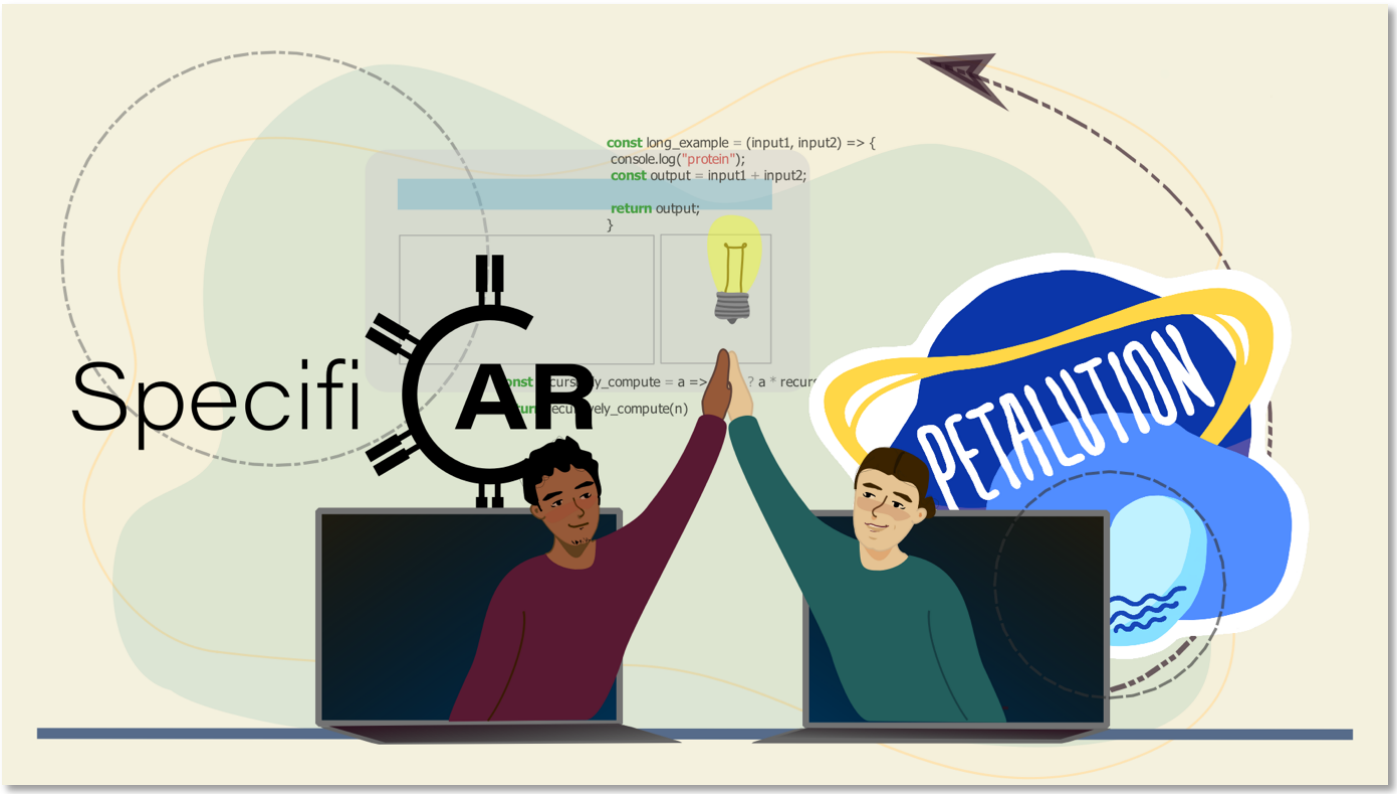
Docking Simulations Tutorial



Authors:

Edinburgh-UHAS Ghana iGEM 2022

Munich iGEM 2022

TABLE OF CONTENTS

1. FOREWORD

Dear future iGEM Team,

This tutorial provides a comprehensive guide on the various types of docking simulations available, and leads you through generating files, docking them, and analyzing data. This tutorial is written for absolute beginners who have no experience in molecular docking, coding, or the command line. We have options listed for every experience level and intention in docking. Any time you may want to see how one molecule interacts with another, you can use docking simulations to predict it. We found them very useful in our projects, and we hope that you will find them equally useful in yours. Once you get the hang of molecular docking, it is very easy to iterate it over a bunch of different protein or ligand structures as well, generating usable data for your iGEM project outside of the wet lab. That being said, go perform some docking! Go generate some data and characterize your parts in a more significant way.

Yours,

The Edinburgh-UHAS_Ghana and Munich iGEM Teams of 2022

2. Generating Receptor and Ligand structural Files

Before you begin any docking simulations, you must have structural files for your ligand and receptor. For all intents and purposes, the ligand is the small molecule which binds your receptor, or protein. There are several different file formats for structural molecules, PDB, MOL2, SMI, SDF, XYZ etc., and it can be confusing which ones you need to use. Basically, the receptor needs to be a PDB file, and the ligand can be pretty much any structural file format (see https://openbabel.org/wiki/Babel#File_Formats for a comprehensive list). However, the types of files that certain programs can take are restricted, so sometimes they need to be converted. This can be easily done with the command line tool OpenBabel.

To convert from one file type to another, use the command

```
babel [OPTIONS] [-i input-type] infile [-o output-type] outfile
```

where input- and output types come from the link mentioned above. For example, if I had a file ligand.sdf and wanted to convert it to ligand.mol2, I would type the command

```
babel -isdf ligand.sdf -omol2 ligand.mol2
```

And that would produce a file in the .mol2 format. The type of file format you need to produce for your ligand depends on the docking software you are using.

However, we must first generate the initial ligand files before converting them between various formats. https://www.rcsb.org/downloads/ligands is a tool by the PDB for downloading ligands, based off their IDs. If you know a ligand's ID (check the ligand tab on proteins you know bind this ligand), you can easily download a .sdf, .mol2 or .ccd file with this. If you cannot find a ligand's ID, it may be easier to generate it from scratch. SMILES is a chemical notation that is used to easily represent chemical structures on a computer, and you can download files based off their SMILES. https://www.novoprolabs.com/tools/smiles2pdb is an online tool which lets you download a .pdb, .mol2 or .sdf based off of a SMILES. This is especially useful if you are using novel ligands, or ligands with inorganic atom types, such as heavy metals. To see more about how SMILES works, visit https://en.wikipedia.org/wiki/Simplified_molecular-input_line-entry_system. A more convenient tool for inorganic ligands is the ZINC database (https://zinc.docking.org/), where you can browse through ligand files, or generate them from an organic chemistry drawing tool under the tab Substances. Molecules can be downloaded as .db2 or .mol2 format using this tool.

To generate the receptor PDB file, there are less options. Either you have to use a structure which has been generated using crystallographic techniques, or you have to generate a novel structure using bioinformatics. The prior is quite simple, you search the PDB (at https://www.rcsb.org/) for the name of your protein, and then download in PDB format using the download files button (See figure 1).

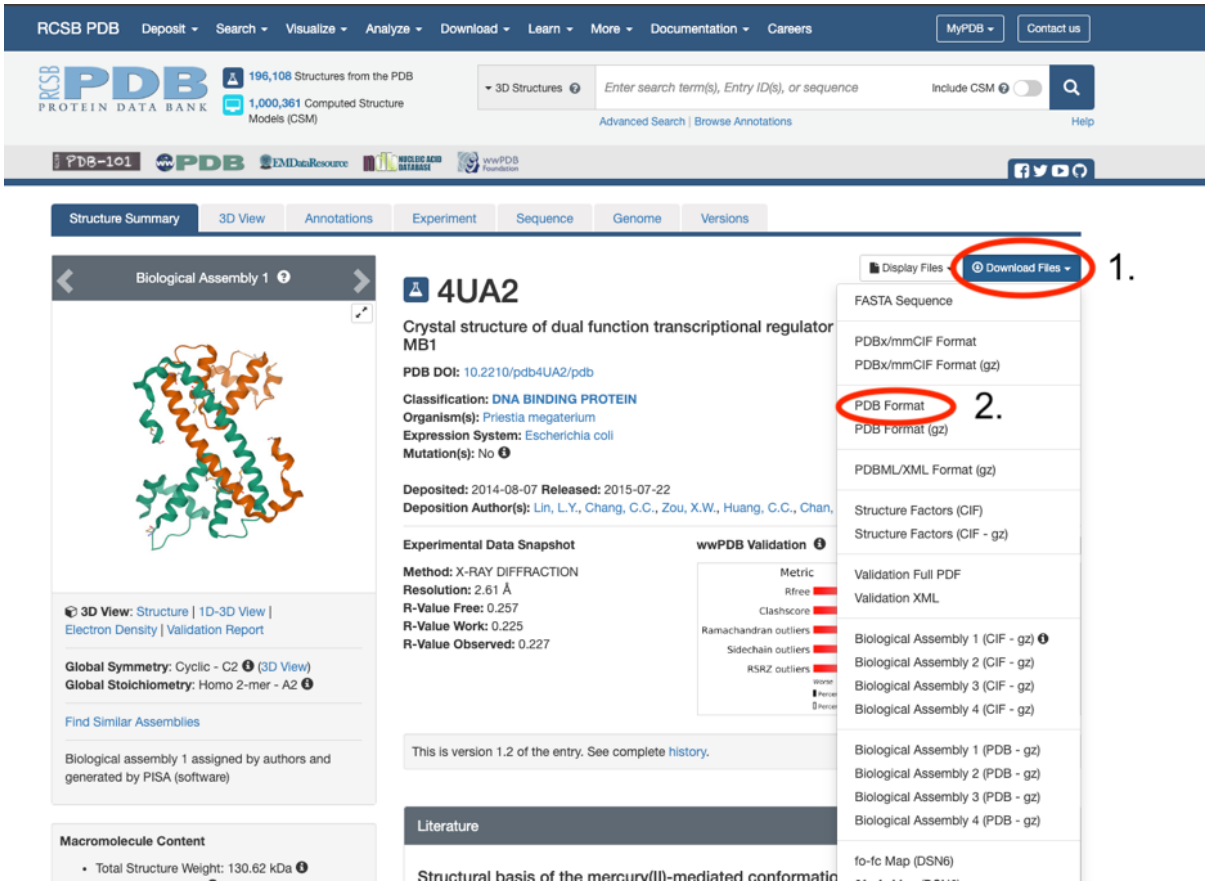


Figure 1: A screenshot of the protein merR on the PDB. 1. Indicates the first button to click, and 2. Indicates the second.

However, sometimes your protein will not exist on the PDB, as nobody has found the structure of it yet. In that case, you must result to artificial intelligence to predict what your protein structure is likely to look like. AlphaFold is an artificial intelligence program for predicting such structures. There are a few different ways to use AlphaFold, so we will cover the three main ways.

1. Look through the AlphaFold Protein Structure Database

AlphaFold has predicted the structures of every protein on UniProt, another protein database, but for sequences. This can be accessed here (https://alphafold.ebi.ac.uk/) and will likely have the protein you are looking for, if it is not entirely novel. Just click on the entry you want, and download the PDB file using the button that appears.

2. Predict the structure using a Google collab notebook

If your protein is neither on the PDB nor on UniProt, it is likely you will have to generate a novel structure. AlphaFold has made this quite easy, with a Google Collab notebook, where you simply have

to insert your sequence, and it will generate and download a structure for you. All of this is run on Google servers, and is very convenient for generating one or two structures. Open this link https://colab.research.google.com/github/deepmind/alphafold/blob/main/notebooks/AlphaFold.ipynb, and follow the instructions listed there.

3. AlphaFold GPU

If you want to generate many novel protein structures, you might need to use AlphaFold GPU. However, you will need to be using a machine running Linux, such as a cluster computer, and be very familiar with Unix. If this is something you would like to do, more instructions are available here (https://github.com/deepmind/alphafold), but if you are trying to use AlphaFold GPU you likely have no use for this tutorial anyway.

Now you will have generated your receptor and ligand structural files, and can proceed with preparing them for docking.

3. Deciding on a docking software

There are two main types of docking software: command line-based and web-based. Command line-based docking software is very powerful, allows for a lot of flexibility and customization, lets you iterate over many different files, but requires knowledge of unix. Web-based docking software is less flexible, is usually not automatable, but requires no knowledge of unix and is generally quite convenient. We will cover both approaches and their positives and negatives.

Web-based docking software

Web-based docking software is very convenient, as it simply requires you to upload files for your protein and ligand, and sometimes even lets you just input sequences. HDOCK (http://hdock.phys.hust.edu.cn/) is a particularly convenient web-based docking software, that allows many different input types. Receptors can be uploaded as a .pdb file, inputted as a .pdb file from PDB ID, or just inputted as a FASTA sequence. Ligand files can also be uploaded as such, and the sequence feature also covers single and double stranded DNA and RNA.

SwissDock (http://www.swissdock.ch/docking) is another web-based docking tool, with a command-line counterpart. The receptor can be a PDB code, protein name, sequence URL or .pdb file, and the ligand can be any type of file, a ZINC accession number, or even a ligand name, category or URL.

There are many other similar tools available online, but these two are probably the most convenient and cover most things one might need docking for.

Command line-based software

Command line-based software is much more powerful for the reasons mentioned before, but therefore also more difficult to use. The two main docking softwares we will cover are AutoDock and Rosetta, two very complete toolkits for various docking applications.

AutoDock is the industry standard for docking, with resources including graphical interfaces for docking, accelerated affinity map generation, covalent docking, and docking using additional atom types. If you need to use these sorts of techniques, AutoDock is probably a good fit. AutoDock has multiple different docking algorithms as well, including AutoDock4.2, AutoDock Vina and AutoDock GPU. AutoDock4.2 is outdated by today's standards, but is still needed for some features like covalent docking. AutoDock Vina is the replacement for AutoDock4.2, and is a fast and reliable docking software. If you are performing hundreds of dockings, it may be beneficial to use AutoDock GPU, which is a GPU accelerated version of AutoDock4.2. However, this requires an NVIDIA graphics card, as CUDA is a dependency.

The Rosetta software suite also has many excellent programs for docking, as well as beyond. Most of the software suite is for modelling various structures, including whole proteins, antibodies, carbohydrates and nucleic acid-protein interactions. However, there are a whole host of docking softwares. RosettaLigand is for normal protein-small molecule docking, with options for flexible peptide

docking and protein protein docking. Additionally, antibody (including camelid antibody) docking is available, as well as terpene synthase specific docking and Mg2+ docking. Rosetta has less functionality than AutoDock for docking, but for very specific tasks like terpene synthases, it could be beneficial.

4. Performing Docking

The docking protocol itself varies greatly from software to software, so this part of the tutorial will chiefly cover AutoDock on the command line. Web based servers are very simple and explained on the web page, and how to use other command line based software can be found in the documentation of those softwares.

For AutoDock, there are several steps to dock molecules. First, ligand and receptor molecules need to be prepared, parameter files generated, grid maps calculated, and then docking itself. This tutorial assumes that the AutoDock suite has been installed, downloads and installation instructions can be found here for AutoDock Vina (https://autodock-vina.readthedocs.io/en/latest/installation.html), here for AutoDock4.2 (https://autodock.scripps.edu/download-autodock4/), here for AutoDock GPU (https://github.com/ccsb-scripps/AutoDock-GPU), and here for AutoDockTools/MGLTools (https://ccsb.scripps.edu/mgltools/downloads/).

First, you must prepare the ligand and receptor molecules. You will do this with AutoDockTools' programs *prepare_receptor4.py* and *prepare_ligand4.py*. Usage follows the format

```
python prepare_receptor4.py -r receptor.pdb [-A Hydrogens] [-o output.pdbqt]
```

```
python prepare_ligand4.py -l ligand.pdb [-o output.pdbqt]
```

The `-A Hydrogens` flag can be specified if you need to add hydrogens into the molecule. The input file for *prepare_ligand4.py* can be a .sdf, a .mol2 or a .pdb. Both of these programs will produce .pdbqt files, which are similar to .pdb files but have gasteiger charges added for all atoms, and the name can be specified with the -o flag. Otherwise, the output name is the same as the input name.

This is where you can prepare a flexible side chain receptor if you want. This allows a degree of flexibility for the binding site on the receptor. These files are generated with AutoDockTools' *prepare_flexreceptor4.py.* Usage follows the format

```
python prepare_flexreceptor4.py -r receptor.pdbqt -s receptor:chain ID:resname
```

Where the -s argument specifies which residues to make flexible, and can take a comma separated, residue underscored list in the format receptor:chain ID:resname, for example

```
hsg1:A:ARG8_ILE84,hsg1:B:THR4
```

Specifies to make residues ARG8 and ILE84 of Chain A, as well as THR4 of chain B flexible in the protein hsg1. This will produce rigid and flexible outputs for each residue specified.

The next step is to prepare grid parameter files (GPFs) and docking parameter files (DPFs), which uses AutoDockTools' *prepare_gpf4.py* and *prepare_dpf4.py* respectively. It follows the format

```
python prepare_gpf4.py -r receptor.pdbqt -l ligand.pdbqt [-x flexres.pdbqt]
[-l gridparameters.gpf]
```

```
python prepare_dpf4.py -r receptor.pdbqt -l ligand.pdbqt [-x flexres.pdbqt]
[-l dockingparameters.dpf]
```

The -x flag can be specified if you choose to do flexible docking, in that case use the flexible file generated on the -x flag and the rigid file on the -r flag. These parameter files are required for AutoDock and AutoGrid to function, which is also the next step. To run these use

```
autogrid4 -p gridparameters.gpf -l gridlog.glg
```

```
autodock4 -p dockingparameters.dpf -l dockinglog.dlg
```

It is very important that it goes in this order, as autogrid grid calculations accelerate autodock. Now you have your docking log, and docking has been done! However, you may have been expecting to get a structure that you can open with pymol, or some kinetic parameters regarding how well something docks. Those still need to be extracted from the docking log.

5. Results analysis

Docking log files are long, complex and have a lot of information. But they all follow the same basic structure:

1. AutoDock attribution
2. Input parameters from the .gpf and .dpf files
3. 10 best docks
4. Clustering analysis and histogram data
5. RMSD table
6. Statistical Mechanical Analysis
7. Lowest energy docked conformations from each cluster

The data you want to use is always the lowest energy conformation, as lower gibbs free energy means something is more likely to happen spontaneously. The coordinates listed in the lowest energy docked conformations from each cluster section can be copied and pasted into your original pdb file for the receptor, to show the docked structure (see figure 2).

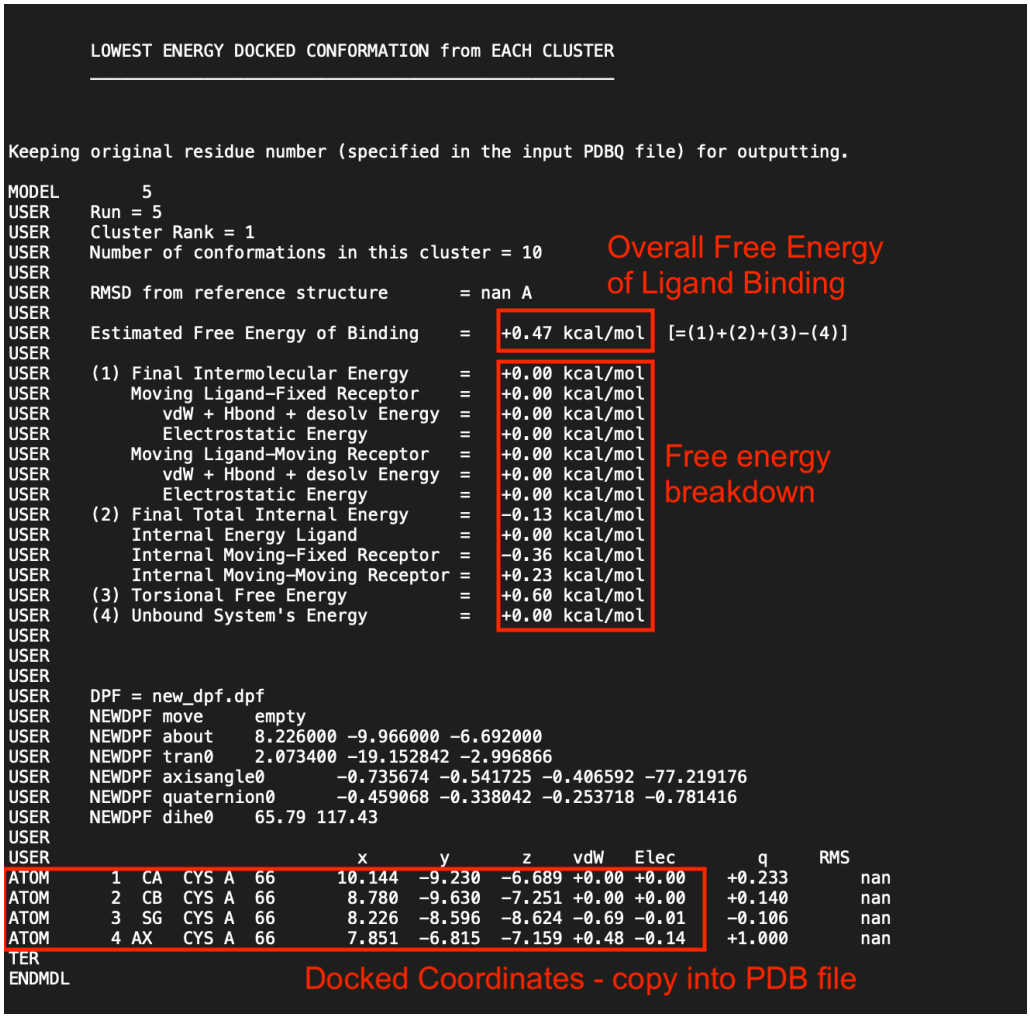


Figure 2: Breakdown of the lowest energy docked conformation of each cluster

One additional step after this would be energy minimization, if you are interested in protein secondary structure changes. This can be done with a command line tool from openbabel, obminimize, where the general usage follows

```
Obminimize [-ff ffid] filename.pdb
```

Where the -ff flag specifies which forcefield algorithm to use. Ghemical and GAFF are two of the most common types. Alternatively, you can use the YASARA energy minimization server (http://www.yasara.org/minimizationserver.htm), but for some ligands such as DNA, this will not work.

6. Protein Mutagenesis

If you want to optimize the binding of a protein to a ligand, you might consider mutating certain residues in the protein and subsequently assessing the effect the mutation has on the binding affinity. A convenient way to achieve this is by using the mutagenesis tool provided by PyMol. You can use the GUI based version of PyMol to do so, but if you want to try out many mutations, it will save you a lot of time to use the PyMol API. The best way to use the PyMol API is by writing a simple python script.

I recommend creating an Anaconda environment as this will simplify the package management. First, you need to install pymol in your conda environment (https://www.pymol.org/conda/).

Mutating an amino acid in a protein using the PyMol API

Start your python script by importing the command tool of the PyMol API:

```
from pymol import cmd
```

Next, load the pdb file of your protein:

```
cmd.load('my_protein.pdb')
```

Then, select the mutagenesis wizard and refresh it – this is especially important if you want to mutate several amino acids.

```
cmd.wizard("mutagenesis")
cmd.do("refresh_wizard")
```

In the next section "Determining feasible mutations", we describe a feasible way to determine mutations you might want to consider in order to have an unbiased approach for a protein mutagenesis. But for now, we will focus on mutating one amino acid into a known other one. Let's say you want change an alanine (ALA) at position 42 in chain A of your protein into a proline (PRO). For this, set the wizard mode to the amino acid you want to introduce to your sequence, in our case 'PRO'.

```
cmd.get_wizard().set_mode('PRO')
```

Subsequently, we select the residue we want to mutate using the command structure "/<protein_name>//<chain>/<position>". The protein name is the name of your pdb file without the .pdb extension. In our example, this would result in the following code:

```
cmd.get_wizard().do_select("/my_protein//A/42")
```

What's left? Well, let's actually perform the mutation!

```
cmd.get_wizard().apply()
```

If you want to mutate several residues of the protein, just repeat the last five steps until you're done.

Great, now we have a mutated protein. Let's save it to a pdb file:

```
pdb = cmd.get_pdbstr("my_protein")
f = open("mutated_protein.pdb", "w")
f.write(pdb)
f.close()
```

If you want to mutate several proteins within one python scripts, reinitialize the PyMol command tool:

```
cmd.set_wizard("done")
cmd.reinitialize('everything')
```

That's it – now you have your mutated protein, saved as "mutated_protein.pdb".

Determining feasible mutations

You're now able to mutate a protein. One thing you might be wondering is how to decide which mutations to introduce in your protein? Let's say you have a protein structure and want to optimize the binding affinity to a DNA motif. The first thing you need to do is to determine which residues actually bind to the DNA, because especially for larger proteins, it would needlessly consume computational power if you mutated residues that do not influence the DNA docking.

If you don't know which residues of your protein are the DNA binding ones, I recommend using PredictProtein (https://predictprotein.org/), a web based tool that will predict which residues in your protein sequence are most likely to be DNA binding ones.

Assuming you have identified the residues influencing the binding affinity, you can now think about a way to determine with which amino acids your want to substitute the residues in your original protein sequence. If your docking simulation isn't computationally expensive, you can consider just trying out all other 19 amino acids. In most cases however, this isn't feasible.

In that case, you might want to restrict your analysis to conservative mutations. Conservative mutations maintain the original conformation of proteins by substituting an amino acid with another with similar physicochemical properties (Miyata, Miyazawa and Yasunaga, 1979; Zhang, 2000). However, there exist several classifications of amino acids into groups with similar physicochemical properties.

According to Miyata, Miyazawa and Yasunaga, 1979; Zhang, 2000, one possible classification is:

- Special: Cys
- Neutral, small: Ala, Gly, Pro, Ser, Thr
- Hydrophilic, small: Asn, Asp, Gln, Glu
- Hydrophilic, large: Arg, His, Lys
- Hydrophobic, small: Ile, Leu, Met, Val
- Hydrophobic, large: Phe, Trp, Tyr

Each amino acid you aim to mutate can then be replaced with each amino acid in the same classification group once, while not introducing any other mutations into the protein. For instance, a threonine, which is part of the “neutral, small” class would be substituted by Ala, Gly, Pro and Ser. So, you would end up with four mutated proteins in order to conservatively mutate the threonine. For each of these mutated proteins, you then need to perform a docking simulation and assess the respective effect of the mutation on the binding affinity by comparing it to the baseline (unmutated) binding affinity.

References

Miyata, T., Miyazawa, S. and Yasunaga, T. (1979) ‘Two types of amino acid substitutions in protein evolution’, Journal of Molecular Evolution, 12(3), pp. 219–236. Available at: https://doi.org/10.1007/BF01732340.

Zhang, J. (2000) ‘Rates of Conservative and Radical Nonsynonymous Nucleotide Substitutions in Mammalian Nuclear Genes’, Journal of Molecular Evolution, 50(1), pp. 56–68. Available at: https://doi.org/10.1007/s002399910007.