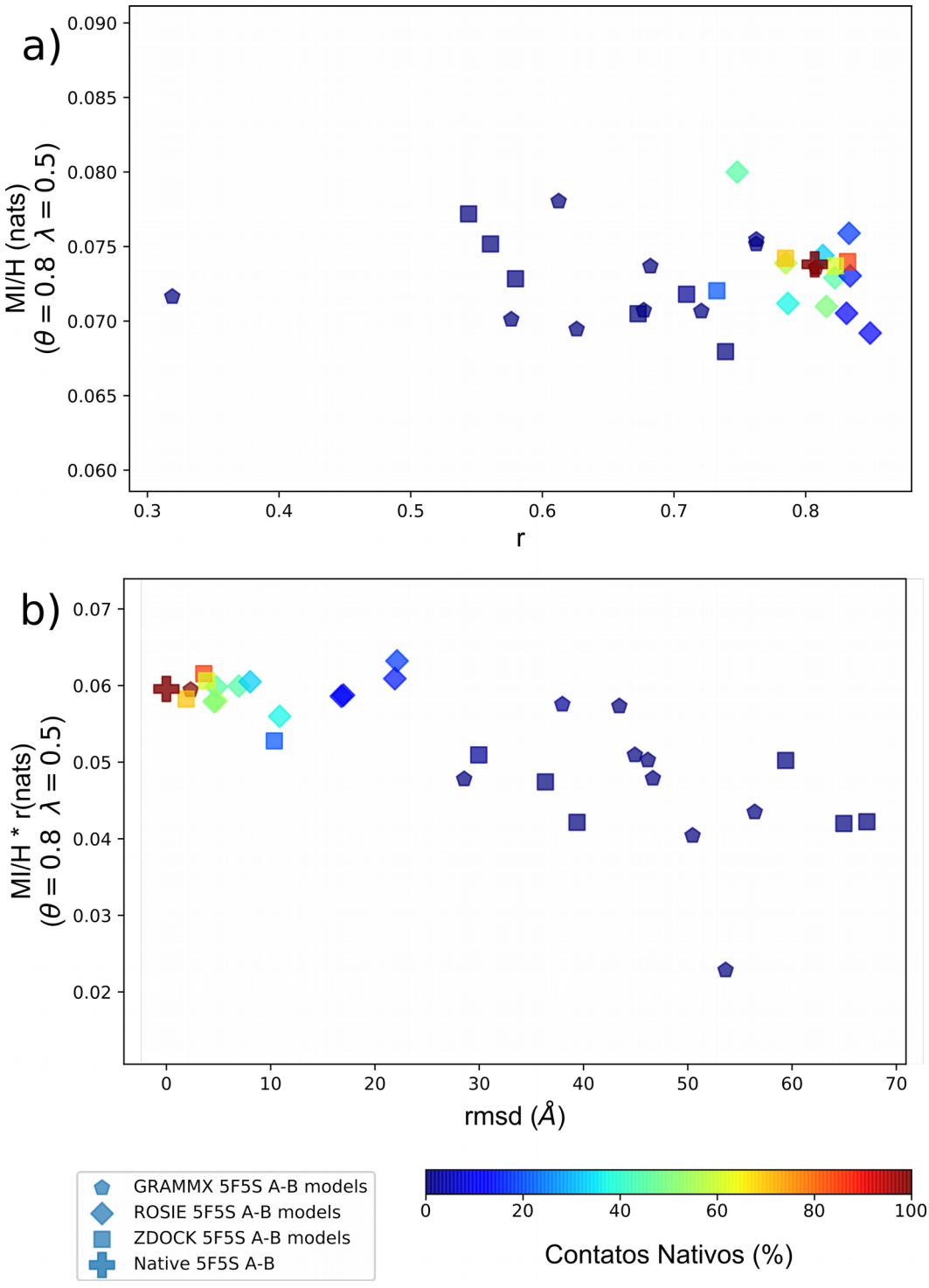

Figura 20: Resultados dos cálculos para o complexo 5F5S A-B. a) valores de MI/H, r e porcentagem de contatos nativos para o complexo 5F5S A-B e seus modelos putativos. **b)** valores do produto de MI/He r , $rmsd$ em relação à estrutura nativa e porcentagem de contatos nativos para o complexo 5F5S A-B e seus modelos putativos. A cor de cada ponto representa a porcentagem de contatos nativos, enquanto sua forma representa o servidor de *docking* no qual ele foi gerado. Estes resultados estão integralmente registrados na **Tabela 12**, no Anexo I.

IV.2.2 – Casos Sensíveis

Alguns modelos que possuem baixo desvio estrutural não puderam ser discriminados mesmo utilizando o produto das duas métricas de coevolução. Este foi o caso dos modelos putativos do complexo 3OAA G-H **(Figura 21)**. O complexo 3OAA G-H é formado pelas unidades γ e ϵ da F1 ATP–sintase de *E.coli*. Como visto na **Figura 22**, essas subunidades também estão em contato com outras proteínas quando encontradas na natureza. Sendo a F1 ATP–sintase um complexo altamente conservado, várias regiões da casca dessas subunidades γ e ϵ também são conservadas. Assim, não é possível discernir o sinal coevolutivo que resulta da ligação das subunidades γ e ϵ do sinal que vem da sua interação com as outras proteínas da F1 ATP–sintase.

Outro caso é o do complexo 2Y69 A-C **(Figura 23)**, formado pelas unidades 1 e 3 do complexo transmembrânico citocromo C-oxidase encontrado em bovinos. Apesar de pertencer à mesma estrutura do complexo 2Y69 A-B **(Figura 16)**, que apresentou bons resultados, a interação entre as cadeias A e C acontece em meio a membrana fosfolipídica **Figura 24**, que, por sua vez, impõe restrições à natureza química dos aminoácidos que estão na casca das proteínas. Também aqui é difícil separar a informação evolutiva que decorre da interação das cadeias A e C da informação que resulta da interação das cadeias com a membrana.

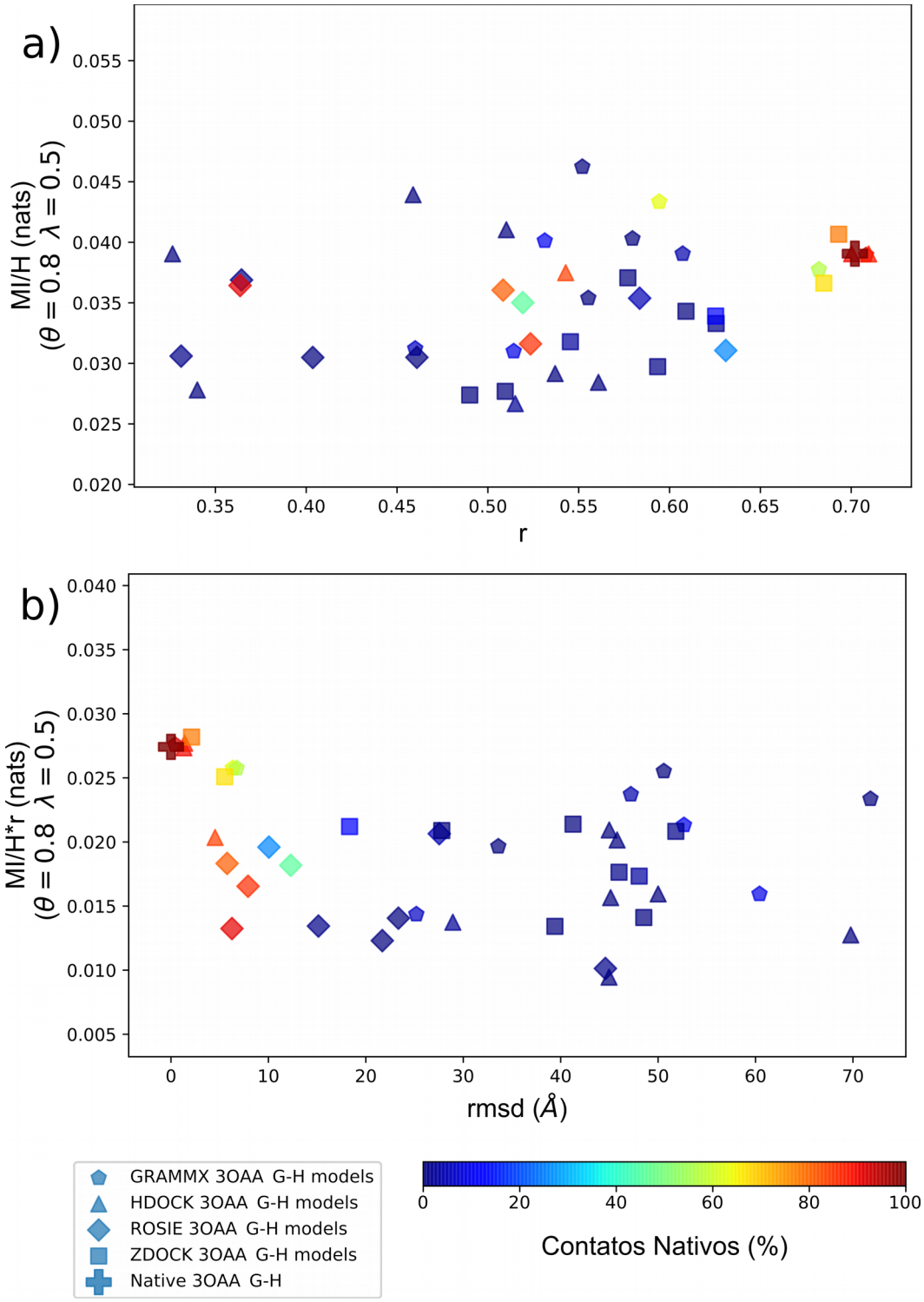

Figura 21: Resultados dos cálculos para o complexo 3OAA G-H . a) valores de MI/H, r e porcentagem de contatos nativos para o complexo 3OAA G-H e seus modelos putativos. **b)** valores do produto de MI/He r , $rmsd$ em relação à estrutura nativa e porcentagem de contatos nativos para o complexo 3OAA G-H e seus modelos putativos. A cor de cada ponto representa a porcentagem de contatos nativos, enquanto sua forma representa o servidor de *docking* no qual ele foi gerado. Estes resultados estão integralmente registrados na **Tabela 13**, no Anexo I.

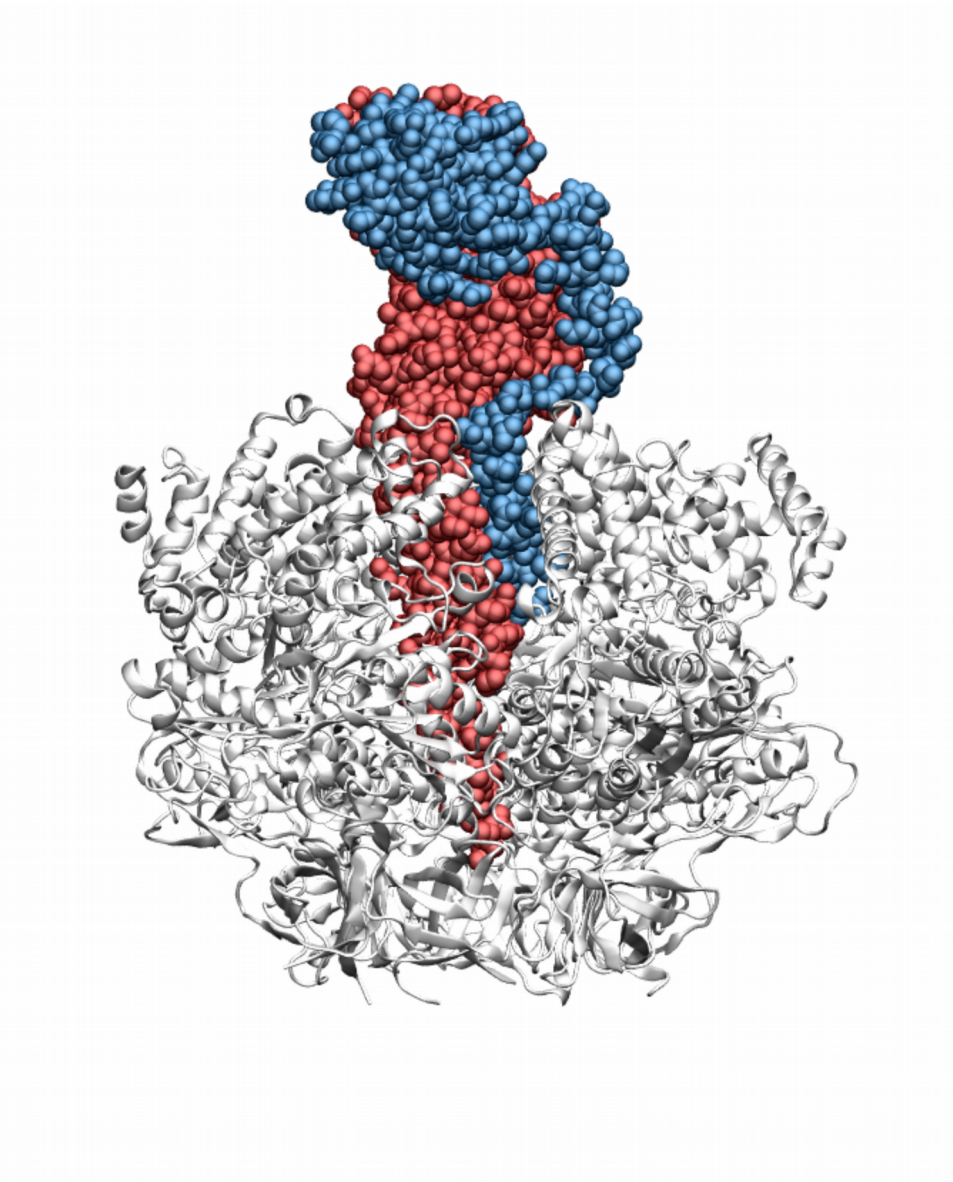

Figura 22: Representação das subunidades ϵ (em azul) e γ (em vermelho) no complexo proteico F1 ATP–sintase de *E.coli*.

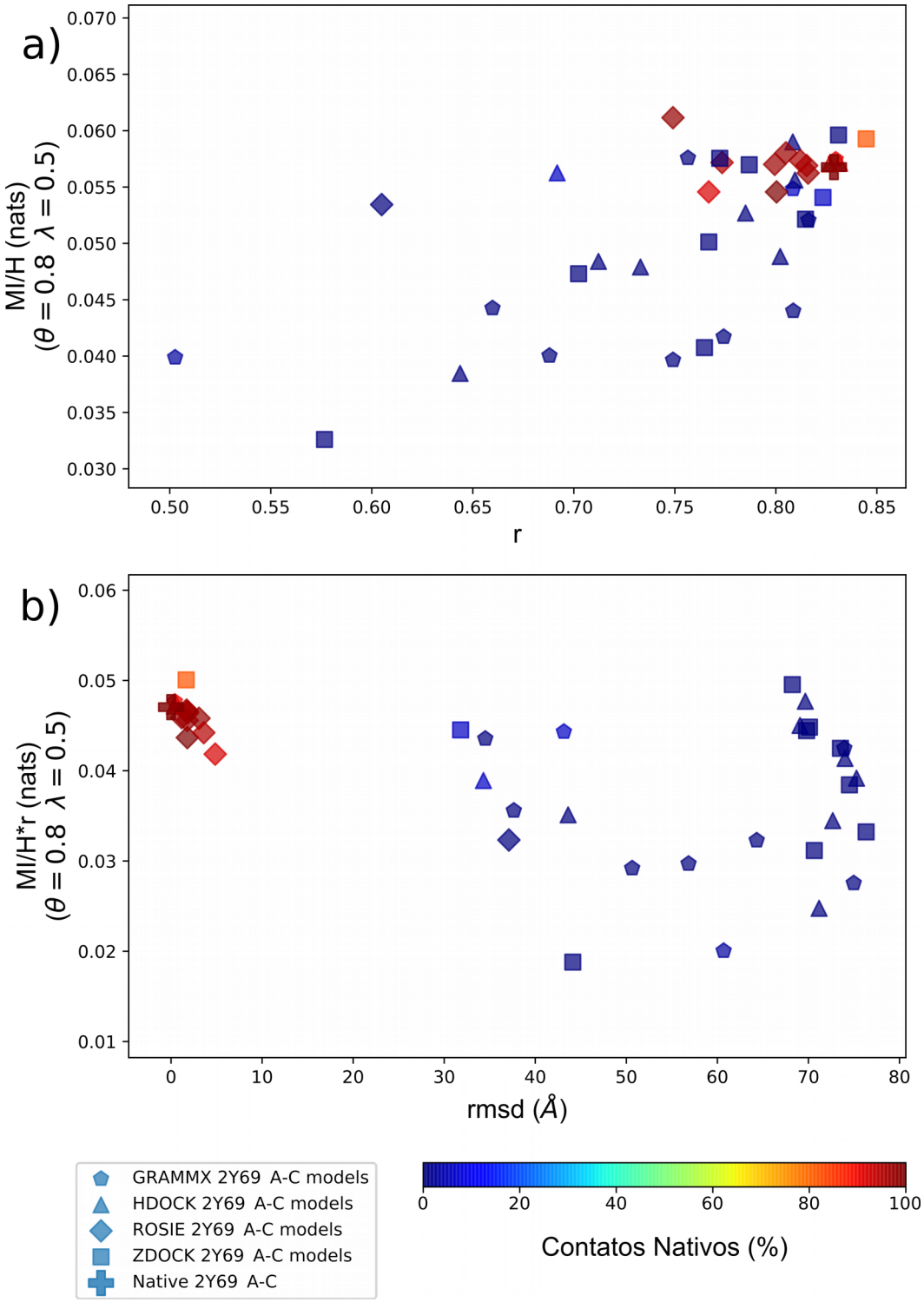

Figura 23: Resultados dos cálculos para o complexo 2Y69 A-C . a) valores de MI/H, r e porcentagem de contatos nativos para o complexo 2Y69 A-C e seus modelos putativos. **b)** valores do produto de MI/He r , $rmsd$ em relação à estrutura nativa e porcentagem de contatos nativos para o complexo 2Y69 A-C e seus modelos putativos. A cor de cada ponto representa a porcentagem de contatos nativos, enquanto sua forma representa o servidor de *docking* no qual ele foi gerado. Estes resultados estão integralmente registrados na **Tabela 14**, no Anexo I.

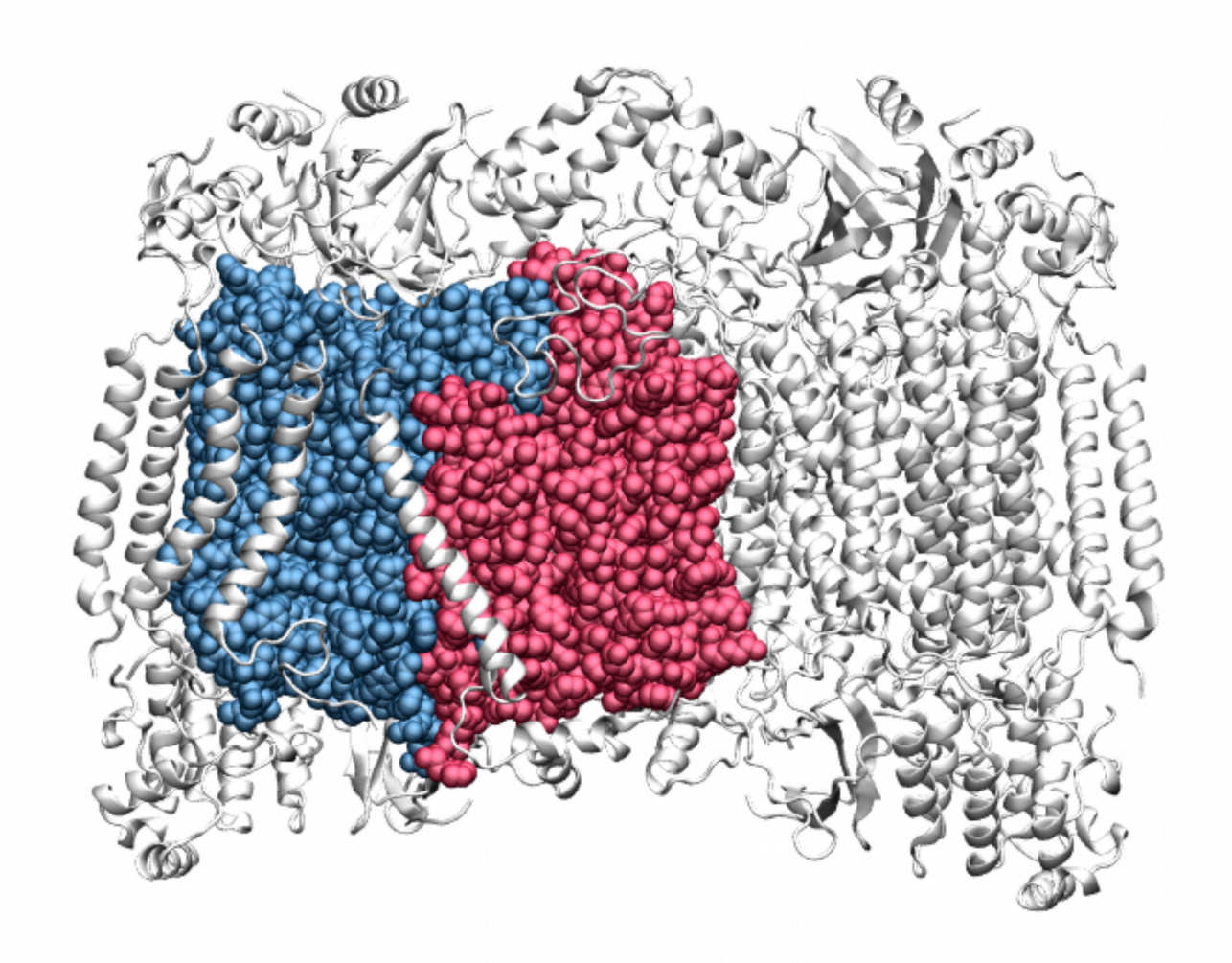

Figura 24: Representação das subunidades 1 (em azul) e 3 (em vermelho) no complexo transmembrânico citocromo C-oxidase.

Outro ponto a ser considerado durante as análises de coevolução, além da estrutura e função das proteínas, é a qualidade do MSA. O alinhamento deve ser feito de modo que possamos recuperar as informações coevolutivas nele implícitas. Como o as análises de coevolução utilizam modelos estatísticos, o tamanho do MSA contribui diretamente para a acurácia da predição. Em um trabalho de 2013[70], Kamisetty e colaboradores sugerem que uma boa predição baseia-se em um número de sequências cinco vezes maior que o número de pares estabelecidos. O número de sequências total, o número de sequências efetivas (M_{eff}) (equação [15]) e a média de pares para os modelos de cada complexo estão registrados na **Tabela 4**. Para os complexos desse trabalho, levando em conta o M_{eff}, os modelos do complexos 2Y69 A-C , 1VET A-B, 3ZET A-B e 5F5S A-B não obedecem a esta razão. No entanto, o insuficiente número de sequências ainda não explica bem a falha na distinção dos modelos, visto o sucesso na distinção de modelos mais próximos ao nativo nos complexos 1VET A-B, 3ZET A-B e 5F5S A-B, e a falha no caso do complexo 3OAA G-H, que apresenta uma razão entre M_{eff} e número médio de pares superior a cinco.

Tabela 3: Dados dos MSAs de cada complexo. Para cada complexo, estão listados o número de sequências presentes no MSA de cada proteína, o M_{eff} para o θ = 0,8, a média de pares de contato dos modelos putativos com $r_C = 0.8\mathring{A}$ e a razão entre o M_{eff} e o número médio de pares de contato.

Complexo	Número de sequências	Meff	Número médio de pares de contato dos modelos putativos	Meff / Número médio de Pares
1BXR A-B	1004	492,1	96,6	5,095
1EP3 A-B	552	436,2	84,4	5,168
1TYG A-B	746	479,2	69,8	6,866
2VPZ A-B	676	437,9	86,2	5,080
2Y69 A-B	1484	712,2	128,6	5,538
2Y69 A-C	863	403,0	123,6	3,260
3OAA G-H	886	603,4	86,8	6,952
1OYH I-L	713	574,1	82,3	6,975
1VET A-B	264	64,4	86,1	0,748
3ZET A-B	363	197,6	82,0	2,410
5F5S A-B	326	49,1	34,5	1,424

IV.3 –*Docking Score Module*

O *software* desenvolvido recebeu o título de *Docking Score Module* (DSM) e apresenta uma interface que possibilita ao usuário selecionar diretamente os arquivos de coordenadas gerados por programas de *docking* e os arquivos de alinhamentos através da navegação entre diretórios. O código fonte do programa encontra-se no Anexo II, juntamente com o UML das classes do programa **(Figura 34)**. Em seguida são apresentadas as janelas do DSM em etapas sequenciais de utilização. Os registros foram feitos durante os cálculos dos modelos do complexo 1BXR A-B gerados pelo servidor GRAMMX.

A **Figura 25** mostra a janela inicial do DSM após a sua iniciação. Os aquivos referentes às coordenadas dos modelos de *docking* e com extensão ".pdb" são selecionados pelo botão "select" **(Figura 25, marcação em vermelho 1)**, que abre uma janela de navegação entre diretórios. Os botões "select" **(Figura 25, marcações em vermelho 2 e 3)** também abrem janelas de navegação entre diretórios para a seleção do arquivos com extensão ".fas" referentes ao alinhamentos múltiplos de sequências das proteínas A e B. Após isso, utiliza-se a caixa de texto **(Figura 25, marcação em vermelho 4)** para inserção de um nome para a identificação do trabalho e dos arquivos de saída. Caso não seja preenchido, o identificador é composto pela palavra "*job"* juntamente com a data e o horário em que o trabalho (execução dos cálculos das análises de coevolução) foi iniciado. Em seguida, atribui-se um valor à pseudocontagem lambda (λ) utilizando a caixa de seleção **(Figura 25, marcação em vermelho 5)**. Esse valor é utilizado no cálculo de frequência dos aminoácidos (equações [13] e [16]). O intervalo se seleção do λ vai de 0,1 à 1,0 em incrementos de 0,1, e seu valor padrão é igual a 0,5. Também atribui-se um valor de similaridade de sequência (θ) (equação [14]), utilizando a caixa de seleção **(Figura 25, marcação em vermelho 6)**. Esse número é utilizado para inferir o número efetivo de sequências (M_eff) (equação [15]). O intervalo se seleção do θ vai de 0,1 à 1,0 em incrementos de 0,1, com valor padrão igual a 0,8. Utilizamos a caixa de seleção seguinte **(Figura 25, marcação em vermelho 7)** para selecionar o valor da distância de

contato (r_C) (Seção III.1). Seu padrão é igual a 8,0. Por último, podemos selecionar nas caixas de marcação **(Figura 25, marcação em vermelho 8)** quais funções serão computadas pelo programa.

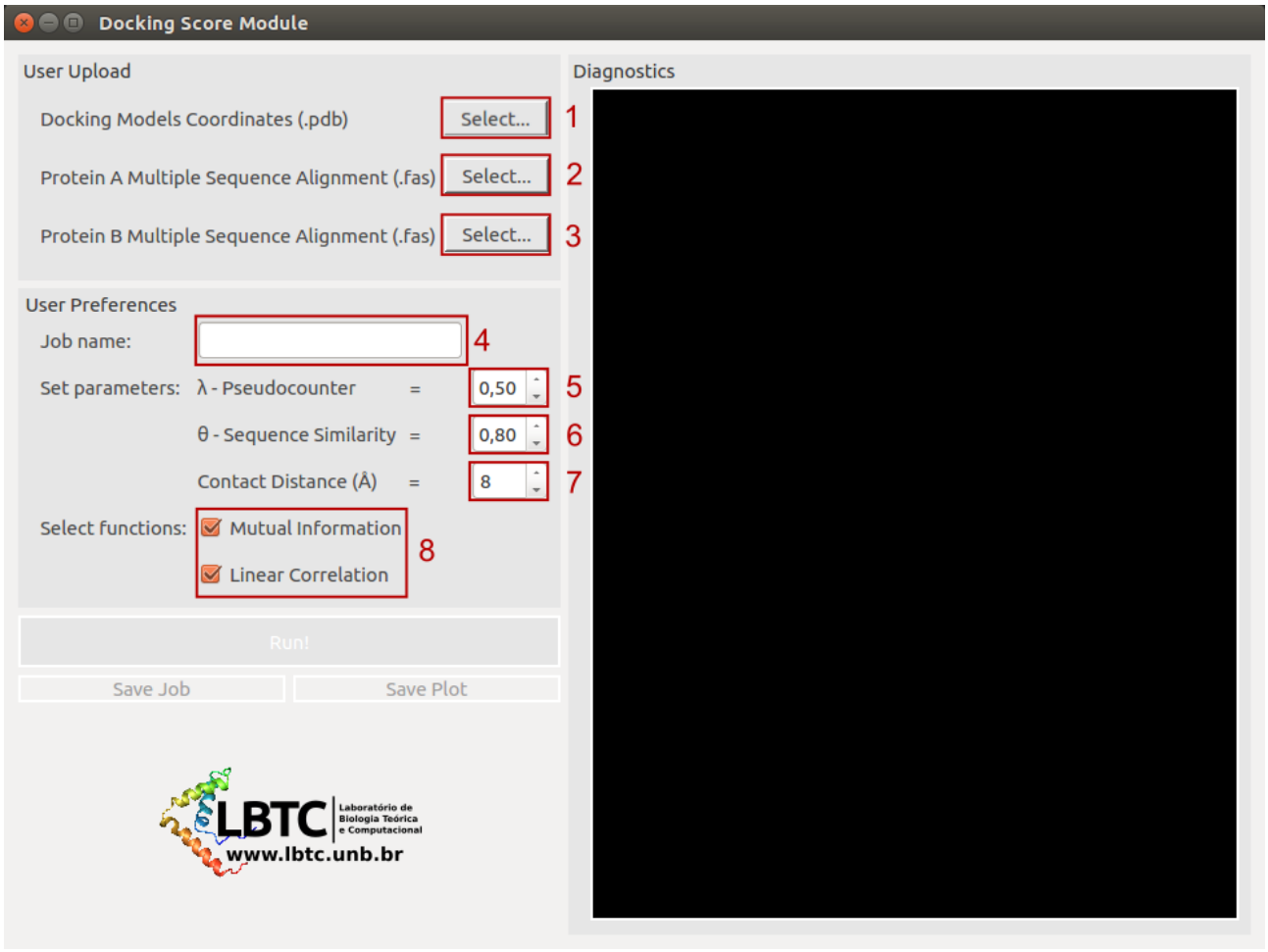

Figura 25: Janela de abertura do DSM.

Após a seleção dos arquivos, o campo de diagnóstico **(Figura 26, marcação em vermelho 1)** exibe o caminho de todos os arquivos selecionados, caso estes sejam válidos, ou uma mensagem de erro, caso contrário. A validação ocorre mediante a verificação da distância de Hamming entre as sequências primárias das proteínas presentes nos modelos e as sequências de referência de seus respectivos alinhamentos múltiplos. Esta distância não deve ser superior a 1%, de forma que o algoritmo pode trabalhar com uma margem de imprecisão nas sequências. Caso os arquivos sejam válidos, o botão "Run" **(Figura 26, marcação em vermelho 2)** é ativado e o usuário pode iniciar o trabalho.

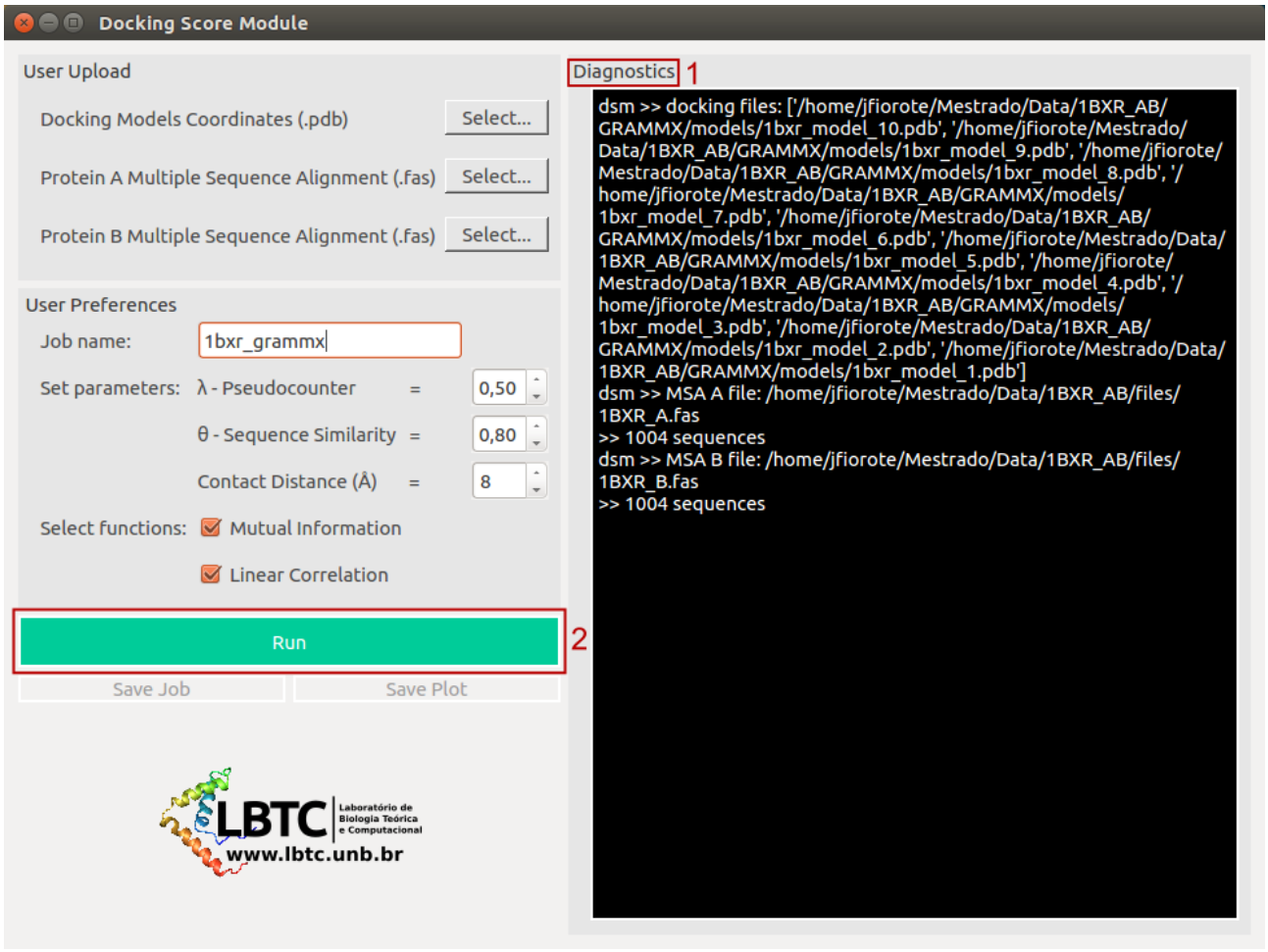

Figura 26: Janela exibida pelo DSM quando os arquivos fornecidos pelo usuário são válidos.

O trabalho é iniciado através do clique no botão "Run". Durante os cálculos, o mesmo *widget* é convertido em um botão de cancelamento **(Figura 27, marcação em vermelho 1)** enquanto os outros *widgets* são inativados, de maneira que não é mais possível alterar qualquer informação. São exibidas no campo de diagnóstico as informações pertinentes ao trabalho, como o nome selecionado para o trabalho na janela de abertura, o valor de similaridade de sequências (θ) escolhido e o número efetivo de sequências (M_{eff}). Caso o usuário deseje o cancelar o trabalho, ele deve clicar no botão "Cancel" e confirmar sua escolha na caixa de diálogo subsequente.

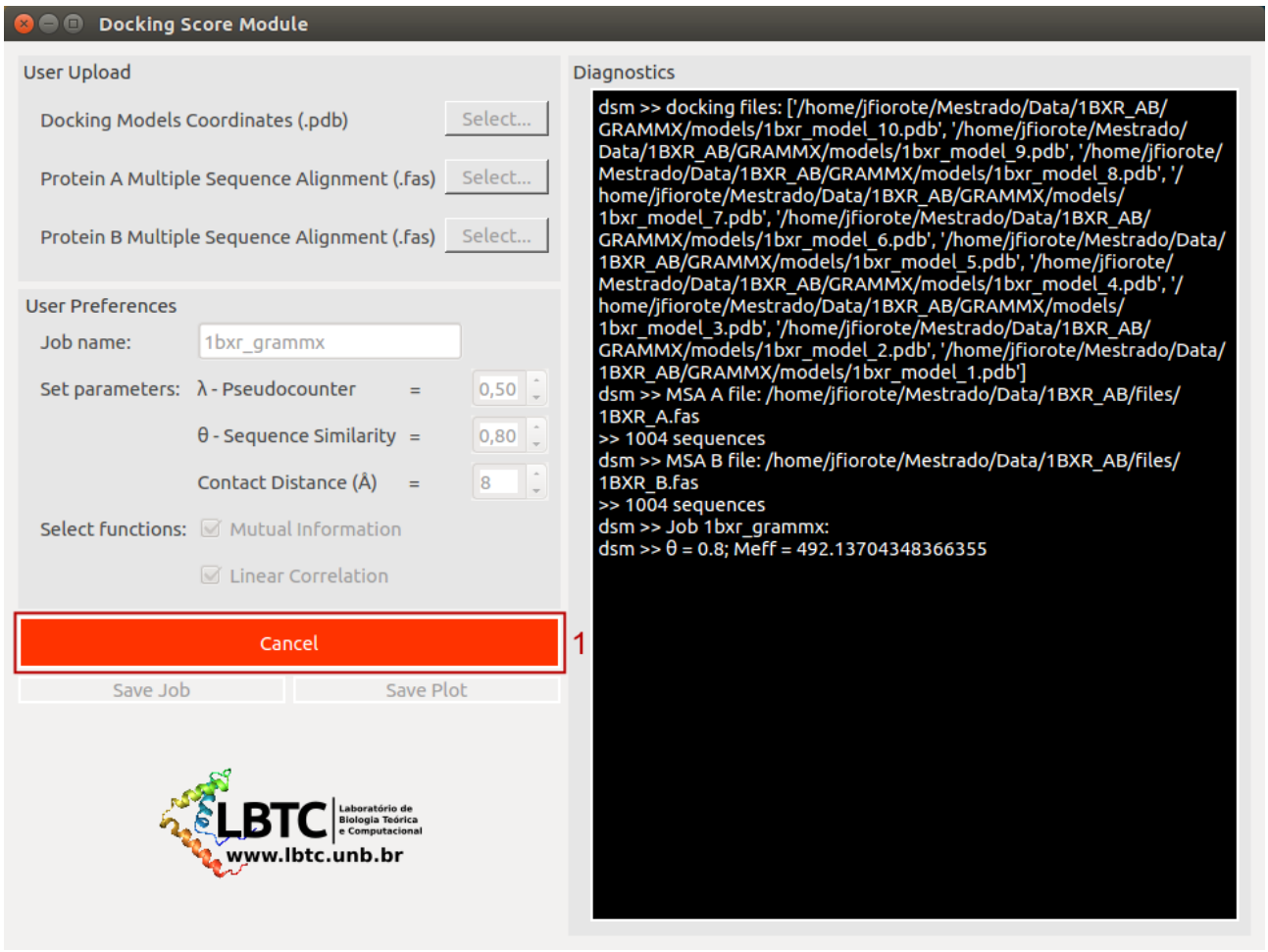

Figura 27: Janela exibida pelo DSM durante os cálculos de coevolução.

Decorrido o trabalho, o botão "Cancel" é alterado para o botão "Plot" **(Figura 28, marcação em vermelho 1)**, que tem a função de exibir um gráfico com os valores de MI/H e r. A **Figura 29** mostra o gráfico gerado na ocasião deste trabalho. Os dados dos resultados são exibidos na janela de diagnóstico e também estão disponíveis em um relatório gerado. Este relatório pode ser salvo em formato de texto ".txt" através do clique no botão "Save Job" **(Figura 28, marcação em vermelho 2)**, agora ativo. Este botão abre a navegação entre diretórios usando como nome padrão para o arquivo o identificador inserido no campo "Job name". Como dito anteriormente, caso ele não tenha sido preenchido, o nome padrão é composto pela palavra "*job*" juntamente com a data e o horário em que o trabalho foi iniciado. A **Figura 30** mostra a visualização do arquivo texto gerado neste exemplo.

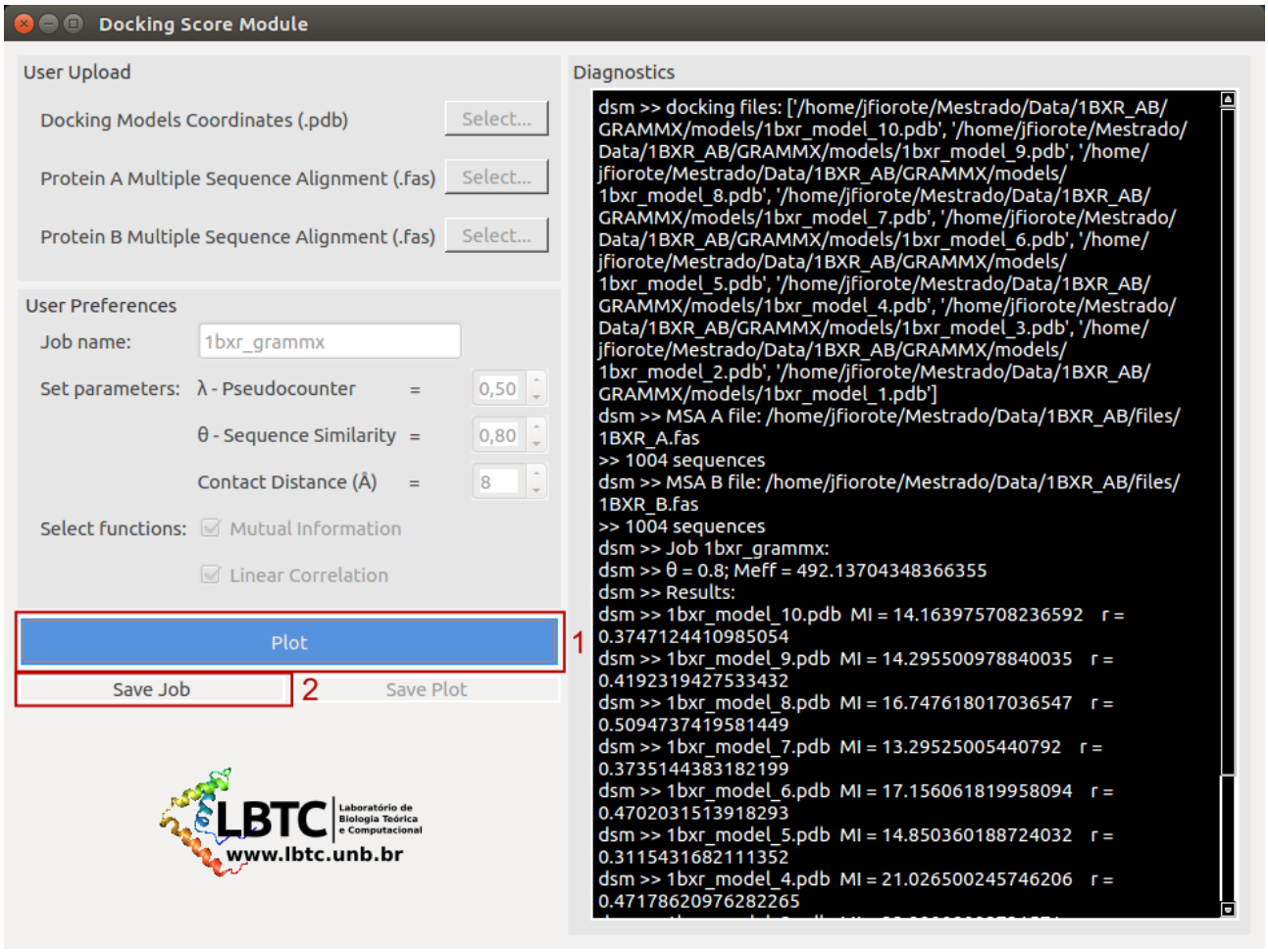

Figura 28: Janela exibida pelo DSM após o término dos cálculos de coevolução.

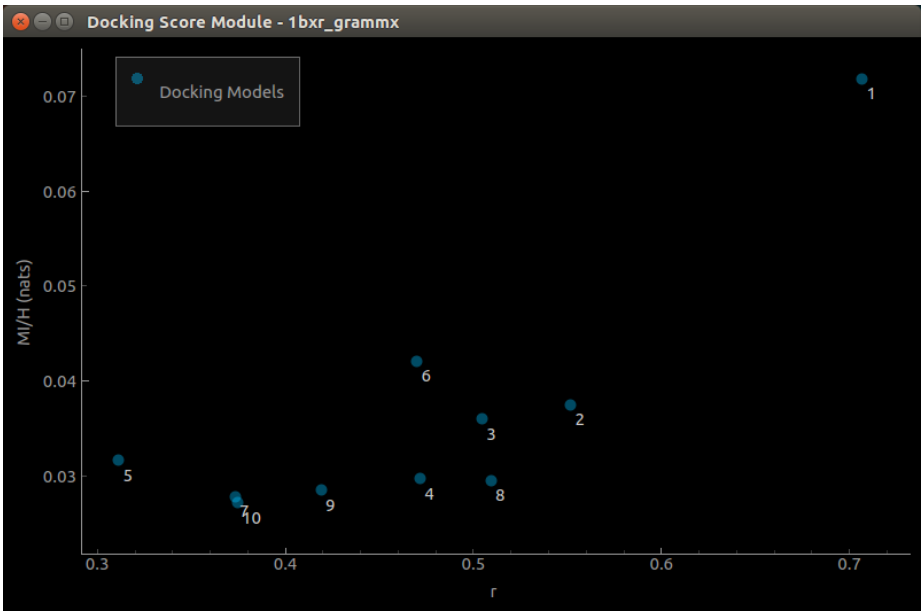

Figura 29: Exemplo do gráfico gerado pelo DSM para o cálculo dos modelos do complexo 1BXR A-B gerados pelo servidor GRAMMX.

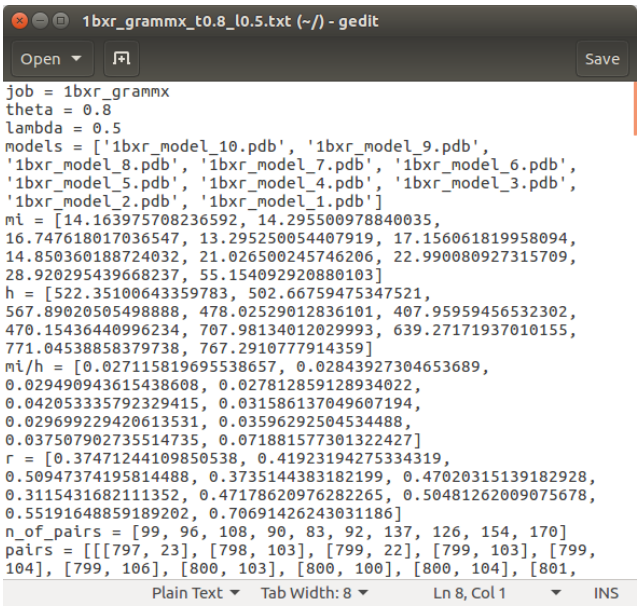

Figura 30: Visualização do arquivo ".txt" gerado pelo DSM através do programa Gedit.

Após a geração do gráfico, o DSM exibe a sua janela final e o botão "Plot" e convertido em "Restart" (**Figura 31, marcação em vermelho 1**). Assim, o usuário pode iniciar outro trabalho, caso queira. O gráfico gerado também pode ser salvo em um arquivo do tipo *Scalable Vector Graphics* ".svg", navegando nos diretórios por meio do botão "Save Plot"(**Figura 31, marcação em vermelho 2**).

Para o caso mostrado nesse exemplo, onde utilizou-se os modelos do complexo 1BXR A-B gerados pelo servidor GRAMMX, o gráfico representado na **Figura 29** indica ao usuário que a solução número 1 possui o maior valor de r e também de MI/H. Desta forma, o modelo número 1 é indicado como o modelo mais próximo ao complexo alvo. De fato, podemos verificar que este modelo tem o menor desvio estrutural dentre os 10 modelos propostos **(Tabela 4)**.

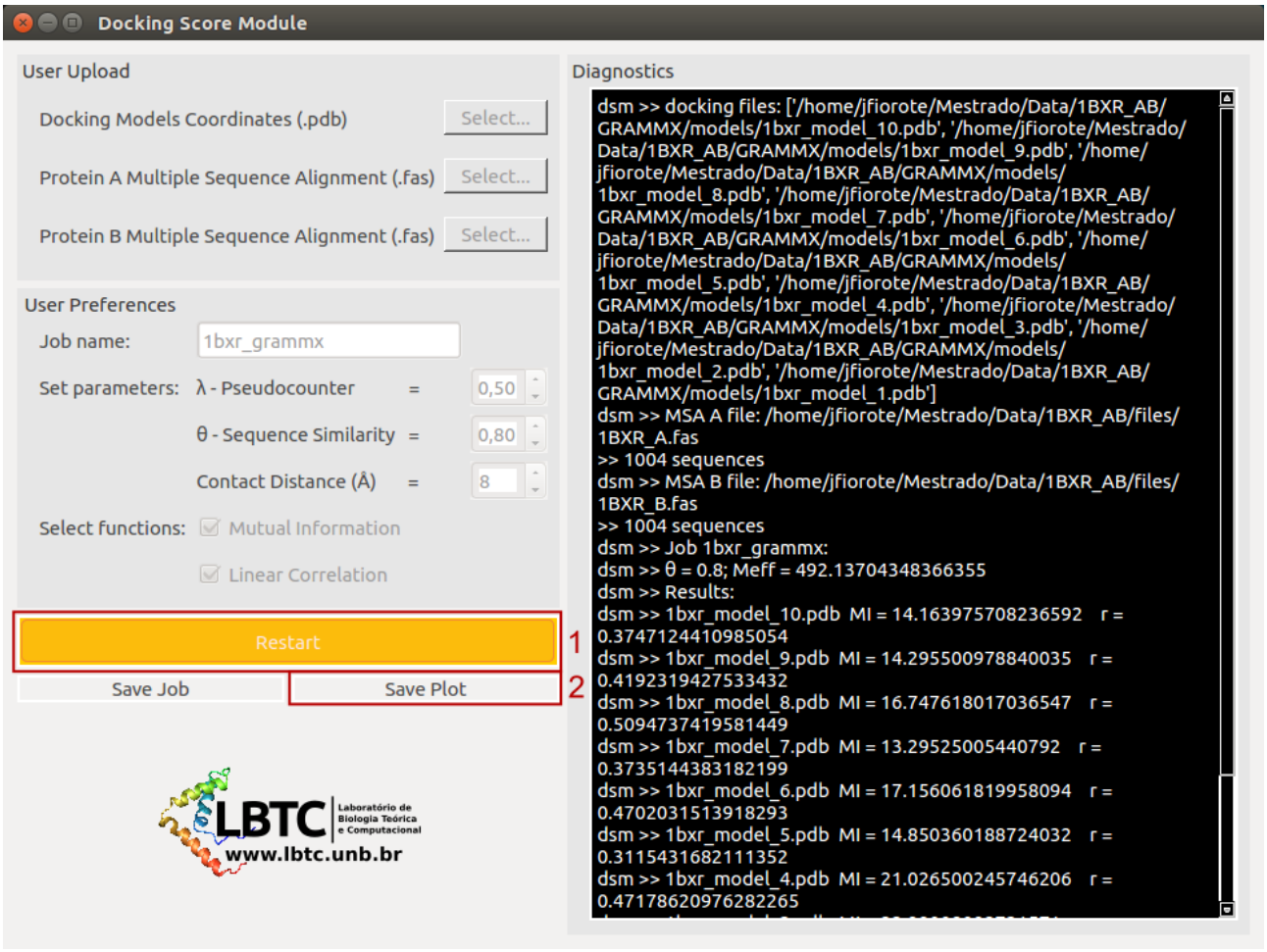

Figura 31: Janela Final do DSM.

V – Conclusão e Perspectivas

Podemos ver nos resultados que o algoritmo, ao tratar de interações entre proteínas globulares, é capaz de atribuir uma alta pontuação para modelos de *docking* mais próximos à estrutura alvo. Precisamos, no entanto, minimizar os efeitos indiretos no cálculo da informação mútua. Um ponto a considerar é a decomposição da covariância entre os aminoácidos. A informação mútua mede a covariância geral entre posições de um MSA. Fares e colaboradores e seu trabalho de 2008[71] sugerem que a covariância entre duas posições de um alinhamento pode ser dividida em cinco diferentes frações:

$$C_{ij} = C_{philogeny} + C_{structure} + C_{function} + C_{interactions} + C_{stochastic}$$

A fração $C_{philogeny}$ se refere ao conceito de dependência histórica entre duas espécies na natureza aplicado às posições dos aminoácidos. As frações $C_{structure}$ e $C_{function}$ agem na manutenção dos sítios funcionais e são difíceis de distinguir, pois não são mutualmente excludentes. A parte $C_{interactions}$ confunde-se com as outras, pois geralmente reflete algum componente funcional ou estrutural. Esta parte leva em conta que a variação no nível da sequência em sítios que estão envolvidos na interação entre proteínas. Por último, a fração $C_{stochastic}$ pode ser originada a partir da convergência da dinâmica mutacional das posições de aminoácidos. Esse efeito pode ser muito significativo quando a inferência da covariância é feita a partir de um MSA de baixa qualidade ou com poucas sequências[72]. Apesar da composição da covariância entre aminoácidos não ter sido desmembrada em nossos estudos, podemos supor que o efeito de ruído pode ser grande parte do sinal detectado no caso dos modelos que possuem alto $rmsd$ em relação à estrutura alvo e altos valores de $MI/H \times r$.

A teoria baseada no trabalho de Fares e colaboradores[68] já vem sendo desenvolvida por Werner e colaboradores, onde a informação mútua pode ser dada por:

$$MI_{r_C \leq 8}/N_{r_C \leq 8} = MI_{coev} + MI_{evol} + MI_{rand}$$

onde $N_{r_C \leq 8}$ é o número de contatos do conjunto, selecionados de acordo com uma distância r_C, MI_{coev} é a parcela estimada de coevolução e abrange as frações de estrutura, função e interação. A MI_{evol} é a parcela derivada da evolução e abrange a fração que decorre da filogenia, e MI_{rand} é a parcela derivada de fatores estocásticos.

A parte da informação mútua estocástica pode ser calculada pela seguinte relação:

$$MI_{rand} = \frac{\langle MI_{r_c} > 8(X;Y \mid z_{rand}) \rangle}{N_{rC>8}}$$

onde z_{rand} é a média de informação mútua para diferentes modelos de coevolução estocásticos. Estes cálculos tem um alto custo computacional, de forma que ainda é preciso estender um pouco mais seu período de implementação.

Outro ponto a ser considerado é a métrica da covariância. A informação mútua baseia-se em um modelo estatístico local, onde todas as interações são consideradas independentes. Como discutido anteriormente, há diversos tipos de restrições que limitam o padrão de substituição dos aminoácidos em uma proteína. Essas amarras não são levadas em conta na MI, de forma que algumas das correlações detectadas por este método são correlações indiretas. Suponha que alguma posição i esteja diretamente acoplada com uma posição j que, por sua vez, esteja acoplada com uma posição k. Mesmo sem um acoplamento direto, as posições i e k irão mostrar covariância. Esse efeito pode tornar-se pronunciado se houver uma grande quantidade de acoplamentos indiretos em um MSA[52,73].

Para solucionar esse problema, outra métrica a ser implementada no DSM é o análise de acoplamento direto (do inglês "*direct coupling analysis"* (DCA)). Este método estatístico global[52] pode distinguir quais interações decorrem de um acoplamento direto entre os resíduos. Porém, ainda é preciso ainda adaptar a DCA ao nosso problema. Ela foi desenvolvida dentro de uma dimensão intraproteica, de modo que, antes da sua implementação propriamente dita, é preciso entender como este método funciona quando se trata de uma interação entre duas proteínas distintas.

Mesmo com as melhorias a serem implementadas, o DSM já é uma ferramenta funcional. A expectativa para a sua disponibilização, juntamente com a redação do artigo sobre este trabalho, está dentro de um médio prazo, pois outros testes serão feitos para ratificar a eficácia do algoritmo. Também é preciso de tempo para a construção dos alinhamentos para os novos estudos de caso, visto que estes testes serão conduzidos utilizando um número bem maior de complexos.

Referências Bibliográficas

1. Bustin, S. Molecular Biology of the Cell, Sixth Edition; ISBN: 9780815344643; and Molecular Biology of the Cell, Sixth Edition, The Problems Book; ISBN 9780815344537. *Int. J. Mol. Sci.* **16**, 28123–28125 (2015).
2. 125th Anniversary Issue: Science Online Special Feature. Available at: http://www.sciencemag.org/site/feature/misc/webfeat/125th/. (Accessed: 10th September 2018)
3. Bienstock, R. J. Computational Drug Design Targeting Protein-Protein Interactions. *Current Pharmaceutical Design* (2012). Available at: http://www.eurekaselect.com/76583/article. (Accessed: 21st October 2017)
4. Coelho, E. D. & Oliveira, J. P. A. and J. L. From Protein-Protein Interactions to Rational Drug Design: Are Computational Methods Up to the Challenge? *Current Topics in Medicinal Chemistry* (2013). Available at: http://www.eurekaselect.com/109234/article. (Accessed: 27th October 2018)
5. Fry, D. C. Targeting Protein-Protein Interactions for Drug Discovery. in *Protein-Protein Interactions: Methods and Applications* (eds. Meyerkord, C. L. & Fu, H.) 93–106 (Springer New York, 2015). doi:10.1007/978-1-4939-2425-7_6
6. Small molecules, big targets: drug discovery faces the protein–protein interaction challenge | Nature Reviews Drug Discovery. Available at: https://www.nature.com/articles/nrd.2016.29. (Accessed: 25th December 2018)
7. Elucidating the druggable interface of protein−protein interactions using fragment docking and coevolutionary analysis | PNAS. Available at: https://www.pnas.org/content/113/50/E8051. (Accessed: 25th December 2018)
8. Reaching for high-hanging fruit in drug discovery at protein–protein interfaces | Nature. Available at: https://www.nature.com/articles/nature06526. (Accessed: 25th December 2018)
9. Foo, J. L., Ching, C. B., Chang, M. W. & Leong, S. S. J. The imminent role of protein engineering in synthetic biology. *Biotechnol. Adv.* **30**, 541–549 (2012).
10. Glasscock, C. J., Lucks, J. B. & DeLisa, M. P. Engineered Protein Machines: Emergent Tools for Synthetic Biology. *Cell Chem. Biol.* **23**, 45–56 (2016).

11. Jones, S. & Thornton, J. M. Principles of protein-protein interactions. *Proc. Natl. Acad. Sci. U. S. A.* **93**, 13–20 (1996).

12. Nooren, I. M. A. & Thornton, J. M. Diversity of protein–protein interactions. *EMBO J.* **22**, 3486–3492 (2003).

13. La, D., Kong, M., Hoffman, W., Choi, Y. I. & Kihara, D. Predicting permanent and transient protein-protein interfaces. *Proteins* **81**, 805–818 (2013).

14. Jones, S. & Thornton, J. M. Protein-protein interactions: A review of protein dimer structures. *Prog. Biophys. Mol. Biol.* **63**, 31–65 (1995).

15. Predicting Protein–Protein Interactions from the Molecular to the Proteome Level - Chemical Reviews (ACS Publications). Available at: https://pubs.acs.org/doi/abs/10.1021/acs.chemrev.5b00683. (Accessed: 18th December 2018)

16. Structural Characterisation and Functional Significance of Transient Protein–Protein Interactions - ScienceDirect. Available at: https://www.sciencedirect.com/science/article/pii/S0022283602012810?via%3Dihub. (Accessed: 19th December 2018)

17. Ehrlich, L. P. & Wade, R. C. Protein-Protein Docking. in *Reviews in Computational Chemistry* 61–97 (Wiley-Blackwell, 2001). doi:10.1002/0471224413.ch2

18. Méndez, R., Leplae, R., Maria, L. D. & Wodak, S. J. Assessment of blind predictions of protein–protein interactions: Current status of docking methods. *Proteins Struct. Funct. Bioinforma.* **52**, 51–67 (2003).

19. Smith, G. R. & Sternberg, M. J. E. Prediction of protein-protein interactions by docking methods. *Curr. Opin. Struct. Biol.* **12**, 28–35 (2002).

20. Halperin, I., Ma, B., Wolfson, H. & Nussinov, R. Principles of docking: An overview of search algorithms and a guide to scoring functions. *Proteins Struct. Funct. Bioinforma.* **47**, 409–443 (2002).

21. Harrison, R. W., Kourinov, I. V. & Andrews, L. C. The Fourier-Green's function and the rapid evaluation of molecular potentials. *Protein Eng. Des. Sel.* **7**, 359–369 (1994).

22. Palma, P. N., Krippahl, L., Wampler, J. E. & Moura, J. J. BiGGER: a new (soft) docking algorithm for predicting protein interactions. *Proteins* **39**, 372–384 (2000).

23. Fischer, D., Lin, S. L., Wolfson, H. L. & Nussinov, R. A geometry-based suite of moleculardocking processes. *J. Mol. Biol.* **248**, 459–477 (1995).

24. Vakser, I. A. Evaluation of GRAMM low-resolution docking methodology on the hemagglutinin-antibody complex. *Proteins Struct. Funct. Bioinforma.* **29**, 226–230 (1997).

25. Yan, Y., Zhang, D., Zhou, P., Li, B. & Huang, S.-Y. HDOCK: a web server for protein–protein and protein–DNA/RNA docking based on a hybrid strategy. *Nucleic Acids Res.* **45**, W365–W373 (2017).

26. Pierce, B. G., Hourai, Y. & Weng, Z. Accelerating protein docking in ZDOCK using an advanced 3D convolution library. *PloS One* **6**, e24657 (2011).

27. Katchalski-Katzir, E. *et al.* Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques. *Proc. Natl. Acad. Sci. U. S. A.* **89**, 2195–2199 (1992).

28. Chen, R. & Weng, Z. A novel shape complementarity scoring function for protein-protein docking. *Proteins Struct. Funct. Bioinforma.* **51**, 397–408 (2003).

29. Protein docking along smooth association pathways. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC58518/. (Accessed: 5th November 2018)

30. Shoichet, B. K. & Kuntz, I. D. Protein docking and complementarity. *J. Mol. Biol.* **221**, 327–346 (1991).

31. Jackson, R. M. & Sternberg, M. J. E. A Continuum Model for Protein–Protein Interactions: Application to the Docking Problem. *J. Mol. Biol.* **250**, 258–275 (1995).

32. Jackson, R. M., Gabb, H. A. & Sternberg, M. J. Rapid refinement of protein interfaces incorporating solvation: application to the docking problem. *J. Mol. Biol.* **276**, 265–285 (1998).

33. Camacho, C. J., Gatchell, D. W., Kimura, S. R. & Vajda, S. Scoring docked conformations generated by rigid-body protein-protein docking. *Proteins Struct. Funct. Bioinforma.* **40**, 525–537 (2000).

34. Pons, C., Grosdidier, S., Solernou, A., Pérez-Cano, L. & Fernández-Recio, J. Present and future challenges and limitations in protein–protein docking. *Proteins Struct. Funct. Bioinforma.* **78**, 95–108 (2010).

35. Andrusier, N., Mashiach, E., Nussinov, R. & Wolfson, H. J. Principles of flexible protein–protein docking. *Proteins Struct. Funct. Bioinforma.* **73**, 271–289 (2008).

36. Bonvin, A. M. Flexible protein–protein docking. *Curr. Opin. Struct. Biol.* **16**, 194–200 (2006).

37. Li, L., Chen, R. & Weng, Z. RDOCK: Refinement of rigid-body protein docking predictions. *Proteins Struct. Funct. Bioinforma.* **53**, 693–707 (2003).

38. Vajda, S. & Kozakov, D. Convergence and combination of methods in protein-protein docking. *Curr. Opin. Struct. Biol.* **19**, 164–170 (2009).

39. Use of pair potentials across protein interfaces in screening predicted docked complexes - Moont - 1999 - Proteins: Structure, Function, and Bioinformatics - Wiley Online Library. Available at: https://onlinelibrary.wiley.com/doi/full/10.1002/%28SICI%291097-0134%2819990515%2935%3A3%3C364%3A%3AAID-PROT11%3E3.0.CO%3B2-4. (Accessed: 5th November 2018)

40. *The Selfish Gene: 40th Anniversary Edition*. (Oxford University Press, 2016).

41. Van Valen, L. Molecular evolution as predicted by natural selection. *J. Mol. Evol.* **3**, 89–101 (1974).

42. Dieckmann, U. & Law, R. The dynamical theory of coevolution: a derivation from stochastic ecological processes. *J. Math. Biol.* **34**, 579–612 (1996).

43. Ehrlich, P. R. & Raven, P. H. Butterflies and Plants: A Study in Coevolution. *Evolution* **18**, 586–608 (1964).

44. Lovell, S. C. & Robertson, D. L. An Integrated View of Molecular Coevolution in Protein–Protein Interactions. *Mol. Biol. Evol.* **27**, 2567–2575 (2010).

45. Capra, J. A. & Singh, M. Predicting functionally important residues from sequence conservation. *Bioinformatics* **23**, 1875–1882 (2007).

46. Lewis, A. C. F., Saeed, R. & Deane, C. M. Predicting protein–protein interactions in the context of protein evolution. *Mol. Biosyst.* **6**, 55–64 (2009).

47. Martin, L. C., Gloor, G. B., Dunn, S. D. & Wahl, L. M. Using information theory to search for co-evolving residues in proteins. *Bioinformatics* **21**, 4116–4124 (2005).

48. Ovchinnikov, S., Kamisetty, H. & Baker, D. Robust and accurate prediction of residue–residue interactions across protein interfaces using evolutionary information. *eLife* **3**, e02030 (2014).

49. Echave, J., Spielman, S. J. & Wilke, C. O. Causes of evolutionary rate variation among protein sites. *Nat. Rev. Genet.* **17**, 109–121 (2016).

50. de Juan, D., Pazos, F. & Valencia, A. Emerging methods in protein co-evolution. *Nat. Rev. Genet.* **14**, 249–261 (2013).

51. Pazos, F. & Valencia, A. Similarity of phylogenetic trees as indicator of protein–protein interaction. *Protein Eng. Des. Sel.* **14**, 609–614 (2001).

52. Morcos, F. *et al.* Direct-coupling analysis of residue coevolution captures native contacts across many protein families. *Proc. Natl. Acad. Sci.* **108**, E1293–E1301 (2011).

53. GRAMM-X public web server for protein–protein docking | Nucleic Acids Research | Oxford Academic. Available at: https://academic.oup.com/nar/article/34/suppl_2/W310/2505594. (Accessed: 4th February 2019)

54. Lyskov, S. & Gray, J. J. The RosettaDock server for local protein–protein docking. *Nucleic Acids Res.* **36**, W233–W238 (2008).

55. Pierce, B. G. *et al.* ZDOCK server: interactive docking prediction of protein-protein complexes and symmetric multimers. *Bioinforma. Oxf. Engl.* **30**, 1771–1773 (2014).

56. A Mathematical Theory of Communication - Shannon - 1948 - Bell System Technical Journal - Wiley Online Library. Available at: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1948.tb01338.x. (Accessed: 27th October 2018)

57. Elements of Information Theory, 2nd Edition. *Wiley.com* Available at: https://www.wiley.com/en-us/Elements+of+Information+Theory%2C+2nd+Edition-p-9780471241959. (Accessed: 16th January 2019)

58. Biological Sequence Analysis by Richard Durbin. Available at: https://www.cambridge.org/core/books/biological-sequence-analysis/921BB7B78B745198829EF96BC7E0F29D. (Accessed: 3rd February 2019)

59. Eddy, S. R. Where did the BLOSUM62 alignment score matrix come from? *Nat. Biotechnol.* **22**, 1035–1036 (2004).

60. Cock, P. J. A. *et al.* Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* **25**, 1422–1423 (2009).

61. Hamelryck, T. & Manderick, B. PDB file parser and structure class implemented in Python. *Bioinformatics* **19**, 2308–2310 (2003).

62. Oliphant, T. E. Python for Scientific Computing. *Comput. Sci. Eng.* **9**, 10–20 (2007).

63. Bloom, J. D., Labthavikul, S. T., Otey, C. R. & Arnold, F. H. Protein stability promotes evolvability. *Proc. Natl. Acad. Sci.* **103**, 5869–5874 (2006).

64. Structure, function, and evolution of transient and obligate protein–protein interactions. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1182425/. (Accessed: 3rd February 2019)

65. Carbamoyl Phosphate Synthetase: Closure of the B-Domain as a Result of Nucleotide Binding, - Biochemistry (ACS Publications). Available at: https://pubs.acs.org/doi/pdf/10.1021/bi982517h. (Accessed: 3rd February 2019)

66. Rowland, P., Nørager, S., Jensen, K. F. & Larsen, S. Structure of Dihydroorotate Dehydrogenase B: Electron Transfer between Two Flavin Groups Bridged by an Iron-Sulphur Cluster. *Structure* **8**, 1227–1238 (2000).

67. Settembre, E. C. *et al.* Thiamin Biosynthesis in Bacillus subtilis: Structure of the Thiazole Synthase/Sulfur Carrier Protein Complex,. *Biochemistry* **43**, 11647–11657 (2004).

68. Kurzbauer, R. *et al.* Crystal structure of the p14/MP1 scaffolding complex: How a twin couple attaches mitogen-activated protein kinase signaling to late endosomes. *Proc. Natl. Acad. Sci.* **101**, 10984–10989 (2004).

69. Kaila, V. R. I. *et al.* A combined quantum chemical and crystallographic study on the oxidized binuclear center of cytochrome c oxidase. *Biochim. Biophys. Acta BBA - Bioenerg.* **1807**, 769–778 (2011).

70. Assessing the utility of coevolution-based residue–residue contact predictions in a sequence- and structure-rich era. Available at: http://www.pnas.org.ez54.periodicos.capes.gov.br/content/110/39/15674.full. (Accessed: 7th December 2017)

71. Codoñer, F. M. & Fares, M. A. Why Should We Care About Molecular Coevolution? *Evol. Bioinforma. Online* **4**, 29–38 (2008).

72. A Novel Method for Detecting Intramolecular Coevolution: Adding a Further Dimension to Selective Constraints Analyses | Genetics. Available at: http://www.genetics.org/content/173/1/9.long. (Accessed: 5th February 2019)

73. Weigt, M., White, R. A., Szurmant, H., Hoch, J. A. & Hwa, T. Identification of direct residue contacts in protein–protein interaction by message passing. *Proc. Natl. Acad. Sci.* **106**, 67–72 (2009).

Anexo I – Resultados dos Estudos de Caso

Tabela 4: Valores de $rmsd$, MI, H, MI/H e r para o complexo 1BXR A-B e seus modelos putativos gerados por diferentes servidores de *docking*.

Servidor	Modelo	rmsd	MI Total	H Total	MI/H	r
X	Nativo	0	50,11983	693,80222	0,07224	0,68621
GRAMMX	1	2,18822	55,15409	767,29108	0,07188	0,70691
	2	81,46600	28,92030	771,04539	0,03751	0,55192
	3	52,77658	22,99008	639,27172	0,03596	0,50481
	4	82,62280	21,02650	707,98134	0,02970	0,47179
	5	79,81957	14,85036	470,15436	0,03159	0,31154
	6	59,11638	17,15606	407,95959	0,04205	0,47020
	7	106,76657	13,29525	478,02529	0,02781	0,37351
	8	76,58395	16,74762	567,89021	0,02949	0,50947
	9	100,35320	14,29550	502,66759	0,02844	0,41923
	10	100,74359	14,16398	522,35101	0,02712	0,37471
HDOCK	1	0,54300	54,16112	754,08097	0,07182	0,71511
	2	60,76893	25,02138	526,05780	0,04756	0,46358
	3	83,13728	29,85664	595,89526	0,05010	0,58602
	4	76,94186	14,45101	337,66241	0,04280	0,35641
	5	68,55561	16,61189	326,46226	0,05088	0,37680
	6	61,63171	13,48409	323,08397	0,04174	0,29274
	7	14,92326	23,47843	316,30457	0,07423	0,62629
	8	98,75919	10,68456	289,11396	0,03696	0,44187
	9	103,08092	15,61314	474,94313	0,03287	0,41415
	10	101,81568	12,36222	296,98977	0,04163	0,33364
ROSIE	1	2,77530	38,24088	527,72846	0,07246	0,66337
	2	5,00445	26,50625	379,35995	0,06987	0,55240
	3	4,94064	25,99888	369,21907	0,07042	0,55594
	4	4,86355	23,46014	326,32603	0,07189	0,51549
	5	5,05647	20,66097	297,33986	0,06949	0,55499
	6	4,16746	21,74143	310,17467	0,07009	0,53213
	7	36,87152	6,07177	131,59056	0,04614	0,28207
	8	5,55870	24,52939	350,74745	0,06993	0,54333
	9	5,20526	15,95095	228,19898	0,06990	0,53479
	10	5,02390	17,33038	252,01184	0,06877	0,60088
ZDOCK	1	2,55619	50,27297	685,35993	0,07335	0,63917
	2	4,73504	35,37194	535,62385	0,06604	0,54143
	3	4,98357	50,89336	768,62620	0,06621	0,64812
	4	59,28601	26,00552	498,38117	0,05218	0,40742
	5	90,23189	24,53072	514,95550	0,04764	0,50790
	6	13,07288	17,33632	302,65457	0,05728	0,42681
	7	6,21535	42,06120	585,18832	0,07188	0,58939
	8	86,41090	15,77823	402,61361	0,03919	0,44120
	9	108,50024	19,02321	457,65562	0,04157	0,41247
	10	59,92193	22,88292	542,02280	0,04222	0,34792

Tabela 5: Valores de $rmsd$, MI, H, MI/H e r para o complexo 1EP3 A-B e seus modelos putativos gerados por diferentes servidores de *docking*.

Servidor	Modelo	rmsd	MI Total	H Total	MI/H	r
X	Nativo	0	24,67180	439,73311	0,05611	0,72694
GRAMMX	1	2,25191	26,82051	483,43137	0,05548	0,73503
	2	8,34263	25,82626	481,20942	0,05367	0,73546
	3	44,99174	28,34506	532,67182	0,05321	0,56119
	4	58,99828	19,50126	418,43972	0,04660	0,50902
	5	61,93951	24,11257	447,48263	0,05388	0,63826
	6	50,04640	20,25319	424,39048	0,04772	0,56264
	7	48,27326	12,25802	405,34139	0,03024	0,24151
	8	58,65607	22,45849	464,83003	0,04832	0,45755
	9	53,20958	20,62709	407,69398	0,05059	0,61200
	10	54,96478	17,25004	481,90551	0,03580	0,34836
HDOCK	1	0,52971	25,53440	447,24355	0,05709	0,71139
	2	0,97396	27,63873	489,93784	0,05641	0,72007
	3	6,76421	22,73319	417,90299	0,05440	0,73330
	4	8,24303	18,86660	343,35503	0,05495	0,69808
	5	56,30980	14,14661	413,22887	0,03423	0,36055
	6	58,18789	18,65481	403,27903	0,04626	0,54785
	7	28,90113	15,35888	290,70374	0,05283	0,59817
	8	59,18753	13,45352	341,15215	0,03944	0,43252
	9	5,17960	19,46334	349,09360	0,05575	0,63483
	10	58,44264	15,58007	348,09495	0,04476	0,49382
ROSIE	1	1,27083	22,84456	395,66101	0,05774	0,71005
	2	1,31802	22,26906	385,89051	0,05771	0,73083
	3	1,56694	22,08768	380,49171	0,05805	0,72336
	4	1,59216	20,46920	351,88270	0,05817	0,72026
	5	1,25549	19,93958	347,92083	0,05731	0,71710
	6	1,67353	19,16063	332,85242	0,05756	0,72556
	7	3,14521	18,99254	329,44415	0,05765	0,69298
	8	1,54132	18,86725	322,77222	0,05845	0,70106
	9	1,60416	18,51232	317,69412	0,05827	0,70708
	10	2,16088	18,74225	322,07709	0,05819	0,65370
ZDOCK	1	2,91697	27,99784	514,66265	0,05440	0,72841
	2	3,74080	32,16960	586,83060	0,05482	0,75082
	3	5,16557	25,41595	503,04274	0,05052	0,71702
	4	6,07119	21,29012	400,85523	0,05311	0,72272
	5	5,56764	26,67406	496,96241	0,05367	0,72060
	6	7,78344	22,50676	396,14678	0,05681	0,73503
	7	44,72924	21,69783	452,99097	0,04790	0,47952
	8	28,71300	20,39503	400,76583	0,05089	0,67949
	9	6,40477	29,23793	548,24895	0,05333	0,72691
	10	29,64298	11,25790	279,67808	0,04025	0,44964

Tabela 6: Valores de $rmsd$, MI, H, MI/H e r para o complexo 1TYG A-B e seus modelos putativos gerados por diferentes servidores de *docking*.

Servidor	Modelo	rmsd	MI Total	H Total	MI/H	r
X	Nativo	0	23,36036	383,40769	0,06093	0,58159
GRAMMX	1	2,18130	23,85867	387,49000	0,06157	0,58159
	2	32,00004	18,52290	409,44694	0,04524	0,52893
	3	36,82898	9,80558	327,24741	0,02996	0,24702
	4	20,94498	11,89946	319,04582	0,03730	0,47089
	5	33,25182	17,17585	326,45611	0,05261	0,47070
	6	63,49081	9,15946	283,73019	0,03228	0,36973
	7	18,04462	14,25247	335,55657	0,04247	0,52784
	8	20,29048	17,24167	302,46868	0,05700	0,69607
	9	62,81004	14,65703	308,10592	0,04757	0,34092
	10	30,66498	19,16027	388,96563	0,04926	0,55525
HDOCK	1	0,71143	23,27286	372,03265	0,06256	0,54173
	2	0,66809	23,19300	372,61754	0,06224	0,54173
	3	60,83894	11,49167	303,06885	0,03792	0,24699
	4	6,15279	21,52313	356,17638	0,06043	0,64185
	5	56,82685	8,73917	206,43249	0,04233	0,33233
	6	32,15646	11,24540	301,13568	0,03734	0,41259
	7	66,18844	10,52828	272,93009	0,03858	0,32624
	8	39,07931	16,48440	325,67559	0,05062	0,43162
	9	48,31024	15,17215	284,55309	0,05332	0,35758
	10	66,53800	6,94422	233,96682	0,02968	0,14904
ROSIE	1	1,69244	23,40317	382,08765	0,06125	0,59401
	2	1,21084	23,38319	382,75162	0,06109	0,58159
	3	0,46247	23,25454	377,24780	0,06164	0,58600
	4	6,30990	15,95104	270,71212	0,05892	0,66587
	5	5,56069	12,54743	195,77820	0,06409	0,60966
	6	6,31971	12,76046	205,64546	0,06205	0,65541
	7	6,16392	14,59063	235,75094	0,06189	0,65387
	8	6,13824	14,67859	223,40548	0,06570	0,68803
	9	5,62263	13,39607	214,98972	0,06231	0,64275
	10	5,19227	13,74141	213,89742	0,06424	0,64497
ZDOCK	1	2,39178	28,68616	486,34258	0,05898	0,59651
	2	1,87165	26,72663	440,78407	0,06063	0,57293
	3	3,15334	25,96844	423,81841	0,06127	0,56934
	4	4,52259	23,80601	426,08344	0,05587	0,51152
	5	4,54428	26,70698	430,04205	0,06210	0,61001
	6	3,89106	24,33825	390,39427	0,06234	0,57140
	7	5,15653	27,15918	440,74743	0,06162	0,65551
	8	5,61287	28,85911	456,06911	0,06328	0,61167
	9	3,81670	29,56020	473,89718	0,06238	0,58944
	10	9,84953	29,25260	530,60508	0,05513	0,62749

Tabela 7: Valores de $rmsd$, MI, H, MI/H e r para o complexo 2VPZ A-B e seus modelos putativos gerados por diferentes servidores de *docking*.

Servidor	Modelo	rmsd	MI Total	H Total	MI/H	r
X	Nativo	0	9,78562	207,59685	0,04714	0,46739
GRAMMX	1	37,72402	17,87833	315,05103	0,05675	0,58516
	2	36,76073	17,34776	407,20879	0,04260	0,55130
	3	41,12182	16,19444	382,08349	0,04238	0,56528
	4	17,80064	14,43994	338,44827	0,04267	0,41308
	5	41,44783	18,35241	354,69804	0,05174	0,55292
	6	40,31641	11,76301	270,08035	0,04355	0,48644
	7	37,62245	12,76398	369,19959	0,03457	0,47283
	8	35,95856	11,55295	317,73326	0,03636	0,34196
	9	41,64521	10,63719	308,28763	0,03450	0,44414
	10	41,51316	15,87107	316,57093	0,05013	0,51643
HDOCK	0	0,33071	10,01029	212,82317	0,04704	0,49226
	1	10,78067	13,84198	245,53202	0,05638	0,62762
	2	43,34034	14,50621	301,25823	0,04815	0,53727
	3	18,88986	15,64599	300,30717	0,05210	0,50374
	4	28,72086	15,10937	261,53907	0,05777	0,59205
	5	41,36775	18,11088	384,22240	0,04714	0,58987
	6	28,17000	13,76347	243,31514	0,05657	0,57266
	7	31,19185	15,38484	264,21447	0,05823	0,59066
	8	1,42973	12,48128	263,18765	0,04742	0,52982
	9	29,46642	9,85614	268,80176	0,03667	0,38702
ROSIE	1	9,05271	8,81420	211,20373	0,04173	0,47194
	2	21,65802	7,58546	160,17077	0,04736	0,49157
	3	7,45892	6,96974	167,18663	0,04169	0,43483
	4	11,54267	8,48913	189,69656	0,04475	0,49388
	5	19,92635	7,65412	177,92268	0,04302	0,47073
	6	9,00196	7,72034	170,78177	0,04521	0,39385
	7	1,07988	8,60242	187,93631	0,04577	0,46829
	8	4,22750	10,56413	229,47016	0,04604	0,48380
	9	20,80886	6,37132	128,95588	0,04941	0,52976
	10	21,15945	8,28440	181,30098	0,04569	0,48989
ZDOCK	1	34,59880	16,07956	311,72260	0,05158	0,54166
	2	37,27650	20,70822	358,48213	0,05777	0,58834
	3	44,27268	16,31510	344,92975	0,04730	0,48914
	4	48,24964	13,72565	300,85698	0,04562	0,26193
	5	41,45028	19,87081	381,11835	0,05214	0,52814
	6	50,51942	8,94431	183,75300	0,04868	0,16950
	7	34,55696	15,31389	335,21732	0,04568	0,47883
	8	40,11567	20,31729	392,33814	0,05179	0,52217
	9	39,19139	15,33819	300,09548	0,05111	0,59426
	10	35,93018	19,74792	391,83997	0,05040	0,55335

Tabela 8: Valores de $rmsd$, MI, H, MI/H e r para o complexo 2Y69 A-B e seus modelos putativos gerados por diferentes servidores de *docking*.

Servidor	Modelo	rmsd	MI Total	H Total	MI/H	r
X	Nativo	0	64,60610	1209,35283	0,05342	0,86297
GRAMMX	1	0,44875	68,36598	1289,25686	0,05303	0,87862
	2	13,91467	37,31134	799,30709	0,04668	0,72401
	3	59,33742	30,99755	676,02816	0,04585	0,75583
	4	57,03286	25,40191	566,82675	0,04481	0,69044
	5	13,51905	29,39415	630,50109	0,04662	0,73961
	6	58,21637	24,08393	591,53632	0,04071	0,76647
	7	51,81720	28,88670	668,60491	0,04320	0,75668
	8	21,34093	25,44294	600,60108	0,04236	0,73286
	9	77,49424	17,62954	610,53275	0,02888	0,79696
	10	6,45071	39,93673	776,76626	0,05141	0,78415
HDOCK	1	0,38667	64,63411	1207,92032	0,05351	0,86679
	2	0,50964	68,73122	1281,92068	0,05362	0,86496
	3	5,58367	29,63158	626,26216	0,04731	0,74312
	4	79,33230	15,24545	523,68312	0,02911	0,78683
	5	54,29158	12,25709	456,98803	0,02682	0,65378
	6	44,12736	23,79455	509,14159	0,04673	0,72999
	7	52,64659	16,20388	416,45863	0,03891	0,55832
	8	64,44655	15,44844	492,03589	0,03140	0,65629
	9	53,06673	19,77097	521,12638	0,03794	0,52841
	10	55,85469	17,86594	477,55355	0,03741	0,69933
ROSIE	1	3,38289	36,79499	716,08722	0,05138	0,85085
	2	3,25479	37,78804	724,97653	0,05212	0,85930
	3	3,02486	33,26610	635,07694	0,05238	0,80802
	4	4,33922	29,61852	501,86191	0,05902	0,75323
	5	5,62297	28,36346	495,92065	0,05719	0,72668
	6	4,71251	26,91565	476,08878	0,05653	0,74404
	7	50,97955	1,42884	37,12582	0,03849	0,08599
	8	7,52299	21,96412	398,00613	0,05519	0,70555
	9	6,26725	25,41211	447,57903	0,05678	0,73937
	10	8,02559	20,36155	375,57005	0,05422	0,70658
ZDOCK	1	2,69774	77,70458	1362,80176	0,05702	0,89069
	2	56,61472	17,19736	541,16809	0,03178	0,67345
	3	77,28600	19,56505	656,69964	0,02979	0,80815
	4	48,05544	22,45247	658,12991	0,03412	0,73915
	5	65,64400	23,65223	734,09115	0,03222	0,66060
	6	61,55315	17,68386	638,63032	0,02769	0,64946
	7	52,27491	32,48659	854,81928	0,03800	0,74541
	8	43,74703	42,69678	899,06387	0,04749	0,78068
	9	59,08335	15,55139	507,62453	0,03064	0,58792
	10	48,87935	18,47219	518,24229	0,03564	0,58007

Tabela 9: Valores de $rmsd$, MI, H, MI/H e r para o complexo 1OYH I-L e seus modelos putativos gerados por diferentes servidores de *docking*.

Servidor	Modelo	rmsd	MI Total	H Total	MI/H	r
X	Nativo	0	12,88751	359,83892	0,03581	0,62587
GRAMMX	1	0,63396	13,01391	359,71993	0,03618	0,61358
	2	58,76893	13,05124	598,38032	0,02181	0,25001
	3	94,64998	14,02444	551,24085	0,02544	0,43395
	4	51,36621	17,23528	662,50995	0,02602	0,81168
	5	60,74833	15,45095	642,15142	0,02406	0,48998
	6	118,39006	17,36519	532,51495	0,03261	0,49279
	7	104,03479	16,63537	571,13910	0,02913	0,46143
	8	69,35757	12,17550	495,83062	0,02456	0,28530
	9	73,08819	13,13946	419,30176	0,03134	0,42603
	10	92,84541	19,07593	563,86826	0,03383	0,56345
ROSIE	1	1,43766	12,70194	354,77638	0,03580	0,62587
	2	1,94738	12,68339	354,64355	0,03576	0,63372
	3	1,17250	12,39142	344,31743	0,03599	0,62587
	4	3,52644	12,52150	356,16042	0,03516	0,63060
	5	0,68707	12,31012	338,88449	0,03633	0,62732
	6	2,23616	12,27303	345,13399	0,03556	0,61865
	7	6,94236	11,95804	336,10125	0,03558	0,64327
	8	6,04613	11,48435	319,72689	0,03592	0,61113
	9	2,54227	12,20063	338,77789	0,03601	0,59829
	10	1,44890	11,75634	329,54096	0,03567	0,59396
ZDOCK	1	6,78514	17,79658	477,38662	0,03728	0,64365
	2	116,98582	8,73411	284,73201	0,03067	0,29387
	3	112,01656	8,07768	302,40719	0,02671	0,34597
	4	45,67249	18,10849	472,62700	0,03831	0,71909
	5	114,82874	7,86646	262,85436	0,02993	0,29980
	6	109,05186	12,24868	451,28337	0,02714	0,38822
	7	7,26178	16,37435	443,30842	0,03694	0,68256
	8	12,50082	15,86344	430,15043	0,03688	0,67982
	9	50,13633	15,31975	524,04356	0,02923	0,59557
	10	110,88330	9,85661	336,78253	0,02927	0,51248

Tabela 10: Valores de $rmsd$, MI, H, MI/H e r para o complexo 1VET A-B e seus modelos putativos gerados por diferentes servidores de *docking*.

Servidor	Modelo	rmsd	MI Total	H Total	MI/H	r
X	Nativo	0	30,64013	464,94785	0,06590	0,89276
GRAMMX	1	0,47320	30,93798	469,75530	0,06586	0,89222
	2	22,62362	22,37528	349,13322	0,06409	0,88090
	3	23,14933	22,07905	344,71479	0,06405	0,84410
	4	10,18292	21,31154	332,85365	0,06403	0,86945
	5	19,81645	24,27264	388,23117	0,06252	0,85124
	6	19,33679	23,30018	352,51956	0,06610	0,74696
	7	36,71198	19,09095	321,15282	0,05945	0,87332
	8	20,79438	17,26362	267,66012	0,06450	0,82251
	9	40,45727	24,93144	421,66953	0,05913	0,82317
	10	22,54176	20,07935	304,89384	0,06586	0,76834
ROSIE	1	0,72116	27,75165	423,93956	0,06546	0,89087
	2	1,08985	27,93743	426,04019	0,06557	0,88166
	3	3,24774	24,53478	384,24536	0,06385	0,89266
	4	1,09004	25,90418	391,99497	0,06608	0,87585
	5	3,35804	25,61950	397,59341	0,06444	0,89932
	6	2,27173	23,66137	368,94904	0,06413	0,89726
	7	2,06669	25,27978	393,56547	0,06423	0,89735
	8	3,30474	20,02446	290,99189	0,06881	0,86987
	9	3,13299	19,28313	304,99145	0,06323	0,89681
	10	4,00315	20,26125	313,04041	0,06472	0,88736
ZDOCK	1	2,86182	31,85779	491,17239	0,06486	0,88480
	2	2,96058	31,21358	491,89943	0,06346	0,89248
	3	2,88400	42,94547	652,60379	0,06581	0,90054
	4	7,27093	26,88544	428,67980	0,06272	0,84977
	5	12,02390	29,00800	462,38117	0,06274	0,89153
	6	21,00268	29,52386	482,01556	0,06125	0,88141
	7	6,44615	28,84184	454,53015	0,06345	0,91604
	8	21,54700	27,07903	426,68311	0,06346	0,87958
	9	14,04008	24,31009	378,17304	0,06428	0,87142
	10	20,05164	34,68263	559,68300	0,06197	0,88231

Tabela 11: Valores de $rmsd$, MI, H, MI/H e r para o complexo 3ZET A-B e seus modelos putativos gerados por diferentes servidores de *docking*.

Servidor	Modelo	rmsd	MI Total	H Total	MI/H	r
X	Nativo	0	28,85532	497,33500	0,05802	0,64111
GRAMMX	1	0,43047	31,44015	546,30296	0,05755	0,64642
	2	64,60040	15,13947	432,13853	0,03503	0,52564
	3	66,59539	13,71332	399,93281	0,03429	0,49656
	4	29,96139	25,68105	414,76380	0,06192	0,60743
	5	12,29256	26,69517	478,77351	0,05576	0,68441
	6	78,31339	10,60642	316,50830	0,03351	0,38341
	7	25,31356	25,19012	403,40594	0,06244	0,64641
	8	35,31819	24,02341	416,52421	0,05768	0,60632
	9	65,75690	13,15449	392,97683	0,03347	0,59157
	10	25,63438	19,25037	372,63632	0,05166	0,49709
ZDOCK	1	2,10475	32,03649	564,12867	0,05679	0,64782
	2	7,29972	28,64543	477,77104	0,05996	0,63043
	3	53,16295	13,65987	288,47695	0,04735	0,59665
	4	68,05671	14,56592	366,29877	0,03977	0,41997
	5	6,92726	25,95168	466,52055	0,05563	0,64338
	6	27,23154	12,12818	249,70831	0,04857	0,46983
	7	33,07157	14,61626	303,51275	0,04816	0,54635
	8	19,63485	20,94391	449,87199	0,04656	0,63486
	9	23,83879	13,85898	285,43187	0,04855	0,41451
	10	31,54210	19,97404	431,00197	0,04634	0,61066

Tabela 12: Valores de $rmsd$, MI, H, MI/H e r para o complexo 5F5S A-B e seus modelos putativos gerados por diferentes servidores de *docking*.

Servidor	Modelo	rmsd	MI Total	H Total	MI/H	r
X	Nativo	0	8,01794	108,62972	0,07381	0,80688
GRAMMX	1	50,45901	14,54184	207,41438	0,07011	0,57638
	2	46,65118	11,48481	162,41179	0,07071	0,67702
	3	43,43495	18,55848	246,90388	0,07516	0,76245
	4	37,97095	16,76304	222,15205	0,07546	0,76254
	5	56,41646	14,42020	207,63042	0,06945	0,62604
	6	44,92567	12,71619	179,96186	0,07066	0,72074
	7	2,32890	7,67737	104,34404	0,07358	0,80779
	8	28,55534	14,64768	187,70032	0,07804	0,61238
	9	46,17104	14,19401	192,64831	0,07368	0,68206
	10	53,62406	14,41210	201,18389	0,07164	0,31888
ROSIE	1	8,02861	11,10131	149,21043	0,07440	0,81312
	2	16,97367	14,57556	210,63797	0,06920	0,84898
	3	4,76123	9,42620	127,55357	0,07390	0,78492
	4	16,77915	14,13519	200,43005	0,07052	0,83106
	5	22,11725	13,30533	175,31961	0,07589	0,83293
	6	4,56610	8,41944	118,61003	0,07098	0,81574
	7	6,94566	7,88336	108,13187	0,07291	0,82212
	8	10,86190	8,76935	123,18831	0,07119	0,78648
	9	4,78419	7,09441	88,68157	0,08000	0,74800
	10	21,90630	14,97299	205,00553	0,07304	0,83380
ZDOCK	1	3,87037	12,12434	164,50337	0,07370	0,82253
	2	10,35034	10,50245	145,79004	0,07204	0,73258
	3	1,88671	10,11254	136,25206	0,07422	0,78465
	4	59,38614	12,18203	179,27685	0,06795	0,73913
	5	64,96775	9,42654	122,11796	0,07719	0,54396
	6	36,34544	6,69380	94,95544	0,07049	0,67270
	7	29,95659	13,55000	188,71636	0,07180	0,70952
	8	39,37932	10,11326	134,53225	0,07517	0,56061
	9	3,58734	7,13864	96,47515	0,07399	0,83187
	10	67,17562	7,08451	97,25967	0,07284	0,57959

Tabela 13: Valores de $rmsd$, MI, H, MI/H e r para o complexo 3OAA G-H e seus modelos putativos gerados por diferentes servidores de *docking*.

Servidor	Modelo	rmsd	MI Total	H Total	MI/H	r
X	Nativo	0	35,34395	904,88508	0,03906	0,70203
GRAMMX	1	0,42828	38,03862	976,85205	0,03894	0,70819
	2	6,33635	16,91599	390,29576	0,04334	0,59431
	3	71,79626	18,83839	467,47762	0,04030	0,57958
	4	33,59546	18,95677	535,70719	0,03539	0,55528
	5	6,73998	20,64433	546,94941	0,03774	0,68221
	6	60,40608	17,29731	558,06989	0,03099	0,51420
	7	52,65519	14,75468	367,77388	0,04012	0,53131
	8	50,58595	21,03575	455,03272	0,04623	0,55197
	9	47,19949	13,88728	355,67075	0,03905	0,60723
	10	25,18080	15,20946	487,28558	0,03121	0,45998
HDOCK	1	1,32045	35,94527	920,46027	0,03905	0,70025
	2	1,40377	37,52128	961,24310	0,03903	0,70960
	3	4,52583	19,90732	531,07366	0,03749	0,54282
	4	45,79199	15,29435	348,18483	0,04393	0,45883
	5	28,93011	9,62691	360,93489	0,02667	0,51508
	6	69,77486	10,96262	280,77683	0,03904	0,32641
	7	44,98163	14,86268	362,18307	0,04104	0,51016
	8	44,96164	8,14047	292,81214	0,02780	0,34003
	9	45,12724	11,88700	407,71999	0,02915	0,53702
	10	50,00306	8,44011	296,87395	0,02843	0,56086
ROSIE	1	5,75930	12,82718	355,82963	0,03605	0,50835
	2	21,68804	10,82790	355,17545	0,03049	0,40360
	3	15,13461	11,53729	312,76497	0,03689	0,36437
	4	7,90942	8,54524	270,34056	0,03161	0,52344
	5	10,04947	8,40394	270,53090	0,03106	0,63094
	6	12,28738	8,42635	240,65776	0,03501	0,51922
	7	44,58963	8,28701	270,81217	0,03060	0,33114
	8	27,53770	10,65456	301,14461	0,03538	0,58354
	9	6,25770	11,27426	309,48381	0,03643	0,36360
	10	23,34147	8,01973	262,97332	0,03050	0,46092
ZDOCK	1	2,10751	40,72060	1001,39778	0,04066	0,69310
	2	39,40734	7,80339	284,92587	0,02739	0,49002
	3	48,53128	9,52347	343,92681	0,02769	0,50953
	4	41,26620	12,86115	346,96735	0,03707	0,57705
	5	48,05610	13,13594	413,27600	0,03178	0,54541
	6	51,81950	17,72614	532,44142	0,03329	0,62564
	7	18,33218	17,43972	514,36118	0,03391	0,62538
	8	5,51482	24,61097	671,96898	0,03663	0,68491
	9	45,98769	11,97129	402,70756	0,02973	0,59346
	10	27,79308	14,34667	418,24659	0,03430	0,60904

Tabela 14: Valores de $rmsd$, MI, H, MI/H e r para o complexo 2Y69 A-C e seus modelos putativos gerados por diferentes servidores de *docking*.

Servidor	Modelo	rmsd	MI Total	H Total	MI/H	r
X	Nativo	0	56,53563	995,90066	0,05677	0,82876
GRAMMX	1	0,41135	62,47407	1087,59008	0,05744	0,82962
	2	74,97321	29,05911	725,80809	0,04004	0,68794
	3	56,83171	20,97178	528,94200	0,03965	0,74909
	4	37,64346	30,40657	690,98769	0,04400	0,80859
	5	43,14142	33,74801	615,62561	0,05482	0,80849
	6	34,50925	27,55703	478,62319	0,05758	0,75647
	7	64,31672	27,78356	666,16813	0,04171	0,77415
	8	50,63936	19,64479	443,95649	0,04425	0,65974
	9	60,68787	21,64756	542,81674	0,03988	0,50264
	10	73,93504	22,29979	428,40804	0,05205	0,81612
HDOCK	1	0,34877	56,77450	995,23994	0,05705	0,82654
	2	0,51662	59,49817	1041,77554	0,05711	0,83159
	3	69,65576	24,20364	410,24477	0,05900	0,80838
	4	43,60828	21,31466	444,95666	0,04790	0,73298
	5	34,30569	21,63994	384,68332	0,05625	0,69175
	6	69,07533	17,24054	310,00138	0,05561	0,80945
	7	74,01598	19,87136	377,13686	0,05269	0,78499
	8	72,67232	14,19735	293,33262	0,04840	0,71220
	9	71,17114	16,69119	433,98444	0,03846	0,64372
	10	75,28078	18,99209	388,89438	0,04884	0,80215
ROSIE	1	1,16503	44,92876	798,90435	0,05624	0,81595
	2	1,78713	40,77247	747,21572	0,05457	0,80034
	3	1,79877	42,14717	734,94336	0,05735	0,81208
	4	1,81334	39,99065	702,73786	0,05691	0,81523
	5	1,77968	38,14414	669,04079	0,05701	0,79943
	6	1,70231	33,29072	573,66563	0,05803	0,80497
	7	4,85194	25,34251	464,42233	0,05457	0,76677
	8	3,07295	28,87428	472,15730	0,06115	0,74907
	9	3,59543	24,12585	421,96791	0,05717	0,77340
	10	37,10186	20,10165	376,12457	0,05344	0,60489
ZDOCK	1	1,65884	65,48764	1104,82424	0,05927	0,84470
	2	76,33500	26,18530	553,52395	0,04731	0,70236
	3	70,11675	23,46699	411,87060	0,05698	0,78681
	4	74,50252	31,32446	624,91591	0,05013	0,76680
	5	73,51429	35,42714	679,35906	0,05215	0,81468
	6	70,64044	33,80355	829,48550	0,04075	0,76467
	7	44,11952	16,44876	504,54933	0,03260	0,57659
	8	68,23035	27,30657	458,05263	0,05961	0,83101
	9	69,79732	44,07557	765,86494	0,05755	0,77223
	10	31,80129	36,20859	669,56995	0,05408	0,82332

Anexo II – Código Fonte do DSM e UML

Docking_Score_Module.py

```
from MainWindow import MainWindow
from PyQt5 import QtWidgets
import sys

def main():

    app = QtWidgets.QApplication(sys.argv)
    GUI = MainWindow()
    GUI.show()
    sys.exit(app.exec_())

main()
```

MainWindow.py

```
from Manager import Manager
from Docking import Docking
from MSA import MSA
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import *
import pyqtgraph as pg
import datetime as dt
from time import sleep
import traceback, sys
import numpy as np

class WorkerSignals(QObject):

    finished = pyqtSignal()
    error = pyqtSignal(tuple)
    progress = pyqtSignal(int)



class Worker(QRunnable):


    def __init__(self, function):
        super(Worker, self).__init__()
        self.function = function
        self.signals = WorkerSignals()

    @pyqtSlot()
    def run(self):
        try:
            self.function()

        except:
            traceback.print_exc()
            exctype, value = sys.exc_info()[:2]
            self.signals.error.emit((exctype, value, traceback.format_exc()))
```

```
        finally:
            self.signals.finished.emit()


class MainWindow(QWidget):


    """
    Define the interface for Docking Score Module.

    """

    def __init__(self):
        super() .__init__()

        self.initUI()
        self.threadpool = QThreadPool()
        print("Multithreading with maximum %d threads" %
self.threadpool.maxThreadCount())
        self.__manager_objct = Manager()


    def initUI(self):

        self.setGeometry(100, 100, 900, 700)
        self.setWindowTitle("Docking Score Module")
        self.setWindowIcon(QIcon('icon.png'))


    #define widgets

        #top layout labels
        self.__PDB_lbl = QLabel("Docking Models Coordinates (.pdb)", self)
        self.__msaA_lbl = QLabel("Protein A Multiple Sequence Alignment (.fas)", self)
        self.__msaB_lbl = QLabel("Protein B Multiple Sequence Alignment (.fas)", self)

        #mid layout labels
        self.__job = QLabel("Job name:")
        self.__values = QLabel("Set parameters:  ")
        self.__lambda = QLabel("\u03BB - Pseudocounter            =", self)
        self.__theta = QLabel("\u03B8 - Sequence Similarity   =", self)
        self.__cutoff = QLabel("Contact Distance (\u00c5)" + 6 * " " + " =", self)
        self.__funcs = QLabel("Select functions:")

        #logo
        self.label = QLabel(self)
        self.__logo = QPixmap('logo.png')
        self.label.setPixmap(self.__logo)

        #buttons
        self.__PDB_btn = QtWidgets.QPushButton("Select...", self)
        self.__PDB_btn.setFixedWidth(80)
        self.__PDB_btn.clicked.connect(self.openPDBs)

        self.__msaA_btn = QtWidgets.QPushButton("Select...", self)
        self.__msaA_btn.setFixedWidth(80)
        self.__msaA_btn.clicked.connect(self.openMsaA)

        self.__msaB_btn = QtWidgets.QPushButton("Select...", self)
        self.__msaB_btn.setFixedWidth(80)
        self.__msaB_btn.clicked.connect(self.openMsaB)
```

```
        self.__save_job_btn = QtWidgets.QPushButton("Save Job", self)
        self.__save_job_btn.setEnabled(False)
        self.__save_job_btn.clicked.connect(self.save_job)
        self.__save_job_btn.setStyleSheet("; border: 2px solid white;")

        self.__save_plot_btn = QtWidgets.QPushButton("Save Plot", self)
        self.__save_plot_btn.setEnabled(False)
        self.__save_plot_btn.clicked.connect(self.save_plot)
        self.__save_plot_btn.setStyleSheet("; border: 2px solid white;")

        self.__runButton = QtWidgets.QPushButton("Run!", self)
        self.__runButton.setStyleSheet("background-color:  #e6e6e6; border: 2px solid
white; color: #ffffff")
        self.__runButton.setMinimumHeight(40)
        self.__runButton.setEnabled(False)

        #text boxes
        self.__jobTxt = QLineEdit(self)
        self.__jobTxt.setStyleSheet("background-color: #ffffff")

        self.__lambdaBox = QDoubleSpinBox(self)
        self.__lambdaBox.setStyleSheet("background-color: #ffffff")
        self.__lambdaBox.setRange(0.1, 1)
        self.__lambdaBox.setDecimals(2)
        self.__lambdaBox.setSingleStep(0.1)
        self.__lambdaBox.setValue(0.5)
        self.__lambdaBox.setFixedWidth(60)

        self.__thetaBox = QDoubleSpinBox(self)
        self.__thetaBox.setStyleSheet("background-color: #ffffff")
        self.__thetaBox.setRange(0.1, 1)
        self.__thetaBox.setDecimals(2)
        self.__thetaBox.setSingleStep(0.1)
        self.__thetaBox.setValue(0.8)
        self.__thetaBox.setFixedWidth(60)

        self.__cutoffBox = QSpinBox(self)
        self.__cutoffBox.setStyleSheet("background-color: #ffffff")
        self.__cutoffBox.setRange(0, 50)
        self.__cutoffBox.setValue(8)
        self.__cutoffBox.setFixedWidth(60)

        #diagnostics_widget
        self.__txt = QTextEdit(self)
        self.__txt.setStyleSheet("color: #ffffff; background-color: #000000; border: 2px
solid white")
        self.__txt.setReadOnly(True)
        self.__txt.setFixedWidth(500)

        QToolTip.setFont(QFont('SansSerif', 10))
        self.__lambdaBox.setToolTip("Pseudocounter value. Must be between <b>0.1</b> and
<b>1</b>.")
        self.__thetaBox.setToolTip("Sequence similarity value. Must be between <b>0.1</b>
and <b>1</b>.")
        self.__cutoffBox.setToolTip("Distance between interprotein amino acids wich forms
a contact.")

        #checkboxes
        self.__cbMI = QtWidgets.QCheckBox("Mutual Information", self)
        self.__cbMI.toggle()

        #self.__cbDI = QtWidgets.QCheckBox("Direct Information", self)
        #self.__cbDI.toggle()
```

```
        self.__cbr = QtWidgets.QCheckBox("Linear Correlation", self)
        self.__cbr.toggle()

    #define layout

        #top left
        for widget in [self.__PDB_lbl, self.__msaA_lbl,
self.__msaB_lbl]:widget.setMinimumHeight(30)

        self.__topgroupbox = QtWidgets.QGroupBox("User Upload")
        self.__topgroupbox.setStyleSheet("background-color: #e6e6e6")

        self.__uploads_grid = QtWidgets.QGridLayout()
        self.__uploads_grid.addWidget(self.__PDB_lbl, 0, 0)
        self.__uploads_grid.addWidget(self.__msaA_lbl, 1, 0)
        self.__uploads_grid.addWidget(self.__msaB_lbl, 2, 0)
        self.__uploads_grid.addWidget(self.__PDB_btn, 0, 1)
        self.__uploads_grid.addWidget(self.__msaA_btn, 1, 1)
        self.__uploads_grid.addWidget(self.__msaB_btn, 2, 1)

        self.__topgroupbox.setLayout(self.__uploads_grid)

        #mid left
        for widget in [self.__jobTxt, self.__lambdaBox, self.__thetaBox,
                       self.__cutoffBox, self.__cbMI, self.__cbr] :
widget.setMinimumHeight(30)

        self.__midgroupbox = QtWidgets.QGroupBox("User Preferences")
        self.__midgroupbox.setStyleSheet("background-color: #e6e6e6")

        self.__param_grid = QtWidgets.QGridLayout()
        self.__param_grid.addWidget(self.__job, 0, 0)
        self.__param_grid.addWidget(self.__jobTxt, 0, 1)
        self.__param_grid.addWidget(self.__values, 1, 0)
        self.__param_grid.addWidget(self.__lambda, 1, 1)
        self.__param_grid.addWidget(self.__lambdaBox, 1, 2)
        self.__param_grid.addWidget(self.__theta, 2, 1)
        self.__param_grid.addWidget(self.__thetaBox, 2, 2)
        self.__param_grid.addWidget(self.__cutoff, 3, 1)
        self.__param_grid.addWidget(self.__cutoffBox, 3, 2)
        self.__param_grid.addWidget(self.__funcs, 4, 0)
        self.__param_grid.addWidget(self.__cbMI, 4, 1)
        self.__param_grid.addWidget(self.__cbr, 5, 1)
        #self.__param_grid.addWidget(self.__cbDI, 6, 1)
        self.__midgroupbox.setLayout(self.__param_grid)

        self.__save_btn_hlayout = QtWidgets.QHBoxLayout()
        self.__save_btn_hlayout.addWidget(self.__save_job_btn)
        self.__save_btn_hlayout.addWidget(self.__save_plot_btn)

        #bottom - left
        self.__bottom_hlayout = QtWidgets.QHBoxLayout()
        self.__bottom_hlayout.addStretch(0)
        self.__bottom_hlayout.addWidget(self.label)
        self.__bottom_hlayout.addStretch(0)

        #general - left
        self.__left_vlayout = QtWidgets.QVBoxLayout()
        self.__left_vlayout.addWidget(self.__topgroupbox)
        self.__left_vlayout.addWidget(self.__midgroupbox)
        self.__left_vlayout.addWidget(self.__runButton)
        self.__left_vlayout.addLayout(self.__save_btn_hlayout)
        self.__left_vlayout.addLayout(self.__bottom_hlayout)
```

```
        #general - right

        self.__rightgroupbox = QtWidgets.QGroupBox("Diagnostics")
        self.__rightgroupbox.setStyleSheet("background-color: #e6e6e6")

        self.__right_grid = QtWidgets.QGridLayout()
        self.__right_grid.addWidget(self.__txt, 0, 0)

        self.__rightgroupbox.setLayout(self.__right_grid)

        self.__right_vlayout = QtWidgets.QVBoxLayout()
        self.__right_vlayout.addWidget(self.__rightgroupbox)

        #general layout
        self.__gen_hlayout = QtWidgets.QHBoxLayout()
        self.__gen_hlayout.addLayout(self.__left_vlayout)
        self.__gen_hlayout.addLayout(self.__right_vlayout)

        self.setLayout(self.__gen_hlayout)


    #define actions

    def button_manager(self):
        self.__runButton.disconnect()
        self.__runButton.clicked.connect(self.run)
        if self.__manager_objct.valid_align_A and self.__manager_objct.valid_align_B and
self.__manager_objct.valid_dockings:
            self.__runButton.setText("Run")
            self.__runButton.setStyleSheet("background-color:  #00cc99; border: 2px solid
white; color: #ffffff")
            self.__runButton.setEnabled(True)
            self.__running = False


    def turn_cancel_btn(self):
        self.__runButton.disconnect()
        self.__runButton.setText("Cancel")
        self.__runButton.setStyleSheet("background-color:  #ff3300; border: 2px solid
white; color: #ffffff")
        self.__runButton.clicked.connect(self.warning)


    def warning(self):
        buttonReply = QMessageBox.question(self, 'Cancel Run', "Are you sure?",
                                           QMessageBox.Yes | QMessageBox.No,
QMessageBox.No)
        if buttonReply == QMessageBox.Yes:
            self.cancel()


    def cancel(self):
        time = d = dt.datetime.now()
        msg = "\n>> Cancel Job:\n>> {} interrupted at
{}\n".format(self.__manager_objct.job_name,
                                                                    d.strftime("%H:%M:
%S"))
        self.update_diagnostics(self.__manager_objct.txt + msg)
        self.__runButton.setEnabled(False)
        self.widgets_on_run(True)
        self.__manager_objct.cancel = True
        self.button_manager()
```

```
    def turn_plot_btn(self):

        if not self.__manager_objct.cancel:
            self.__manager_objct.output()
            self.update_diagnostics(self.__manager_objct.txt)
            self.__runButton.disconnect()
            self.__runButton.setText("Plot")
            self.__runButton.setStyleSheet("background-color:  #3399ff; border: 2px solid
white; color: #ffffff")
            self.__runButton.clicked.connect(self.plot)


    def turn_restart_btn(self):

        self.__runButton.disconnect()
        self.__runButton.setText("Restart")
        self.__runButton.setStyleSheet("background-color:  #ffcc00; border: 2px solid
white; color: #ffffff")
        self.__runButton.clicked.connect(self.restart())


    def restart(self):

        self.__manager_objct.restart
        self.button_manager()


    def plot(self):

       self.__plotWidget = pg.plot(self.__manager_objct.r_list,
self.__manager_objct.mi_list, size=10, pen=pg.mkPen(None), symbol='o',
brush=pg.mkBrush(0, 191, 255, 100), title="Docking Score Module -
{}".format(self.__manager_objct.job_name))
        self.__plotWidget.setLabel("left", "MI/H (nats)")
        self.__plotWidget.setLabel("bottom", "r")

        for i, mi in enumerate(self.__manager_objct.mi_list):
            label = pg.TextItem(text=str(self.__manager_objct.models_list[i]),
color=(200, 200, 200), html=None, anchor=(0,0), border=None, fill=None, angle=0,
rotateAxis=None)
            label.setPos(self.__manager_objct.r_list[i], self.__manager_objct.mi_list[i])
            self.__plotWidget.addItem(label)

        self.__save_plot_btn.setEnabled(True)
        self.turn_restart_btn()


    def update_diagnostics(self, txt):

        self.__txt.setText(txt)


    def finish_job(self):

        self.turn_plot_btn()
        self.__save_job_btn.setEnabled(True)


    def openPDBs(self):

        options = QFileDialog.Options()
        options |= QFileDialog.DontUseNativeDialog
        docking_list, _ = QFileDialog.getOpenFileNames(self,"Select docking solutions",
```

```
                                                    "","PDB Files (*.pdb);;All Files
(*)",
                                                    options=options)
        if docking_list:
            self.__manager_objct.set_docking_list(docking_list)
            self.update_diagnostics(self.__manager_objct.txt)
            self.button_manager()


    def openMsaA(self):

        options = QFileDialog.Options()
        options |= QFileDialog.DontUseNativeDialog
        alignment, _ = QFileDialog.getOpenFileName(self,"Select chain A alignment",
                                                   "","FASTA Files (*.fas);;All Files
(*)",
                                                   options=options)
        if alignment:
            self.__manager_objct.MSA_A(alignment)
            self.update_diagnostics(self.__manager_objct.txt)
            self.button_manager()


    def openMsaB(self):

        options = QFileDialog.Options()
        options |= QFileDialog.DontUseNativeDialog
        alignment, _ = QFileDialog.getOpenFileName(self,"Select chain B alignment",
                                                   "","FASTA Files (*.fas);;All Files
(*)",
                                                   options=options)
        if alignment:
            self.__manager_objct.MSA_B(alignment)
            self.update_diagnostics(self.__manager_objct.txt)
            self.button_manager()

    def savefile(self):

        options = QFileDialog.Options()
        options |= QFileDialog.DontUseNativeDialog
        filename, _ = QFileDialog.getSaveFileName(self,"Save Job",
                                                  self.__manager_objct.job_name,"All
Files (*);;Text Files (*.dat)", options=options)

        return filename

    def save_job(self):

        filename = self.savefile()

        if filename:
            self.__manager_objct.savejob(filename)


    def save_plot(self):

        options = QFileDialog.Options()
        options |= QFileDialog.DontUseNativeDialog
        filename, _ = QFileDialog.getSaveFileName(self,"Save Plot",
"{}.svg".format(self.__manager_objct.job_name),"svg Files (*);;Image Files (*.svg)",
options=options)

        self.__plotWidget.writeSvg(filename)
```

```
    def checkParameters(self):

        self.update_diagnostics(self.__manager_objct.txt)
        self.__manager_objct.job_name = self.__jobTxt.text()
        self.__manager_objct.set_lambda(self.__lambdaBox.value())
        self.__manager_objct.set_theta(self.__thetaBox.value())
        self.__manager_objct.set_cutoff(self.__cutoffBox.value())

        if self.__cbMI.isChecked():
            self.__manager_objct.set_MI(True)
        #if self.__cbDI.isChecked():
            #self.__manager_objct.set_DI(True)
        if self.__cbr.isChecked():
            self.__manager_objct.set_r(True)

    def widgets_on_run(self, T):

        self.__PDB_btn.setEnabled(T)
        self.__msaA_btn.setEnabled(T)
        self.__msaB_btn.setEnabled(T)
        self.__jobTxt.setEnabled(T)
        self.__lambdaBox.setEnabled(T)
        self.__thetaBox.setEnabled(T)
        self.__cutoffBox.setEnabled(T)
        self.__cbMI.setEnabled(T)
        #self.__cbDI.setEnabled(T)
        self.__cbr.setEnabled(T)

    def instantiate(self):

        self.__manager_objct.run()

    def run(self):

        self.turn_cancel_btn()
        self.widgets_on_run(False)
        self.checkParameters()
        self.__manager_objct.calculate_Meff
        self.update_diagnostics(self.__manager_objct.txt)
        worker = Worker(self.instantiate)
        worker.signals.finished.connect(self.finish_job)
        self.threadpool.start(worker)
```

Manager.py

```
from PDBReader import PDBReader
from MSA import MSA
from Verifier import Verifier
from Docking import Docking
from Coevolution import Coevolution
from Output import Output
from PyQt5.QtCore import *
from Bio import AlignIO, SeqIO
import datetime as dt
from scipy import spatial
import numpy as np
import multiprocessing as mp

class Manager(QMutex):
```

```
    """
    Manage rotines of API and interact with UI.

    """

    def __init__(self):

        super(Manager, self).__init__()
        self.__mtx = QMutex()
        self.__cancel = False
        self.__txt = ""
        self.__docking_list = []
        self.__MSA = MSA()
        self.__checker = Verifier()
        self.__pdbReader_list = []
        self.__valid_dockings = False
        self.__valid_align_A = False
        self.__valid_align_B = False
        self.__job_name = None
        self.__lambda = 0.5
        self.__theta = 0.8
        self.__cutoff = 8
        self.__MI = False
        self.__DI = False
        self.__r = False
        self.__docking_results = []


    @property
    def cancel(self):
        self.__mtx.lock()
        cancel = self.__cancel
        self.__mtx.unlock()
        return cancel


    @cancel.setter
    def cancel(self, T):
        self.__mtx.lock()
        self.__cancel = T
        self.__mtx.unlock()


    def update_txt(self, msg):
        self.__txt += "dsm >> {}\n".format(msg)


    @property
    def txt(self):
        return self.__txt


    def MSA_A(self, MSA_file):

        msa =  self.__MSA.set_MSA_A(MSA_file)

        msg1 = "MSA A file: {}\n>> {} sequences".format(MSA_file,
self.__MSA.MSA_A_numseqs)
        msg2 = "A alignment and pdb A protein are incompatibles"
        msg3 = "MSA A file invalid. Enter a multiple sequence alignment file"

        if msa:
            if self.__valid_dockings:
```

```
                checker = Verifier()
                if checker.validate_MSA(self.__pdbReader_list[0].fasta[0],
                                        self.__MSA.ref_msa_A):
                    self.__valid_align_A = True
                    self.update_txt(msg1)
                else: self.update_txt(msg2)
            else: self.update_txt(msg1)
        else: self.update_txt(msg3)


    def MSA_B(self, MSA_file):

        msa = self.__MSA.set_MSA_B(MSA_file)

        msg1 = "MSA B file: {}\n>> {} sequences".format(MSA_file,
self.__MSA.MSA_B_numseqs)
        msg2 = "B alignment and pdb B protein are incompatibles"
        msg3 = "MSA B file invalid. Enter a multiple sequence alignment file"

        if msa:
            if self.__valid_dockings:
                checker = Verifier()
                if checker.validate_MSA(self.__pdbReader_list[0].fasta[1],
                                        self.__MSA.ref_msa_B):
                    self.__valid_align_B = True
                    self.update_txt(msg1)
                else: self.update_txt(msg2)
            else: self.update_txt(msg1)
        else: self.update_txt(msg3)


    @property
    def calculate_Meff(self):
        cv = Coevolution()
        cv.set_theta(self.__theta)
        cv.Meff(self.__MSA)
        self.update_txt("\u03B8 = {}; Meff = {}".format(self.__theta, self.__MSA.Meff))


    def set_docking_list(self, docking_list):

        """ Instantiate PDBReaders objects from  docking list. """

        self.__docking_list = docking_list
        self.__pdbReader_list = [PDBReader(docking) for docking in docking_list]

        msg1 = "docking files: {}".format(docking_list)
        msg2 = "docking models must have exactly same proteins."
        msg3 = "A alignment and pdb A protein are incompatibles"
        msg4 = "B alignment and pdb B protein are incompatibles"
        checker = Verifier()

        val_models = checker.validate_models(self.__pdbReader_list)
        val_A = True
        val_B = True

        if self.__valid_align_A and val_models:
            if not checker.validate_MSA(self.__pdbReader_list[0].fasta[0],
self.__MSA.ref_msa_A):
                val_A = False
                self.update_txt(msg3)
        if self.__valid_align_B and val_models:
            if not checker.validate_MSA(self.__pdbReader_list[0].fasta[1],
self.__MSA.ref_msa_B):
```

```
                val_B = False
                self.update_txt(msg4)

        if val_models and val_A and val_B:
            self.__valid_dockings = True
            self.update_txt(msg1)
        else: self.update_txt(msg2)


    def make_calculations(self, docking_file):

        """ Instantiate Coevolution object for calculations. """

        coordinates = PDBReader(docking_file)
        docking = Docking(coordinates, self.__cutoff)
        docking.name = docking_file.split("/")[-1]
        cv = Coevolution()
        cv.set_lambda(self.__lambda)

        if self.__MI and not self.cancel:
            cv.MI(self.__MSA, docking)
        if self.__DI and not self.cancel:
            cv.DI(self.__MSA, docking)
        if self.__r and not self.cancel:
            msa_a = self.__MSA.MSA_A_fasta(docking.A_ICs, docking.name)
            msa_b = self.__MSA.MSA_B_fasta(docking.B_ICs, docking.name)
            docking.r = cv.correlation(msa_a, msa_b)
        return(docking)


    def run(self):

        """ Instantiate Pool object for paralelize calculations. """

        pool = mp.Pool()
        self.__docking_results = pool.map(self.make_calculations, self.__docking_list)
        pool.close()
        pool.join()
        print("look! I'm running!!")


    def output(self):

        """ Print results of calculations for each docking model. """

        self.__out = Output(self.__job_name,
                     self.__docking_results,
                     self.__theta,
                     self.__lambda,
                     self.__MI,
                     self.__DI,
                     self.__r)

        self.update_txt("Results:")
        for i, model in enumerate(self.__out.models):

            msg = str(model)

            if self.__MI:
                msg += "  MI = {}  ".format(self.__out.mis[i])
            if self.__DI:
                msg += "  DI = {}  ".format(self.__out.rs[i])
            if self.__r:
                msg += "  r = {}  ".format(self.__out.rs[i])
```

```
            self.update_txt(msg)

        print(self.__out.scoring_list)


    def savejob(self, filename):

        self.__out.pairs_for_vmd(filename)
        self.__out.w_file(filename)


    @property
    def restart(self):
        pass


    @property
    def job_name(self):
        return self.__job_name


    @job_name.setter
    def job_name(self, name):
        if name:
            self.__job_name = name
        else:
            d = dt.datetime.now()
            self.__job_name = "j_{}".format(d.strftime("%Y%m%d%H%M%S"))
        self.update_txt("Job {}:".format(self.__job_name))


    def set_lambda(self, value):
        self.__lambda = value


    def set_theta(self, value):
        self.__theta = value


    def set_cutoff(self, value):
        self.__cutoff = value


    def set_MI(self, T):
        self.__MI = T


    def set_DI(self, T):
        self.__DI = T


    def set_r(self, T):
        self.__r = T


    def set_dfile(self, T):
        self.__gen_dfile = T


    def set_plot(self, T):
        self.__plot = T
```

```
    @property
    def valid_dockings(self):
        return self.__valid_dockings


    @property
    def valid_align_A(self):
        return self.__valid_align_A


    @property
    def valid_align_B(self):
        return self.__valid_align_B


    @property
    def models_list(self):
        return self.__out.models


    @property
    def mi_list(self):
        return self.__out.mis


    @property
    def di_list(self):
        return self.__out.dis


    @property
    def r_list(self):
        return self.__out.rs
```

PDBReader.py

```
from Bio.PDB import *
from Bio.PDB.Polypeptide import three_to_one


class PDBReader():


    """
    Parse PDB file

    """


    def __init__(self, pdb_file):

        """
        Create PDBREader object

        """

        self.__complex = str(pdb_file)
        self.__model = None
        self.__res_list = [[], []]
        self.__fasta_res_list = []
```

```
        self.__res_id_list = []
        self.__parser()
        self.__get_ids_list()


    def __parser(self):

        """ Instance a PDBParser object and make a list of residues for each chain. """

        pdb = PDBParser(QUIET=True)
        structure = pdb.get_structure(self.__complex, self.__complex)
        self.__model = structure[0]

        chain_res_list = [Selection.unfold_entities(chain, "R") for chain in
self.__model]
        for i, chain in enumerate(chain_res_list):
            for aa in chain:
                if is_aa(aa.get_resname(), standard=True):
                    self.__res_list[0].append(aa) if i == 0 else
self.__res_list[1].append(aa)


    def __get_ids_list(self):

        """ Get a list of residues ids. """

        for chain in self.__res_list:
            res_id = []
            for residue in chain:
                if is_aa(residue.get_resname(), standard=True):
                    res_id.append(residue.get_id()[1])
            self.__res_id_list.append(res_id)

    @property
    def complex(self):
        return self.__complex

    @property
    def chain_residues_list(self):

        """ Return a list of a residues list for each chain. """

        return self.__res_list


    @property
    def id_list(self):

        """ Return a list of a residues list ids for each chain. """

        return self.__res_id_list


    @property
    def missing_ids(self):

        """ Get missing ids in pdb. """

        missing_ids = []

        for idx in (self.__res_id_list):
            start, end = idx[0], idx[-1]
            missing_ids.append(sorted(set(range(start, end + 1)).difference(idx)))
```

```
        return missing_ids

    @property
    def fasta(self):

        """ Return a list of one letter residues list for each chain"""

        for chain in self.__res_list:
            fasta_chain = []
            for res in chain:
                if is_aa(res.get_resname(), standard=True):
                    fasta_chain.append(three_to_one(res.get_resname()))

            fasta_chain = ''.join(fasta_chain)
            self.__fasta_res_list.append(fasta_chain)

        return self.__fasta_res_list

    @property
    def atoms(self):
        return Selection.unfold_entities(self.__model, "A")
```

MSA.py

```
from Bio import AlignIO
import numpy as np
import os

class MSA:

    """
    Receive a MSA.fasta alignment, parse and return the positions of information
channels.

    """

    def __init__(self):


        self.__MSA_A_file = None
        self.__MSA_B_file = None
        self.__MSA_A = None
        self.__MSA_B = None
        self.__encoded_MSA_A = np.empty((0, 0))
        self.__encoded_MSA_B = np.empty((0, 0))
        self.__aa_code = {"A": 0, "R": 1, "N": 2, "D": 3, "Q": 4,
                          "E": 5, "G": 6, "H": 7, "L": 8, "K": 9,
                          "M": 10, "F": 11, "S": 12, "T": 13, "W": 14,
                          "Y": 15, "C": 16, "I": 17, "P": 18, "V": 19,
                          "-": 20, ".": 20, "B": 2, "Z": 4, "X": 20, "J": 20}
        self.__q = 21
        self.__sequence_weight = np.empty([0, 0])
        self.__Meff = 0


    def __set_encoded(self, msa):
        encoded = np.empty((len(msa), len(msa[0])), dtype=np.int16)

        for idx, aa in np.ndenumerate(msa):
            aa = aa.upper()
```

```
            encoded[idx] = int(self.__aa_code[aa])

        return encoded

    def set_MSA_A(self, file_name):
        try:
            self.__MSA_A = AlignIO.read(file_name, "fasta")
        except:
            return False

        self.__MSA_A_file = file_name
        self.__encoded_MSA_A = self.__set_encoded(self.__MSA_A)
        if self.__MSA_B and len(self.__MSA_B) != len(self.__MSA_A):
            return False
        else:
            return True

    def set_MSA_B(self, file_name):
        try:
            self.__MSA_B = AlignIO.read(file_name, "fasta")
        except:
            return False

        self.__MSA_B_file = file_name
        self.__encoded_MSA_B = self.__set_encoded(self.__MSA_B)
        if self.__MSA_A and len(self.__MSA_A) != len(self.__MSA_B):
            return False
        else:
            return True

    def encoded_MSA(self, IC_list):

        encoded_msa = np.take(np.concatenate((self.__encoded_MSA_A,
self.__encoded_MSA_B), axis=1), IC_list, axis=1)
        return encoded_msa


    @property
    def ref_msa_A(self):
        a = self.__MSA_A[0].format("fasta").rsplit()[1:]
        a = "".join(a)
        return a


    def MSA_A_fasta(self, ics, name):

        temp = ""

        for seq in self.__MSA_A:
            temp += ">" + seq.id + "\n"
            line = ""
            for i in ics:
                line += seq[i]
            temp += line + "\n"
        with open("msa_A_{}.fas".format(name), 'w') as fl:
            fl.write(temp)
        fl.close()

        msa_A =  AlignIO.read("msa_A_{}.fas".format(name), "fasta")
        #os.remove("msa_A_{}.fas".format(name))
        return msa_A


    def MSA_A(self, ICs):
```

```
        return np.take(self.__encoded_MSA_A, ICs, axis=1)

    @property
    def MSA_A_sz(self):
        return self.__encoded_MSA_A.shape[1]

    @property
    def MSA_A_numseqs(self):
        return self.__encoded_MSA_A.shape[0]

    @property
    def ref_msa_B(self):
        b = self.__MSA_B[0].format("fasta").rsplit()[1:]
        b = "".join(b)
        return b

    def MSA_B_fasta(self, ics, name):

        temp = ""

        for seq in self.__MSA_B:
            temp += ">" + seq.id + "\n"
            line = ""
            for i in ics:
                line += seq[i]
            temp += line + "\n"
        with open("msa_B_{}.fas".format(name), 'w') as fl:
            fl.write(temp)
        fl.close()

        msa_B =  AlignIO.read("msa_B_{}.fas".format(name), "fasta")
        #os.remove("msa_B_{}.fas".format(name))
        return msa_B


    def MSA_B(self, ICs):
        return np.take(self.__encoded_MSA_B, ICs, axis=1)

    @property
    def MSA_B_sz(self):
        return self.__encoded_MSA_B.shape[1]

    @property
    def MSA_B_numseqs(self):
        return self.__encoded_MSA_B.shape[0]

    @property
    def num_of_seqs(self):
        return self.__encoded_MSA_A.shape[0]

    @property
    def full_encoded(self):
        return np.concatenate((self.__encoded_MSA_A, self.__encoded_MSA_B), axis=1)

    @property
    def sequence_weight(self):
        return self.__sequence_weight

    @sequence_weight.setter
    def sequence_weight(self, vector):
        self.__sequence_weight = vector

    @property
    def Meff(self):
```

```
        return self.__Meff

    @Meff.setter
    def Meff(self, Meff):
        self.__Meff = Meff

    @property
    def q(self):
        return self.__q

    @property
    def reset(self):

        self.__MSA_A_file = None
        self.__MSA_B_file = None
        self.__MSA_A = None
        self.__MSA_B = None
        self.__encoded_MSA_A = np.empty((0, 0))
        self.__encoded_MSA_B = np.empty((0, 0))
        self.__sequence_weight = np.empty([0, 0])
        self.__Meff = 0
```

Docking.Py

```
from Bio.PDB import *
import numpy as np

class Docking():


    """
    Contain the information about docking complex.

    """


    def __init__(self, PDBReader_Objct, cutoff = 8):

        """
        Create a Docking object.


        """

        self.__cutoff = cutoff
        self.__PDB_reader = PDBReader_Objct
        self.__chain_res_list = PDBReader_Objct.chain_residues_list
        self.__chain_A =  self.__chain_res_list[0]
        self.__chain_B =  self.__chain_res_list[1]
        self.__name = ""
        self.__atom_list = []
        self.__contact_pairs = []
        self.__ICs = []
        self.__MI = 0
        self.__h = 0
        self.__r = 0
        self.__DI = 0
        self.__ref_atoms()
        self.__contacts()
```

```
def __ref_atoms(self):

    """ Select a list of atoms of reference for NeighborSearch object. """

    for chain_res_list in self.__chain_res_list:
        for residue in chain_res_list:
            if is_aa(residue):
                if residue.has_id("CB"):
                    self.__atom_list.append(residue["CB"])
                else:
                    self.__atom_list.append(residue["CA"])


def correct_shift(self, residue):

    """ Correct residue id for iteration. """

    A_shift = self.__chain_A[0].get_id()[1]
    B_shift = self.__chain_B[0].get_id()[1]

    res_id = 0

    if residue in self.__chain_A:
        if self.__PDB_reader.missing_ids[0]:
            for i in self.__PDB_reader.missing_ids[0]:
                if residue.get_id()[1] > i:
                    res_id -= 1
        res_id += residue.get_id()[1] - A_shift

    else:
        if self.__PDB_reader.missing_ids[1]:
            for i in self.__PDB_reader.missing_ids[1]:
                if residue.get_id()[1] > i:
                    res_id -= 1
        res_id += residue.get_id()[1] - B_shift + len(self.__chain_A)

    return res_id


def __contacts(self):

    """ Make a list of contact pairs. """

    ns = NeighborSearch(self.__atom_list)

    for atom in self.__atom_list:
        if atom.get_parent() in self.__chain_A:
            neighbors = ns.search(atom.get_coord(), self.__cutoff, "R")

            for res in neighbors:
                if res not in self.__chain_A:
                    self.__contact_pairs.append((atom.get_parent(), res))

@property
def ICs(self):

    """ Return residues id of the complex interface. """


    flat_list = [res for pair in self.__contact_pairs for res in pair]
    ICs = list(set(map(self.correct_shift, flat_list)))
```

```
        return ICs

    @property
    def contact_map(self):

        """ Return contact map array with correct ids. """

        c_map = np.asarray([list(map(self.correct_shift, pair)) for pair in
self.__contact_pairs])
        return c_map


    @property
    def A_ICs(self):
        A = [i[0] for i in self.__contact_pairs]
        A = sorted(list(set(map(self.correct_shift, A))))
        return A

    @property
    def shift_B_ICs(self):
        B = [i[1] for i in self.__contact_pairs]
        B = sorted(list(set(map(self.correct_shift, B))))
        return B

    @property
    def B_ICs(self):
        B = [i[1] for i in self.__contact_pairs]
        B = sorted(list(set(map(self.correct_shift, B))))
        B = list(map(lambda x: x - len(self.__chain_A), B))
        return B

    @property
    def pairs_wtt_correction(self):
        pairs = [list(map(lambda x: x.get_id()[1], pair)) for pair in
self.__contact_pairs]
        return pairs

    @property
    def name(self):
        return self.__name

    @name.setter
    def name(self, name):
        self.__name = name

    @property
    def MI(self):
        return self.__MI

    @MI.setter
    def MI(self, value):
        self.__MI = value

    @property
    def h(self):
        return self.__h

    @h.setter
    def h(self, value):
        self.__h = value

    @property
    def MI_by_h(self):
        return self.__MI / self.__h
```

```
    @property
    def MI_by_npairs(self):
        return self.__MI / len(self.__contact_pairs)

    @property
    def r(self):
        return self.__r

    @r.setter
    def r(self, value):
        self.__r = value

    @property
    def DI(self):
        return self.__DI

    @DI.setter
    def DI(self, value):
        self.__DI = value

    @property
    def n_pairs(self):
        return len(self.__contact_pairs)
```

Verifier.py

```
from Bio.PDB import *
from Bio import pairwise2
from Bio import AlignIO, SeqIO
import Levenshtein

class Verifier():


    """
    Verify and correct the iteration of the PDB chain.

    """


    def __init__(self):
        pass

    def align_identity(self, seq1, seq2):

        """ Calculate distance between sequences. """

        distance = Levenshtein.ratio(seq1, seq2)

        return True if distance > 0.98 else False

    def is_alignment(self, align):

        align = AlignIO.read(align, "fasta")
        return True if align else False

    #TODO
    def sincronizes_gap(self):
```

```
        """ Sicronize gaps in MSA an PDB sequence. """

        pass

    def validate_models(self, pdbs):

        for model in pdbs:
            AIsEqual = self.align_identity(pdbs[0].fasta[0], model.fasta[0])
            BIsEqual = self.align_identity(pdbs[0].fasta[1], model.fasta[1])
            if not AIsEqual or not BIsEqual: return False

        return True

    def validate_MSA(self, pdb, msa):

        isEqual = self.align_identity(pdb, msa)
        return True if isEqual else False
```

Output.py

```
from Docking import Docking
from Bio.PDB import *
import matplotlib.pyplot as plt
import numpy as np
import glob, imp

class Output():


    """
    Manager output.

    """

    def __init__(self, jobname, model_list, theta, lambda_, valid_mi, valid_di, valid_r):
        self.__job_name = jobname
        self.__model_list = model_list
        self.__PDB_REF = "1bxr1.pdb"
        self.__models = []
        self.__valid_mi = valid_mi
        self.__valid_di = valid_di
        self.__valid_r = valid_r
        self.__mis = []
        self.__hs = []
        self.__mis_norm_h = []
        self.__n_pairs = []
        self.__mis_norm_pairs = []
        self.__dis = []
        self.__rs = []
        self.__parameters = {"t": theta, "l": lambda_}
        self.__make_lists()

    def __make_lists(self):

        for model in self.__model_list:
            self.__models.append(model.name)
            self.__mis.append(model.MI)
            self.__hs.append(model.h)
            self.__mis_norm_h.append(model.MI_by_h)
```

```
            self.__n_pairs.append(model.n_pairs)
            self.__mis_norm_pairs.append(model.MI_by_npairs)
            self.__dis.append(model.DI)
            self.__rs.append(model.r)

    def w_file(self, filename):

        with open("{}_t{}_l{}.dat".format(filename, round(self.__parameters["t"], 2),
round(self.__parameters["l"], 2)), 'w') as f:
            f.write("theta = {}\nlambda = {}\n".format(self.__parameters["t"],
self.__parameters["l"]))
            f.write("models = {}\n".format(self.__models))
            if self.__valid_mi:
                f.write("mi = {}\n".format(self.__mis))
                f.write("h = {}\n".format(self.__hs))
                f.write("mi/h = {}\n".format(self.__mis_norm_h))
                f.write("n_of_pairs = {}\n".format(self.__n_pairs))
                f.write("mi/npairs = {}\n".format(self.__mis_norm_pairs))
            if self.__valid_di:
                f.write("DIs = {}\n".format(self.__dis))
            if self.__valid_r:
                f.write("r = {}\n".format(self.__rs))

        f.close()

    @property
    def scoring_list(self):
        res_list = [(model.name, model.MI, model.DI, model.r) for model in
self.__model_list]
        return res_list

    @property
    def models(self):
        return self.__models

    @property
    def mis(self):
        return self.__mis

    @property
    def dis(self):
        return self.__dis

    @property
    def rs(self):
        return self.__rs
```

Coevolution.py

```
import math
import numpy as np
from scipy import spatial
from Bio.Phylo.TreeConstruction import DistanceCalculator
from Bio import AlignIO
import timeit


class Coevolution():
```

```
    """
    Implement direct coupling analisys for a docking solution.

    """


    def __init__(self):


        """ Create Coevolution object for calculations.  """


        self.__theta = 0
        self.__lambda = 0
        self.__site_freq = np.empty((0, 0))
        self.__pair_freq = np.empty((0, 0, 0, 0))



    def set_theta(self, value):
        self.__theta = value

    def set_lambda(self, value):
        self.__lambda = value


    def Meff(self, msa):


        """ Calculate effective number of independent sequences in msa, the sequences
weight and set these values in msa objct. """



        hamming_distance = spatial.distance.pdist(msa.full_encoded, "hamming")
        weight_matrix = spatial.distance.squareform(hamming_distance < (1.0 -
self.__theta))
        msa.sequence_weight = 1.0 / (np.sum(weight_matrix, axis=1) + 1.0)
        msa.Meff = np.sum(msa.sequence_weight)


    def sitefreq(self, msa, ICs):


        """ Calculate the single site frequencies in encoded MSA. """


        if not msa.Meff:
            self.Meff(msa)

        site_freq = np.zeros((len(ICs), msa.q), dtype=float)

        for i in range(len(ICs)):
            vec = np.bincount(msa.encoded_MSA(ICs)[:, i], weights=msa.sequence_weight)
            site_freq[i, 0:vec.size] = vec

        site_freq /= msa.Meff
        self.__site_freq = (1 - self.__lambda) * site_freq + self.__lambda / msa.q


    def pairfreq(self, msa, ICs):


        """ Calculate the double site frequencies in encoded MSA. """
```

```
        if not msa.Meff:
            self.Meff(msa)

        pairfreq = np.zeros((len(ICs), msa.q, len(ICs), msa.q),dtype=float)
        encoded = msa.encoded_MSA(ICs)

        pairfreq = np.zeros((len(ICs), msa.q, len(ICs), msa.q),dtype=float)
        for a in range(msa.num_of_seqs):
            for i in range (len(ICs)):
                for j in range(len(ICs)):
                    pairfreq[i, encoded[a, i], j, encoded[a, j]] +=
msa.sequence_weight[a]

        pairfreq /= msa.Meff
        pair_freq = (1 - self.__lambda)*pairfreq + self.__lambda/(msa.q * msa.q)

        for i, aa in enumerate(ICs):
            for am_i in range(msa.q):
                for am_j in range(msa.q):
                    if (am_i==am_j):
                        pair_freq[i,am_i,i,am_j] = self.__site_freq[i,am_i]
                    else:
                        pair_freq[i,am_i,i,am_j] = 0.0

        self.__pair_freq = pair_freq


    def MI(self, msa, docking):


        """ Calculate mutual information for a protein complex. """


        ICs = docking.A_ICs + docking.shift_B_ICs
        self.sitefreq(msa, ICs)
        self.pairfreq(msa, ICs)

        #enc = msa.encoded_MSA(ICs)
        #with open("enc_{}.py".format(docking.name), 'w') as f:
            #f.write(str(enc[0, :].tolist()))

        contact_map = docking.contact_map
        A_ICs = docking.A_ICs
        B_ICs = docking.shift_B_ICs

        nA = len(A_ICs)
        nB = len(B_ICs)

        mi_matrix_pp = np.zeros((nA, nB), dtype=float)
        h_matrix = np.zeros((nA, nB), dtype=float)

        for i, col_i in enumerate(A_ICs):
            for j, col_j in enumerate(B_ICs):
                tmp1 = (np.sum(self.__pair_freq[i, :, j + nA, :] *
                np.log(self.__pair_freq[i, :, j+nA, :] /
                (np.transpose(np.broadcast_to(self.__site_freq[i, :] , (msa.q, msa.q))) *
                np.broadcast_to(self.__site_freq [j + nA, :], (msa.q, msa.q))))))
                tmp2 = (np.sum(self.__pair_freq[i, :, j + nA, :] *
np.log(self.__pair_freq[i, :, j + nA, :]))) * -1
                if any((x == [col_i, col_j]).all() for x in contact_map):
                    mi_matrix_pp[i,j] = tmp1
```

```
                    h_matrix[i,j] = tmp2

                docking.MI = np.sum(mi_matrix_pp)
        docking.h =  np.sum(h_matrix)
        print("MI = {}, MI/h = {}".format(np.sum(mi_matrix_pp),
np.sum(mi_matrix_pp)/np.sum(h_matrix)))


    def correlation(self, msaA, msaB):

        """ Calculate the correlation value for the distance arrays of chains. """

        calculator = DistanceCalculator('blosum62')
        dm_A = calculator.get_distance(msaA)
        dm_B = calculator.get_distance(msaB)
        av_A = np.average(dm_A)
        av_B = np.average(dm_B)

        num = 0
        den_A = 0
        den_B = 0
        num_of_seqs = len(msaA)

        for i in range(num_of_seqs):
            for j in range(num_of_seqs):
                num += (dm_A[i, j] - av_A) * (dm_B[i, j] - av_B)
                den_A += (dm_A[i, j] - av_A)**2
                den_B += (dm_B[i, j] - av_B)**2
        r = num / (np.sqrt(den_A) * np.sqrt(den_B))

        print("r = {}".format(r))
        return r
```

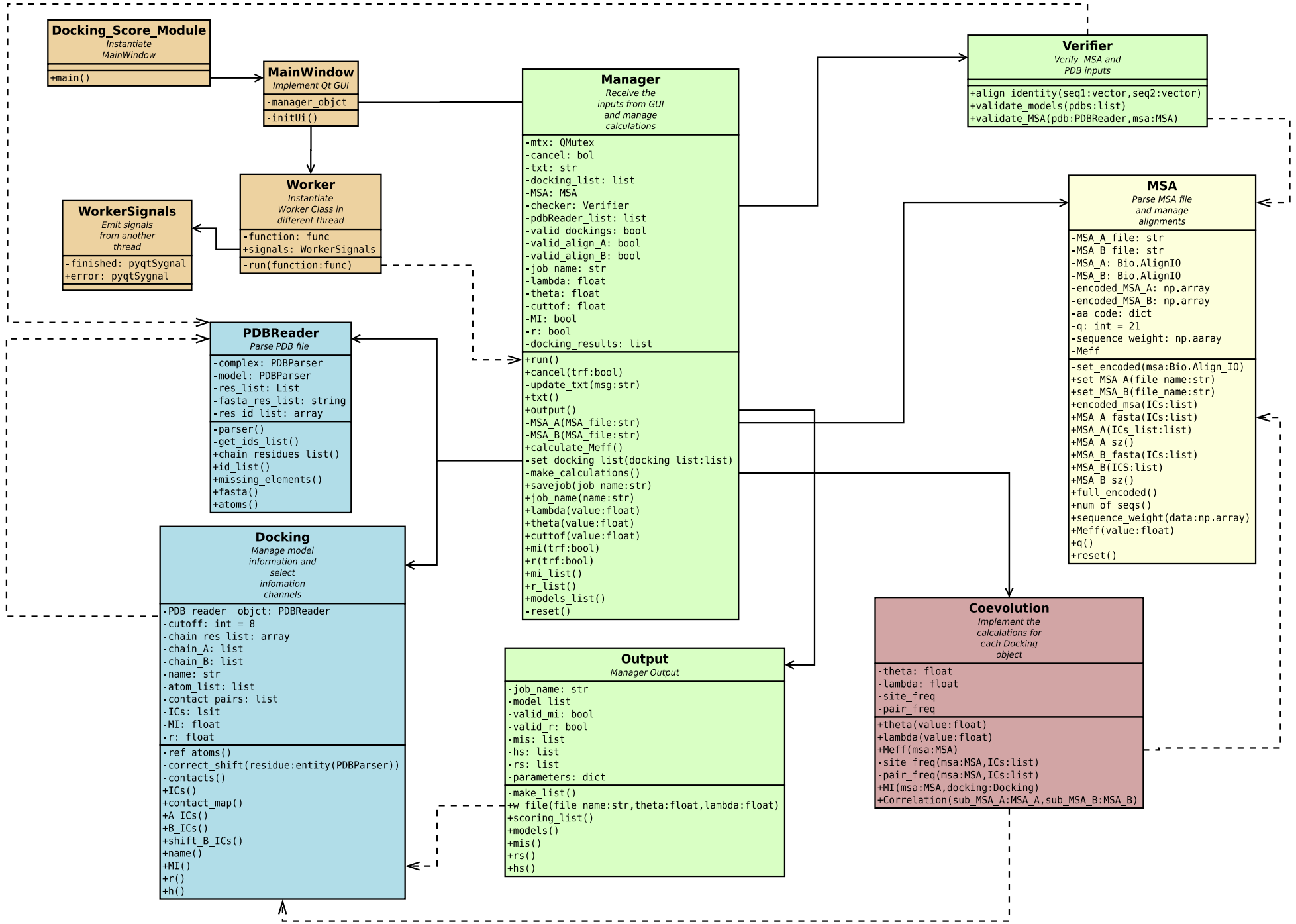

Figura 32: UML - ***Docking Score Module***