GPA∗

1.0.0

Generated by Doxygen 1.8.16

1 Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 Namespace Documentation

3.1 compare_db_perf_new_format Namespace Reference

Functions

- def main ()
- def gen_all (list filenames_db, list legend_names,str common_path)

  Takes the tasks and processes them either one by one or in parallel.
- def best_traj (int fig_num, list filenames_db, list legend_names,str guide_metr,str common_path)

  This is just a basic comparison among metrics.
- int plot_all_best_traj (int fig_num, list cur_arr, list filenames_db, list legend_names,str guide_metr,str common_path)
- def plot_sep_best_traj (fig_num, cur_arr, filenames_db, legend_names, guide_metr, common_path)
- int guide_metr_usage (int fig_num, list filenames_db, list legend_names,str guide_metr,str common_path)
- int plot_all_metrics (int fig_num, list cur_arr, list filenames_db, list legend_names,str guide_metr,str common_path)

  General force field comparison: sampling, best_so_far, dist traveled.
- int plot_only_one_metric (int fig_num, list cur_arr, list filenames_db, float init_rmsd, list legend_names,str metric_name,str guide_metr,str common_path)
- int plot_set (int fig_num, list to_goal_arr, list legend_names, float max_len, float max_non_init_rmsd, float init_metr, list bsf_arr, float common_point, float max_trav, list trav_arr,str full_cut,str metric,str metr_units,str same,str custom_path, bool shrink, list non_shrink_arr=None)
- int single_plot (int fig_num,dict ax_prop, list arr_A, list arr_B, list filenames_db,str marker, float mark_size, bool bsf, bool rev, bool shrink,str xlab,str ylab,str title,str filename, list extra_line=None, int mdpi=400,dict second_ax=None, list sec_arr=None)

  Main plotting function.

3.1.1 Function Documentation

3.1.1.1 best_traj() def compare_db_perf_new_format.best_traj (int fig_num, list filenames_db, list legend_names, str guide_metr, str common_path)

This is just a basic comparison among metrics.

```
list  fig_num: figure number for matplotlib
list  filenames_db: databases with data
list  legend_names: database names
str  guide_metr:
str  common_path:
```

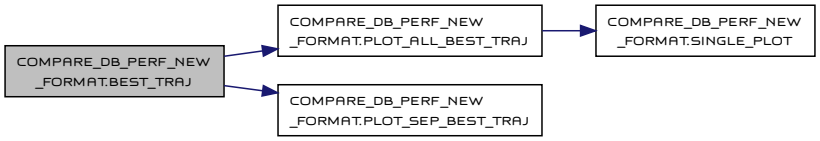
Definition at line 114 of file compare_db_perf_new_format.py.

```
00114     """
00115     print('Working with ', filenames_db, ' guide metr: ', guide_metr, ' common path: ', common_path)
00116     con_arr = [lite.connect(db_name, check_same_thread=False, isolation_level=None) for db_name in filenames_db]
00117     cur_arr = [con.cursor() for con in con_arr]
00118 
00119     common_path = os.path.join(common_path, guide_metr)
00120     try:
00121         os.mkdir(common_path)
00122     except:
00123         pass
00124     plot_all_best_traj(fig_num, cur_arr, filenames_db, legend_names, guide_metr, common_path)
00125     plot_sep_best_traj(fig_num, cur_arr, filenames_db, legend_names, guide_metr, common_path)
00126 
00127 
00128 def plot_all_best_traj(fig_num: int, cur_arr: list, filenames_db: list, legend_names: list, guide_metr: str, common_path: str) -> int:
00129     """
00130 
00131     Args:
00132          int fig_num:
00133          list cur_arr:
00134          list filenames_db:
00135          list legend_names:
00136          str  guide_metr:
00137          str  common_path:
00138 
00139     Returns:
00140         :return: figure number
00141         return type: int
```

References plot_all_best_traj(), and plot_sep_best_traj().

Referenced by gen_all().

Here is the call graph for this function:



Here is the caller graph for this function:

3.1.1.2 gen_all()

```
def compare_db_perf_new_format.gen_all (
        list  filenames_db,
        list  legend_names,
        str  common_path )
```

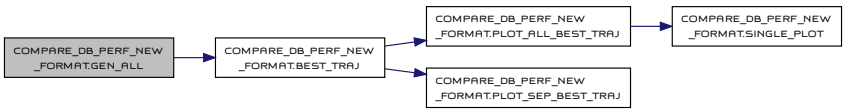
Takes the tasks and processes them either one by one or in parallel.

```
list  filenames_db:   list  of databases
list  legend_names: correct names for DBs
str  common_path: where to store plots
```

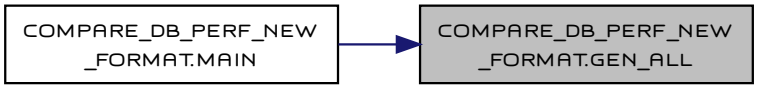
Definition at line 75 of file compare_db_perf_new_format.py.

```
00075     """
00076     fig_num = 0
00077     try:
00078         os.mkdir(common_path)
00079     except:
00080         pass
00081     # mdpi = 400
00082     #
00083     # font = {'family': 'serif',
00084     #         'color':  'darkred',
00085     #         'weight': 'normal',
00086     #         'size': 12,
00087     #         }
00088     parallel = True  # both work, use parallel to generate everything fast, use debug otherwise
00089     if parallel:
00090         pool = mp.Pool(len(['rmsd', 'angl', 'andh', 'and', 'xor']))  # we are IO bound in graphs, no need to use exact number of CPUs
       mp.cpu_count()
00091         results1 = pool.starmap_async(guide_metr_usage, [(fig_num, filenames_db, legend_names, guide_metr, common_path) for guide_metr in
       ['rmsd', 'angl', 'andh', 'and', 'xor']])
00092         results2 = pool.starmap_async(best_traj, [(fig_num, filenames_db, legend_names, guide_metr, common_path) for guide_metr in ['rmsd',
       'angl', 'andh', 'and', 'xor']])
00093         results1.get()
00094         results2.get()
00095         pool.close()
00096     else:  # then debug
00097         # for guide_metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00098         #     fig_num = guide_metr_usage(fig_num, filenames_db, legend_names, guide_metr, common_path)
00099
00100         for guide_metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00101             best_traj(fig_num, filenames_db, legend_names, guide_metr, common_path)
00102
00103
```

References best_traj().
Referenced by main().
Here is the call graph for this function:



Here is the caller graph for this function:



3.1.1.3 guide_metr_usage()

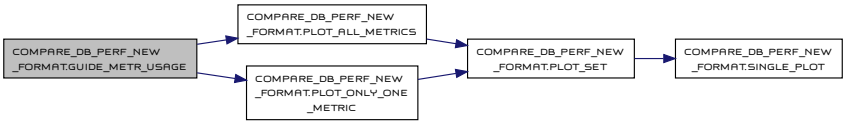
```
int compare_db_perf_new_format.guide_metr_usage (
        int  fig_num,
        list  filenames_db,
        list  legend_names,
        str  guide_metr,
        str  common_path )
```

Definition at line 482 of file compare_db_perf_new_format.py.

```
00482     """
00483
00484     con_arr = [lite.connect(db_name, check_same_thread=False, isolation_level=None) for db_name in filenames_db]
00485     cur_arr = [con.cursor() for con in con_arr]
00486
00487     common_path = os.path.join(common_path, guide_metr)
00488     try:
00489         os.mkdir(common_path)
00490     except:
00491         pass
00492
00493     fig_num, init_rmsd = plot_all_metrics(fig_num, cur_arr, filenames_db, legend_names, guide_metr, common_path)
00494
00495     for partial_metr in ["RMSD", "ANGL", "AND_H", "AND", "XOR"]:
00496         pers_path = os.path.join(common_path, partial_metr)
00497         try:
00498             os.mkdir(pers_path)
00499         except:
00500             pass
00501         fig_num = plot_only_one_metric(fig_num, cur_arr, filenames_db, init_rmsd, legend_names, partial_metr, guide_metr, pers_path)
00502
00503     [con.close() for con in con_arr]
00504     return fig_num
00505
00506
```

References plot_all_metrics(), and plot_only_one_metric().

Here is the call graph for this function:



3.1.1.4 main()

```
def compare_db_perf_new_format.main ( )
```

Definition at line 17 of file compare_db_perf_new_format.py.

```
00017     """
00018     batch_arr = list()
00019     ffs = ['amber', 'charm', 'gromos', 'opls']
00020     #########   TRP   ######################
00021     # for ff in ffs:
00022     #     filenames_db = ['results_{}_trp_300_fixed.sqlite3'.format(ff), 'results_{}_trp_300_2_fixed.sqlite3'.format(ff)]
00023     #     legend_names = ['TRP {}_1'.format(ff), 'TRP {}_2'.format(ff)]
00024     #     common_path = '../trp_{}_compar'.format(ff)
00025     #     batch_arr.append((filenames_db, legend_names, common_path))
```

```
00026
00027     filenames_db = ['results_amber_trp_300_2_fixed.sqlite3', 'results_charm_trp_300_2_fixed.sqlite3', 'results_gromos_trp_300_2_fixed.sqlite3',
      'results_opls_trp_300_2_fixed.sqlite3']
00028     # legend_names = ['TRP amber_2', 'TRP charm_2', 'TRP gromos_2', 'TRP opls_2']
00029     legend_names = ['1L2Y, 2nd run with AMBER ff', '1L2Y, 2nd run with CHARM ff', '1L2Y, 2nd run with GROMOS ff', '1L2Y, 2nd run with OPLS ff']
00030     common_path = '../trp_all_2_compar'
00031     batch_arr.append((filenames_db, legend_names, common_path))
00032
00033     filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_charm_trp_300_fixed.sqlite3', 'results_gromos_trp_300_fixed.sqlite3',
      'results_opls_trp_300_fixed.sqlite3']
00034     # legend_names = ['TRP amber_1', 'TRP charm_1', 'TRP gromos_1', 'TRP opls_1']
00035     legend_names = ['1L2Y, 1st run with AMBER ff', '1L2Y, 1st run with CHARM ff', '1L2Y, 1st run with GROMOS ff', '1L2Y, 1st run with OPLS ff']
00036     common_path = '../trp_all_1_compar'
00037     batch_arr.append((filenames_db, legend_names, common_path))
00038
00039     filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_amber_trp_300_2_fixed.sqlite3', 'results_charm_trp_300_fixed.sqlite3',
      'results_charm_trp_300_2_fixed.sqlite3', 'results_gromos_trp_300_fixed.sqlite3', 'results_gromos_trp_300_2_fixed.sqlite3',
      'results_opls_trp_300_fixed.sqlite3', 'results_opls_trp_300_2_fixed.sqlite3']
00040     legend_names = ['1L2Y, 1st run with AMBER ff', '1L2Y, 2nd run with AMBER ff', '1L2Y, 1st run with CHARM ff', '1L2Y, 2nd run with CHARM ff',
      '1L2Y, 1st run with GROMOS ff', '1L2Y, 2nd run with GROMOS ff', '1L2Y, 1st run with OPLS ff', '1L2Y, 2nd run with OPLS ff']
00041     # legend_names = ['TRP amber_1', 'TRP amber_2', 'TRP charm_1', 'TRP charm_2', 'TRP gromos_1', 'TRP gromos_2', 'TRP opls_1', 'TRP opls_2']
00042     legend_names = ['1L2Y, 1st run with AMBER ff', '1L2Y, 2nd run with AMBER ff', '1L2Y, 1st run with CHARM ff', '1L2Y, 2nd run with CHARM ff',
      '1L2Y, 1st run with GROMOS ff', '1L2Y, 2nd run with GROMOS ff', '1L2Y, 1st run with OPLS ff', '1L2Y, 2nd run with OPLS ff']
00043     common_path = '../trp_all_compar'
00044     batch_arr.append((filenames_db, legend_names, common_path))
00045
00046     # # ##################  VIL  ######################
00047
00048     filenames_db = ['results_amber_vil_300.sqlite3', 'results_charm_vil_300.sqlite3', 'results_gromos_vil_300.sqlite3',
      'results_opls_vil_300.sqlite3']
00049     # legend_names = ['VIL amber', 'VIL charm', 'VIL gromos', 'VIL opls']
00050     legend_names = ['1YRF with AMBER ff', '1YRF with CHARM ff', '1YRF with GROMOS ff', '1YRF with OPLS ff']
00051     common_path = '../vil_all_compar'
00052     batch_arr.append((filenames_db, legend_names, common_path))
00053
00054     # # ##################  GB1  ######################
00055     # #
00056     filenames_db = ['results_amber_gb1_300.sqlite3', 'results_charm_gb1_300.sqlite3', 'results_gromos_gb1_300.sqlite3',
      'results_opls_gb1_300.sqlite3']
00057     # legend_names = ['GB1 amber', 'GB1 charm', 'GB1 gromos', 'GB1 opls']
00058     legend_names = ['1GB1 with AMBER ff', '1GB1 with CHARM ff', '1GB1 with GROMOS ff', '1GB1 with OPLS ff']
00059     common_path = '../gb1_all_compar'
00060     batch_arr.append((filenames_db, legend_names, common_path))
00061
00062
00063     for filenames_db, legend_names, common_path in batch_arr:
00064         gen_all(filenames_db, legend_names, common_path)
00065
00066
00067
```

References gen_all().

Here is the call graph for this function:

3.1.1.5 plot_all_best_traj()

```
int compare_db_perf_new_format.plot_all_best_traj (
        int  fig_num,
        list  cur_arr,
        list  filenames_db,
        list  legend_names,
        str  guide_metr,
        str  common_path )
```

Definition at line 142 of file compare_db_perf_new_format.py.

```
00142     """
00143     print('Working with ', filenames_db, ' guide metr: ', guide_metr, ' common path: ', common_path)
00144     qry = "select a.name from main_storage a where a.{0}_goal_dist= ( select min(b.{0}_goal_dist) from main_storage b)".format(guide_metr)
00145     result_arr = [cur.execute(qry) for cur in cur_arr]
00146     fetched_one_arr = [res.fetchone() for res in result_arr]
00147     names = [all_res[0] for all_res in fetched_one_arr]
00148     spnames = [name.split('_') for name in names]
```

```
00149     all_prev_names_s = [['\'{}\"'.format('_'.join(spname[:i])) for i in range(1, len(spname)+1)] for spname in spnames]
00150     long_lines = ['', ''.join(all_prev_names) for all_prev_names in all_prev_names_s]
00151     qrys = ["select a.rmsd_goal_dist, a.angl_goal_dist, a.andh_goal_dist, a.and_goal_dist, a.xor_goal_dist, a.rmsd_tot_dist, a.angl_tot_dist,
      a.andh_tot_dist, a.and_tot_dist, a.xor_tot_dist, a.name, a.hashed_name from main_storage a where a.name in ( {1} ) order by
      a.id".format(guide_metr, long_line) for long_line in long_lines]
00152     result_arr = list()
00153     for i, cur in enumerate(cur_arr):
00154         result_arr.append(cur.execute(qrys[i]))
00155     fetched_all_arr = [res.fetchall() for res in result_arr]
00156 
00157     rmsd_dist_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00158     angl_dist_arr = [[dist[1] for dist in goal_dist] for goal_dist in fetched_all_arr]
00159     andh_dist_arr = [[dist[2] for dist in goal_dist] for goal_dist in fetched_all_arr]
00160     and_dist_arr = [[dist[3] for dist in goal_dist] for goal_dist in fetched_all_arr]
00161     xor_dist_arr = [[dist[4] for dist in goal_dist] for goal_dist in fetched_all_arr]
00162 
00163 
00164     rmsd_tot_dist_arr = [[dist[5] for dist in goal_dist] for goal_dist in fetched_all_arr]
00165     angl_tot_dist_arr = [[dist[6] for dist in goal_dist] for goal_dist in fetched_all_arr]
00166     andh_tot_dist_arr = [[dist[7] for dist in goal_dist] for goal_dist in fetched_all_arr]
00167     and_tot_dist_arr = [[dist[8] for dist in goal_dist] for goal_dist in fetched_all_arr]
00168     xor_tot_dist_arr = [[dist[9] for dist in goal_dist] for goal_dist in fetched_all_arr]
00169 
00170     goal_dist = [rmsd_dist_arr, angl_dist_arr, andh_dist_arr, and_dist_arr, xor_dist_arr]
00171     tot_dist = [rmsd_tot_dist_arr, angl_tot_dist_arr, andh_tot_dist_arr, and_tot_dist_arr, xor_tot_dist_arr]
00172     metrics = ['rmsd', 'angl', 'andh', 'and', 'xor']
00173     metr_units = {'rmsd': 'Å', 'angl': '', 'andh': 'contacts', 'and': 'contacts', 'xor': 'contacts'}
00174 
00175 
00176 
00177     for i, dist_arr in enumerate(goal_dist):  # iterate over metric
00178         max_len = max([len(arr) for arr in dist_arr])
00179         max_pos_metr_val = max([max(arr) for arr in dist_arr])
00180         init_metr = dist_arr[0][0]
00181 
00182         ax_prop = {"min_lim_x": 0 - max_len / 80, "max_lim_x": max_len + max_len / 80, "min_lim_y": 0 - max_pos_metr_val / 80, "max_lim_y":
      max_pos_metr_val + max_pos_metr_val / 80, "min_ax_x": 0,
00183                    "max_ax_x": max_len + max_len / 80, "min_ax_y": 0, "max_ax_y": max_pos_metr_val + max_pos_metr_val / 80, "ax_step_x":
      math.floor(max_len / 16), "ax_step_y": max_pos_metr_val / 20}
00184         if metr_units[metrics[i]] == 'contacts':
00185             extra_line = [{"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({} {})".format(metrics[i].upper(), int(init_metr),
      metr_units[metrics[i]]), "col": "darkmagenta"}]
00186         else:
00187             extra_line = [{"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metrics[i].upper(), init_metr,
      metr_units[metrics[i]]), "col": "darkmagenta"}]
00188         if metrics[i] == 'rmsd':
00189             extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units[metrics[i]]), "col":
      "midnightblue"})
00190         title = "{} version of the best trajectory | {} view".format(guide_metr, metrics[i])
00191         filename = "{}_version_of_best_traj_{}".format(guide_metr, metrics[i])
00192         filename = os.path.join(common_path, filename)
00193         fig_num = single_plot(fig_num, ax_prop, dist_arr, None, legend_names.copy(), '-', 1, bsf=False, rev=False, extra_line=extra_line,
      shrink=True, xlab="Steps (20ps each)", ylab="Distance to the goal, {}".format(metr_units[metrics[i]]), title=title, filename=filename)
00194 
00195         max_tot_dist = max([dist[-1] for dist in tot_dist[i]])
00196         ax_prop = {"min_lim_x": max_pos_metr_val + max_pos_metr_val / 80, "max_lim_x": 0 - max_pos_metr_val / 80, "min_lim_y": 0 - max_tot_dist
      / 80, "max_lim_y": max_tot_dist + max_tot_dist / 80, "min_ax_x": 0, "max_ax_x": max_pos_metr_val + max_pos_metr_val / 80, "min_ax_y": 0,
      "max_ax_y": max_tot_dist + max_tot_dist / 80, "ax_step_x": max_pos_metr_val / 20, "ax_step_y": max_tot_dist / 20}
00197         if metr_units[metrics[i]] == 'contacts':
00198             extra_line = [{"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({} {})".format(metrics[i].upper(), int(init_metr),
      metr_units[metrics[i]]), "col": "darkmagenta"}]
00199         else:
00200             extra_line = [{"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metrics[i].upper(), init_metr,
      metr_units[metrics[i]]), "col": "darkmagenta"}]
00201         if metrics[i] == 'rmsd':
00202             extra_line.append({"ax_type": 'ver', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units[metrics[i]]), "col":
      "midnightblue"})
00203         title = "{} version of the best trajectory vs distance traveled | {} view".format(guide_metr, metrics[i])
00204         filename = '{}_version_of_best_traj_{}_vs_dist'.format(guide_metr, metrics[i])
00205         filename = os.path.join(common_path, filename)
00206         fig_num = single_plot(fig_num, ax_prop, dist_arr, tot_dist[i], legend_names.copy(), '-', 1, bsf=False, rev=True, extra_line=extra_line,
      shrink=False, xlab="Distance to the goal, {}".format(metr_units[metrics[i]]), ylab="Past distance, {}".format(metr_units[metrics[i]]),
      title=title, filename=filename)
00207 
00208         for j in range(len(dist_arr)):  # iterate over dbs
00209             max_pos_metr_val = max(dist_arr[j])
00210             ax_prop = {"min_lim_x": 0 - max_len / 80, "max_lim_x": max_len + max_len / 80, "min_lim_y": 0, "max_lim_y": max_pos_metr_val +
      max_pos_metr_val / 80, "min_ax_x": 0,
00211                        "max_ax_x": max_len + max_len / 80, "min_ax_y": 0, "max_ax_y": max_pos_metr_val + max_pos_metr_val / 80, "ax_step_x":
      max_len / 16, "ax_step_y": max_pos_metr_val / 20}
00212             if metr_units[metrics[i]] == 'contacts':
```

```
00213                 extra_line = [{"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({} {})".format(metrics[i].upper(),
      int(init_metr), metr_units[metrics[i]]), "col": "darkmagenta"},
00214                               {"ax_type": 'hor', "val": min(dist_arr[j]), "name": "The lowest {} metric ({} {})".format(metrics[i].upper(),
      int(min(dist_arr[j])), metr_units[metrics[i]]), "col": "darkgreen"}]
00215             else:
00216                 extra_line = [{"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metrics[i].upper(),
      init_metr, metr_units[metrics[i]]), "col": "darkmagenta"},
00217                               {"ax_type": 'hor', "val": min(dist_arr[j]), "name": "The lowest {} metric ({:3.2f}
      {})".format(metrics[i].upper(), min(dist_arr[j]), metr_units[metrics[i]]), "col": "darkgreen"}]
00218
00219             if metrics[i] == 'rmsd':
00220                 extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units[metrics[i]]), "col":
      "midnightblue"})
00221             title = "{} version of the best trajectory | {} view".format(guide_metr, metrics[i])
00222             filename = "{}_version_of_best_traj_{}_only_{}".format(guide_metr, metrics[i], filenames_db[j].split('.')[0])
00223             filename = os.path.join(common_path, filename)
00224             fig_num = single_plot(fig_num, ax_prop, [dist_arr[j]], None, [legend_names[j]], '-', 1, bsf=False, rev=False,
      extra_line=extra_line, shrink=True, xlab="Steps (20ps each)", ylab="Distance to the goal, {}".format(metr_units[metrics[i]]), title=title,
      filename=filename)
00225
00226             max_tot_dist = max([dist[-1] for dist in [tot_dist[i][j]]])
00227             ax_prop = {"min_lim_x": max_pos_metr_val + max_pos_metr_val / 80, "max_lim_x": 0 - max_pos_metr_val / 80, "min_lim_y": 0 -
      max_tot_dist / 80, "max_lim_y": max_tot_dist + max_tot_dist / 80, "min_ax_x": 0,
00228                           "max_ax_x": max_pos_metr_val + max_pos_metr_val / 80, "min_ax_y": 0, "max_ax_y": max_tot_dist + max_tot_dist / 80,
      "ax_step_x": max_pos_metr_val / 20, "ax_step_y": max_tot_dist / 20}
00229             if metr_units[metrics[i]] == 'contacts':
00230                 extra_line = [{"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({} {})".format(metrics[i].upper(),
      int(init_metr), metr_units[metrics[i]]), "col": "darkmagenta"},
00231                               {"ax_type": 'ver', "val": min(dist_arr[j]), "name": "The lowest {} metric ({} {})".format(metrics[i].upper(),
      int(min(dist_arr[j])), metr_units[metrics[i]]), "col": "darkgreen"}]
00232             else:
00233                 extra_line = [{"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metrics[i].upper(),
      init_metr, metr_units[metrics[i]]), "col": "darkmagenta"},
00234                               {"ax_type": 'ver', "val": min(dist_arr[j]), "name": "The lowest {} metric ({:3.2f}
      {})".format(metrics[i].upper(), min(dist_arr[j]), metr_units[metrics[i]]), "col": "darkgreen"}]
00235             if metrics[i] == 'rmsd':
00236                 extra_line.append({"ax_type": 'ver', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units[metrics[i]]), "col":
      "midnightblue"})
00237             title = "{} version of the best trajectory vs distance traveled | {} view".format(guide_metr, metrics[i])
00238             filename = '{}_version_of_best_traj_{}_vs_dist_only_{}'.format(guide_metr, metrics[i], filenames_db[j].split('.')[0])
00239             filename = os.path.join(common_path, filename)
00240             fig_num = single_plot(fig_num, ax_prop, [dist_arr[j]], [tot_dist[i][j]], [legend_names[j]], '-', 1, bsf=False, rev=True,
      extra_line=extra_line, shrink=False, xlab="Distance to the goal, {}".format(metr_units[metrics[i]]), ylab="Past distance,
      {}".format(metr_units[metrics[i]]), title=title, filename=filename)
00241
00242             max_pos_metr_val = dist_arr[j][0]
00243             min_pos_metr_val = dist_arr[j][-1]
00244             if min_pos_metr_val > max_pos_metr_val:
00245                 min_pos_metr_val, max_pos_metr_val = max_pos_metr_val, min_pos_metr_val
00246
00247
00248             loc_len = len(dist_arr[j])
00249             for k in range(len(goal_dist)):
00250                 if i != k:
00251                     max_pos_metr2_val = goal_dist[k][j][0]
00252                     min_pos_metr2_val = goal_dist[k][j][-1]
00253                     if max_pos_metr2_val < min_pos_metr2_val:
00254                         max_pos_metr2_val, min_pos_metr2_val = min_pos_metr2_val, max_pos_metr2_val
00255
00256                     divider_min = 15.0
00257                     divider_max = 10.0
00258
00259                     while divider_min > 0.1:
00260                         if (min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / divider_min) < min(goal_dist[k][j]) and
      min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) / divider_min < min(
00261                                 dist_arr[j]):
00262                             break
00263                         divider_min -= 0.05
00264
00265                     while divider_max > 0.1:
00266                         if (max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max) > max(goal_dist[k][j]) and
      max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max > max(
00267                                 dist_arr[j]):
00268                             break
00269                         divider_max -= 0.05
00270
00271                     ax_prop = {"min_lim_x": 0 - loc_len / 80, "max_lim_x": loc_len + loc_len / 80, "min_lim_y": min_pos_metr_val -
      (max_pos_metr_val - min_pos_metr_val) / divider_min,
00272                                "max_lim_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max, "min_ax_x": 0,
00273                                "max_ax_x": loc_len + loc_len / 80, "min_ax_y": min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) /
      divider_min, "max_ax_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max,
```

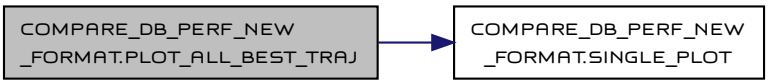
```
00274                             "ax_step_x": math.floor(loc_len / 16), "ax_step_y": (max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max - min_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_min) / 20}
00275                     ax2_prop = {"min_lim_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / divider_min, "max_lim_y": max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max,
00276                                 "min_ax_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / divider_min, "max_ax_y": max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max,  "ax_step_y": (max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max - min_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_min) / 20,
00277                                 "label": "Distance to the goal ({}), {}".format(metrics[k].upper(), metr_units[metrics[k]]), "line_name": '{} ({})'.format(legend_names[j], metrics[k].upper())}
00278                     if metr_units[metrics[i]] == 'contacts':
00279                         extra_line = [
00280                             {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({} {})".format(metrics[i].upper(), int(init_metr), metr_units[metrics[i]]), "col": "darkmagenta"},
00281                             {"ax_type": 'hor', "val": min(dist_arr[j]), "name": "The lowest {} metric ({} {})".format(metrics[i].upper(), int(min(dist_arr[j])), metr_units[metrics[i]]), "col": "darkgreen"}]
00282                     else:
00283                         extra_line = [
00284                             {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metrics[i].upper(), init_metr, metr_units[metrics[i]]), "col": "darkmagenta"},
00285                             {"ax_type": 'hor', "val": min(dist_arr[j]), "name": "The lowest {} metric ({:3.2f} {})".format(metrics[i].upper(), min(dist_arr[j]), metr_units[metrics[i]]), "col": "darkgreen"}]
00286                     if metrics[i] == 'rmsd':
00287                         extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units[metrics[i]]), "col": "midnightblue"})
00288                     title = "{} version of the best trajectory | {} view vs {} view".format(guide_metr, metrics[i], metrics[k])
00289                     filename = "{}_version_of_best_traj_{}_only_{}_vs_{}".format(guide_metr, metrics[i], filenames_db[j].split('.')[0], metrics[k])
00290                     filename = os.path.join(common_path, filename)
00291                     try:
00292                         fig_num = single_plot(fig_num, ax_prop, [dist_arr[j]], None, ['{} ({})'.format(legend_names[j], metrics[i].upper())], '-', 1, bsf=False, rev=False, extra_line=extra_line, shrink=True, xlab="Steps (20ps each)",
00293                                               ylab="Distance to the goal ({}), {}".format(metrics[i].upper(), metr_units[metrics[i]]), title=title, filename=filename, second_ax=ax2_prop, sec_arr=goal_dist[k][j])
00294                     except Exception as e:
00295                         print('Error in generation of {}'.format(filename))
00296
00297             loc_len = len(dist_arr[j])
00298             # prot_name, ff = legend_names[j].split(' ')
00299             if 'AMBER' in legend_names[j].upper():
00300                 ff = 'amber'
00301             elif 'CHARM' in legend_names[j].upper():
00302                 ff = 'charm'
00303             elif 'GROMOS' in legend_names[j].upper():
00304                 ff = 'gromos'
00305             elif 'OPLS' in legend_names[j].upper():
00306                 ff = 'opls'
00307
00308             if 'TRP' in legend_names[j].upper() or '1L2Y' in legend_names[j].upper():
00309                 prot_name = 'TRP'
00310             elif 'VIL' in legend_names[j].upper() or '1YRF' in legend_names[j].upper():
00311                 prot_name = 'VIL'
00312             elif 'GB1' in legend_names[j].upper():
00313                 prot_name = 'GB1'
00314
00315             if '2ND' in legend_names[j].upper():
00316                 rn = 2
00317             elif '1ST' in legend_names[j].upper():
00318                 rn = 1
00319             else:
00320                 rn = None
00321             # if '_' in ff:
00322             #     ff, rn = ff.split('_')
00323             path_to_ener = "/home/vanya/Documents/Phillips/GMDA/Latest_results"
00324             path_to_ener1 = os.path.join(path_to_ener, prot_name)
00325             if rn is not None:
00326                 path_to_ener1 = os.path.join(path_to_ener1, "run_{}".format(rn))
00327             # path_to_ener2 = os.path.join(path_to_ener1, ff, 'LJ_energy')
00328             # np_ener_file = os.path.join(path_to_ener2, '{}_combined_energy_best_full_step.npy'.format(guide_metr))
00329             # ener_arr = np.load(np_ener_file).swapaxes(0, 1)[1]
00330             # ener_arr = ener_arr[-loc_len:]  # trim, so we have same number of steps
00331             # if len(ener_arr) != loc_len:
00332             #     print('kva')
00333             #
00334             # max_pos_metr2_val = ener_arr[0]
00335             # min_pos_metr2_val = ener_arr[-1]
00336             #
00337             # ax_prop = {"min_lim_x": 0 - loc_len / 80, "max_lim_x": loc_len + loc_len / 80, "min_lim_y": min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) / 5.0,
00338             #            "max_lim_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / 10, "min_ax_x": 0,
00339             #            "max_ax_x": loc_len + loc_len / 80, "min_ax_y": min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) / 5.0,
00340             #            "max_ax_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / 10,
```

```
00341             #            "ax_step_x": loc_len / 16, "ax_step_y": (max_pos_metr_val - min_pos_metr_val) / 20}
00342             # ax2_prop = {"min_lim_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / 5.0, "max_lim_y": max_pos_metr2_val +
      (max_pos_metr2_val - min_pos_metr2_val) / 10,
00343             #             "min_ax_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / 5.0, "max_ax_y": max_pos_metr2_val +
      (max_pos_metr2_val - min_pos_metr2_val) / 10,
00344             #             "ax_step_y": (max_pos_metr2_val - min_pos_metr2_val) / 20,
00345             #             "label": "LJ energy, {}".format('kJ/mol'), "line_name": 'LJ:SR interaction energy ({})'.format('kJ/mol')}
00346             # extra_line = [{"ax_type": 'hor', "val": init_metr, "name": "initial {} metric ({:3.2f} {})".format(metrics[i], init_metr,
      metr_units[metrics[i]]), "col": "darkmagenta"}]
00347             # if metrics[i] == 'rmsd':
00348             #     extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "typical folding mark (2.7 {})".format(metr_units[metrics[i]]),
      "col": "midnightblue"})
00349             # title = "{} version of the best trajectory | {} view vs LJ:SR view".format(guide_metr, metrics[i])
00350             # filename = "{}_version_of_best_traj_{}_only_{}_vs_{}".format(guide_metr, metrics[i], filenames_db[j].split('.')[0], 'lj_energy')
00351             # filename = os.path.join(common_path, filename)
00352             # fig_num = single_plot(fig_num, ax_prop, [dist_arr[j]], None, ['{} ({})'.format(legend_names[j], metrics[i])], '-', 1, bsf=False,
      rev=False, extra_line=extra_line, shrink=True,
00353             #                       xlab="steps (20ps each)",
00354             #                       ylab="to goal ({}), {}".format(metrics[i], metr_units[metrics[i]]), title=title, filename=filename,
      second_ax=ax2_prop, sec_arr=ener_arr)
00355             #
00356             #
00357             # path_to_ener2 = os.path.join(path_to_ener1, ff, 'CL_energy')
00358             # np_ener_file = os.path.join(path_to_ener2, '{}_combined_energy_best_full_step.npy'.format(guide_metr))
00359             # ener_arr = np.load(np_ener_file).swapaxes(0, 1)[1]
00360             # ener_arr = ener_arr[-loc_len:]  # trim, so we have same number of steps
00361             #
00362             # max_pos_metr2_val = ener_arr[0]
00363             # min_pos_metr2_val = ener_arr[-1]
00364             #
00365             # ax_prop = {"min_lim_x": 0 - loc_len / 80, "max_lim_x": loc_len + loc_len / 80, "min_lim_y": min_pos_metr_val - (max_pos_metr_val
      - min_pos_metr_val) / 5.0,
00366             #            "max_lim_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / 10, "min_ax_x": 0,
00367             #            "max_ax_x": loc_len + loc_len / 80, "min_ax_y": min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) / 5.0,
00368             #            "max_ax_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / 10,
00369             #            "ax_step_x": loc_len / 16, "ax_step_y": (max_pos_metr_val - min_pos_metr_val) / 20}
00370             # ax2_prop = {"min_lim_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / 5.0, "max_lim_y": max_pos_metr2_val +
      (max_pos_metr2_val - min_pos_metr2_val) / 10,
00371             #             "min_ax_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / 5.0, "max_ax_y": max_pos_metr2_val +
      (max_pos_metr2_val - min_pos_metr2_val) / 10,
00372             #             "ax_step_y": (max_pos_metr2_val - min_pos_metr2_val) / 20,
00373             #             "label": "CL energy, {}".format('kJ/mol'), "line_name": 'CL:SR interaction energy ({})'.format('kJ/mol')}
00374             # extra_line = [{"ax_type": 'hor', "val": init_metr, "name": "initial {} metric ({:3.2f} {})".format(metrics[i], init_metr,
      metr_units[metrics[i]]), "col": "darkmagenta"}]
00375             # if metrics[i] == 'rmsd':
00376             #     extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "typical folding mark (2.7 {})".format(metr_units[metrics[i]]),
      "col": "midnightblue"})
00377             # title = "{} version of the best trajectory | {} view vs CL:SR view".format(guide_metr, metrics[i])
00378             # filename = "{}_version_of_best_traj_{}_only_{}_vs_{}".format(guide_metr, metrics[i], filenames_db[j].split('.')[0], 'cl_energy')
00379             # filename = os.path.join(common_path, filename)
00380             # fig_num = single_plot(fig_num, ax_prop, [dist_arr[j]], None, ['{} ({})'.format(legend_names[j], metrics[i])], '-', 1, bsf=False,
      rev=False, extra_line=extra_line, shrink=True,
00381             #                       xlab="steps (20ps each)",
00382             #                       ylab="to goal ({}), {}".format(metrics[i], metr_units[metrics[i]]), title=title, filename=filename,
      second_ax=ax2_prop, sec_arr=ener_arr)
00383
00384
00385
00386
00387             path_to_ener2 = os.path.join(path_to_ener1, ff, 'PT_energy')
00388             np_ener_file = os.path.join(path_to_ener2, '{}_correct_index_energy.npy'.format(guide_metr))
00389             ener_arr = np.load(np_ener_file).swapaxes(0, 1)[1]
00390             ener_arr = ener_arr[-loc_len:]  # trim, so we have same number of steps
00391
00392             max_pos_metr2_val = ener_arr[0]
00393             min_pos_metr2_val = ener_arr[-1]
00394
00395             divider_min = 5.0
00396             divider_max = 10.0
00397
00398             while divider_min > 0.1:
00399                 if (min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / divider_min) < min(ener_arr) and min_pos_metr_val -
      (max_pos_metr_val - min_pos_metr_val) / divider_min < min(
00400                         dist_arr[j]):
00401                     break
00402                 divider_min -= 0.05
00403
00404             while divider_max > 0.1:
00405                 if (max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max) > max(ener_arr) and max_pos_metr_val +
      (max_pos_metr_val - min_pos_metr_val) / divider_max > max(
00406                         dist_arr[j]):
```

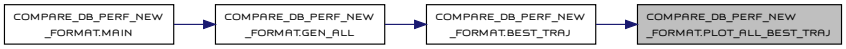
```
00407                   break
00408               divider_max -= 0.05
00409
00410           ax_prop = {"min_lim_x": 0 - loc_len / 80, "max_lim_x": loc_len + loc_len / 80, "min_lim_y": min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) / divider_min,
00411                      "max_lim_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max, "min_ax_x": 0,
00412                      "max_ax_x": loc_len + loc_len / 80, "min_ax_y": min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) / divider_min,
00413                      "max_ax_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max,
00414                      "ax_step_x": math.floor(loc_len / 16), "ax_step_y": (max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max - min_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_min) / 20}
00415           ax2_prop = {"min_lim_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / divider_min, "max_lim_y": max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max,
00416                       "min_ax_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / divider_min, "max_ax_y": max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max,
00417                       "ax_step_y": (max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max - min_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_min) / 20,
00418                       "label": "Potential energy, {}".format('kJ/mol'), "line_name": 'Potential energy ({})'.format('kJ/mol')}
00419           if metr_units[metrics[i]] == 'contacts':
00420               extra_line = [
00421                   {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({} {})".format(metrics[i].upper(), int(init_metr), metr_units[metrics[i]]), "col": "darkmagenta"},
00422                   {"ax_type": 'hor', "val": min(dist_arr[j]), "name": "The lowest {} metric ({} {})".format(metrics[i].upper(), int(min(dist_arr[j])), metr_units[metrics[i]]), "col": "darkgreen"}]
00423           else:
00424               extra_line = [
00425                   {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metrics[i].upper(), init_metr, metr_units[metrics[i]]), "col": "darkmagenta"},
00426                   {"ax_type": 'hor', "val": min(dist_arr[j]), "name": "The lowest {} metric ({:3.2f} {})".format(metrics[i].upper(), min(dist_arr[j]), metr_units[metrics[i]]), "col": "darkgreen"}]
00427           if metrics[i] == 'rmsd':
00428               extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units[metrics[i]]), "col": "midnightblue"})
00429           title = "{} version of the best trajectory | {} view vs Potential energy view".format(guide_metr, metrics[i])
00430           filename = "{}_version_of_best_traj_{}_only_{}_vs_{}".format(guide_metr, metrics[i], filenames_db[j].split('.')[0], 'pt_energy')
00431           filename = os.path.join(common_path, filename)
00432           fig_num = single_plot(fig_num, ax_prop, [dist_arr[j]], None, ['{} ({})'.format(legend_names[j], metrics[i].upper())], '-', 1, bsf=False, rev=False, extra_line=extra_line, shrink=True,
00433                                 xlab="Steps (20ps each)",
00434                                 ylab="Distance to the goal ({}), {}".format(metrics[i].upper(), metr_units[metrics[i]]), title=title, filename=filename, second_ax=ax2_prop, sec_arr=ener_arr)
00435
00436
00437
00438       # max_len = max([len(arr) for arr in rmsd_dist_arr])
00439       # init_metr = rmsd_dist_arr[0][0]
00440       # metr_units = 'A'
00441       # ax_prop = {"min_lim_x": 0 - +max_len/80, "max_lim_x": max_len + max_len/80, "min_lim_y": 0 - init_metr/80, "max_lim_y": init_metr + init_metr/80, "min_ax_x": 0, "max_ax_x": max_len + max_len/80, "min_ax_y": 0, "max_ax_y": init_metr+init_metr/80, "ax_step_x": max_len / 16, "ax_step_y": init_metr / 20}
00442       # extra_line = {"ax_type": 'hor', "val": init_metr, "name": "initial {} metric ({:3.2f} {})".format('rmsd', init_metr, metr_units)}
00443       # # title = "{} | to goal vs traveled | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00444       # # filename = "{}_to_goal_vs_traveled_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00445       # # filename = os.path.join(custom_path, filename)
00446       # title = 'kva'
00447       # filename = 'test_best'
00448       # fig_num = single_plot(fig_num, ax_prop, rmsd_dist_arr, None, legend_names.copy(), '-', 1, bsf=False, rev=False, extra_line=extra_line, shrink=True, xlab="steps (20ps each)", ylab="to goal, {}".format(metr_units), title=title, filename=filename)
00449       #
00450       # max_tot_dist = max([dist[-1] for dist in rmsd_tot_dist_arr])
00451       # # ax_prop = {"min_lim_x": 0 - +max_len/80, "max_lim_x": max_tot_dist + max_tot_dist/80, "min_lim_y": 0 - init_metr/80, "max_lim_y": init_metr + init_metr/80, "min_ax_x": 0, "max_ax_x": max_tot_dist + max_tot_dist/80, "min_ax_y": 0, "max_ax_y": init_metr+init_metr/80, "ax_step_x": max_tot_dist / 16, "ax_step_y": init_metr / 20}
00452       # ax_prop = {"min_lim_x": init_metr + init_metr / 80, "max_lim_x": 0 - init_metr / 80, "min_lim_y": 0 - +max_len / 80, "max_lim_y": max_tot_dist + max_tot_dist / 80, "min_ax_x": 0,
00453       #            "max_ax_x": init_metr + init_metr / 80, "min_ax_y": 0, "max_ax_y": max_tot_dist + max_tot_dist / 80, "ax_step_x": init_metr / 20, "ax_step_y": max_tot_dist / 16}
00454       # extra_line = {"ax_type": 'ver', "val": init_metr, "name": "initial {} metric ({:3.2f} {})".format('rmsd', init_metr, metr_units)}
00455       # # title = "{} | to goal vs traveled | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00456       # # filename = "{}_to_goal_vs_traveled_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00457       # # filename = os.path.join(custom_path, filename)
00458       # title = 'kva'
00459       # filename = 'test_best'
00460       # fig_num = single_plot(fig_num, ax_prop, rmsd_dist_arr, rmsd_tot_dist_arr, legend_names.copy(), '-', 1, bsf=False, rev=True, extra_line=extra_line, shrink=False, xlab="to goal, {}".format(metr_units), ylab="steps (20ps each)", title=title, filename=filename)
00461
00462
00463
00464
00465
```

References single_plot().
Referenced by best_traj().

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.1.6 plot_all_metrics()

```
int compare_db_perf_new_format.plot_all_metrics (
        int  fig_num,
        list  cur_arr,
        list  filenames_db,
        list  legend_names,
        str  guide_metr,
        str  common_path )
```

General force field comparison: sampling, best_so_far, dist traveled.

```
    int  fig_num: figure number, it should not matter, since we close all figures regularly
    list  cur_arr:
    list  filenames_db:
    list  legend_names:
    str  guide_metr:
    str  common_path:
```

Returns

```
     :return: figure number, it should not matter, since we close all figures regularly
```

Definition at line 520 of file compare_db_perf_new_format.py.

```
00520     """
00521     best_metr_dic = {'rmsd': 'bsfr', 'angl': 'bsfn', 'andh': 'bsfh', 'and': 'bsfa', 'xor': 'bsfx'}
00522     metr_units = {'rmsd': 'Å', 'angl': '', 'andh': 'contacts', 'and': 'contacts', 'xor': 'contacts'}
00523     qry = 'SELECT a.{}_goal_dist FROM main_storage a join visited b on a.id=b.id order by b.vid'.format(guide_metr)
00524     result_arr = [cur.execute(qry) for cur in cur_arr]
00525     fetched_all_arr = [res.fetchall() for res in result_arr]
00526     filt_res_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00527     init_rmsd = filt_res_arr[0][0]
00528     max_non_init_rmsd = max(max(elem) for elem in filt_res_arr)
00529     common_point = max([min(elem) for elem in filt_res_arr])
00530 
00531     ind_arr = list()
00532     for rmsd_for_db in filt_res_arr:
00533         i = 0
00534         while common_point < rmsd_for_db[i]:
00535             i += 1
00536         ind_arr.append(i)
00537 
00538     # print('To reach common min point of {}A ({})'.format(common_point, guide_metr))
00539     # for i, db in enumerate(filenames_db):
00540     #     print('{} : {} steps'.format(db.split('.')[0], ind_arr[i]))
00541 
00542 
00543 
00544     #     ################# CUT ######################
00545 
00546     # qry = "select a.bsfr, b.rmsd_tot_dist, b.rmsd_goal_dist from log a join main_storage b on a.id=b.id where a.dst='VIZ' and a.bsfr>'{}'
       order by a.lid".format(common_point)
```

```
00547     qry = "select a.{0}, b.{1}_tot_dist, b.{1}_goal_dist, c.vid from main_storage b join visited c on c.id=b.id join (select id, {0} from log
       where dst='VIZ' group by id) a on a.id=b.id where a.{0}>'{2}'  order by c.vid".format(best_metr_dic[guide_metr], guide_metr, common_point)
00548     result_arr = [cur.execute(qry) for cur in cur_arr]
00549     [res.fetchone() for res in result_arr]
00550     fetched_all_arr = [res.fetchall() for res in result_arr]
00551     bsf_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00552     for i in range(len(bsf_arr)):
00553         bsf_arr[i].insert(0, init_rmsd)
00554     for j in range(len(bsf_arr)):
00555         for i in range(len(bsf_arr[j]) - 1):
00556             if bsf_arr[j][i] < bsf_arr[j][i + 1]:
00557                 bsf_arr[j][i+1] = bsf_arr[j][i]
00558     trav_arr = [[dist[1] for dist in goal_dist] for goal_dist in fetched_all_arr]
00559     to_goal_arr = [[dist[2] for dist in goal_dist] for goal_dist in fetched_all_arr]
00560 
00561     max_len = max([len(goal_dist) for goal_dist in fetched_all_arr])
00562     custom_path = '{}/ALL/'.format(common_path)
00563     try:
00564         os.mkdir(custom_path)
00565     except:
00566         pass
00567 
00568     try:
00569         max_trav = max([max(elem) for elem in trav_arr])
00570         custom_path = '{}/ALL/cut/'.format(common_path)
00571         try:
00572             os.mkdir(custom_path)
00573         except:
00574             pass
00575         # shrink is True since everything is in order, there is no difference whether to pass index or generate it
00576         fig_num = plot_set(fig_num, to_goal_arr, legend_names, max_len, max_non_init_rmsd, init_rmsd, bsf_arr, common_point, max_trav,
       trav_arr, "cut", guide_metr, metr_units[guide_metr], 'all', custom_path, shrink=True)
00577     except:
00578         print('Not all trajecotories have a common point', [len(elem) for elem in trav_arr])
00579 
00580     #     ################ FULL ####################
00581 
00582     # qry = "select a.bsfr, b.rmsd_tot_dist, b.rmsd_goal_dist from log a join main_storage b on a.id=b.id where a.dst='VIZ' order by a.lid"
00583     qry = "select a.{0}, b.{1}_tot_dist, b.{1}_goal_dist, c.vid from main_storage b join visited c on c.id=b.id join (select id, max({0}) as
       {0} from log where dst='VIZ' group by id) a on a.id=b.id order by c.vid".format(best_metr_dic[guide_metr], guide_metr)
00584     result_arr = [cur.execute(qry) for cur in cur_arr]
00585     [res.fetchone() for res in result_arr]
00586     fetched_all_arr = [res.fetchall() for res in result_arr]
00587     bsf_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00588     for i in range(len(bsf_arr)):
00589         bsf_arr[i].insert(0, init_rmsd)
00590     for j in range(len(bsf_arr)):
00591         for i in range(len(bsf_arr[j]) - 1):
00592             if bsf_arr[j][i] < bsf_arr[j][i + 1]:
00593                 bsf_arr[j][i+1] = bsf_arr[j][i]
00594 
00595     trav_arr = [[dist[1] for dist in goal_dist] for goal_dist in fetched_all_arr]
00596     to_goal_arr = [[dist[2] for dist in goal_dist] for goal_dist in fetched_all_arr]
00597 
00598     max_len = max([len(goal_dist) for goal_dist in fetched_all_arr])
00599     max_trav = max([max(elem) for elem in trav_arr])
00600     common_point = min([min(elem) for elem in filt_res_arr])
00601 
00602     custom_path = '{}/ALL/full/'.format(common_path)
00603     try:
00604         os.mkdir(custom_path)
00605     except:
00606         pass
00607     # shrink is True since everything is in order, there is no difference whether to pass index or generate it
00608     fig_num = plot_set(fig_num, to_goal_arr, legend_names, max_len, max_non_init_rmsd, init_rmsd, bsf_arr, common_point, max_trav, trav_arr,
       "full", guide_metr, metr_units[guide_metr], 'all', custom_path, shrink=True)
00609 
00610 
00611     return fig_num, init_rmsd
00612 
00613 
00614 def plot_only_one_metric(fig_num: int, cur_arr: list, filenames_db: list, init_rmsd: float, legend_names: list, metric_name: str, guide_metr:
       str, common_path: str) -> int:
00615     """
00616 
00617     Args:
00618          int fig_num:
00619          list cur_arr:
00620          list filenames_db:
00621          float init_rmsd:
00622          list legend_names:
```

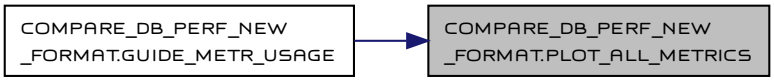
```
00623          str  metric_name:
00624          str  guide_metr:
00625          str  common_path:
00626
00627      Returns:
00628          :return: figure number
```

References plot_set().

Referenced by guide_metr_usage().

Here is the call graph for this function:

Here is the caller graph for this function:



3.1.1.7 plot_only_one_metric()

```
int compare_db_perf_new_format.plot_only_one_metric (
                int  fig_num,
                list  cur_arr,
                list  filenames_db,
                float  init_rmsd,
                list  legend_names,
                str  metric_name,
                str  guide_metr,
                str  common_path )
```

Definition at line 629 of file compare_db_perf_new_format.py.

```
00629     """
00630     best_metr_dic = {'rmsd': 'bsfr', 'angl': 'bsfn', 'andh': 'bsfh', 'and': 'bsfa', 'xor': 'bsfx'}
00631     metr_units = {'rmsd': 'Å', 'angl': '', 'andh': 'contacts', 'and': 'contacts', 'xor': 'contacts'}
00632     # qry = "SELECT a.rmsd_goal_dist, b.vid FROM main_storage a join visited b on a.id=b.id join log c on a.id=c.id where c.cur_metr='{}' order
       by b.vid".format(metric_name)
00633     qry = "select a.{0}_goal_dist, b.vid from main_storage a join visited b on a.id=b.id join (select id, cur_metr from log where dst='VIZ'
       group by id) c on c.id=b.id where c.cur_metr='{1}' order by b.vid".format(guide_metr, metric_name)
00634     result_arr = [cur.execute(qry) for cur in cur_arr]
00635     fetched_all_arr = [res.fetchall() for res in result_arr]
00636     filt_res_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00637     # init_rmsd = filt_res_arr[0][0]
00638     max_non_init_rmsd = max(max(elem) for elem in filt_res_arr)
00639     common_point = max([min(elem) for elem in filt_res_arr])
00640
00641     ind_arr = list()
00642     for rmsd_for_db in filt_res_arr:
00643         i = 0
00644         while common_point < rmsd_for_db[i]:
00645             i += 1
00646         ind_arr.append(i)
00647
00648     # print('To reach common min point of {}A (rmsd)'.format(common_point))
00649     # for i, db in enumerate(filenames_db):
00650     #     print('{} : {} steps'.format(db.split('.')[0], ind_arr[i]))
00651
00652
00653     #     ################ FULL ####################
00654
00655     # qry = "select a.bsfr, b.rmsd_tot_dist, b.rmsd_goal_dist, c.vid from log a join main_storage b on a.id=b.id join visited c on c.id=a.id
       where a.dst='VIZ' and a.cur_metr='{}' order by a.lid".format(metric_name)
```

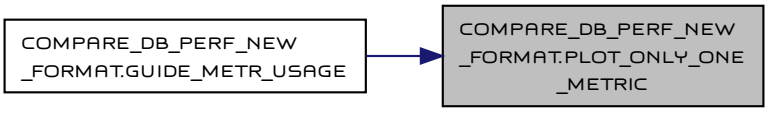
```
00656     qry = "select c.{0}, a.{1}_tot_dist, a.{1}_goal_dist, b.vid from main_storage a join visited b on a.id=b.id join (select id, max({0}) as
      {0}, cur_metr from log where dst='VIZ' group by id) c on c.id=b.id where c.cur_metr='{2}' order by b.vid".format(best_metr_dic[guide_metr],
      guide_metr, metric_name)
00657     result_arr = [cur.execute(qry) for cur in cur_arr]
00658     [res.fetchone() for res in result_arr]
00659     fetched_all_arr = [res.fetchall() for res in result_arr]
00660     bsf_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00661     for i in range(len(bsf_arr)):
00662         bsf_arr[i].insert(0, init_rmsd)
00663     for j in range(len(bsf_arr)):
00664         for i in range(len(bsf_arr[j]) - 1):
00665             if bsf_arr[j][i] < bsf_arr[j][i + 1]:
00666                 bsf_arr[j][i+1] = bsf_arr[j][i]
00667     trav_arr = [[dist[1] for dist in goal_dist] for goal_dist in fetched_all_arr]
00668     to_goal_arr = [[dist[2] for dist in goal_dist] for goal_dist in fetched_all_arr]
00669     non_shr = [[dist[3] for dist in goal_dist] for goal_dist in fetched_all_arr]
00670     # for i in range(len(non_shr)):
00671     #     non_shr[i].insert(0, 0)
00672 
00673     max_len = max([len(goal_dist) for goal_dist in fetched_all_arr])
00674     max_trav = max([max(elem) for elem in trav_arr])
00675     common_point = min([min(elem) for elem in filt_res_arr])
00676     custom_path = '{}/full/'.format(common_path)
00677     try:
00678         os.mkdir(custom_path)
00679     except:
00680         pass
00681 
00682     fig_num = plot_set(fig_num, to_goal_arr, legend_names, max_len, max_non_init_rmsd, init_rmsd, bsf_arr, common_point, max_trav, trav_arr,
      "full", guide_metr, metr_units[guide_metr], metric_name, custom_path, shrink=True)
00683     max_len = max([max(arr) for arr in non_shr])
00684     fig_num = plot_set(fig_num, to_goal_arr, legend_names, max_len, max_non_init_rmsd, init_rmsd, bsf_arr, common_point, max_trav, trav_arr,
      "full", guide_metr, metr_units[guide_metr], metric_name, custom_path, shrink=False, non_shrink_arr=non_shr)
00685 
00686     return fig_num
00687 
00688 
00689 def plot_set(fig_num: int, to_goal_arr: list, legend_names: list, max_len: float, max_non_init_rmsd: float,
00690              init_metr: float, bsf_arr: list, common_point: float, max_trav: float, trav_arr: list, full_cut: str,
00691              metric: str, metr_units: str, same: str, custom_path: str, shrink: bool, non_shrink_arr: list = None) -> int:
00692     """
00693 
00694     Args:
00695          int fig_num:
00696          list to_goal_arr:
00697          list legend_names:
00698          float max_len:
00699          float max_non_init_rmsd:
00700          float init_metr:
00701          float list bsf_arr:
00702          float common_point:
00703          float max_trav:
00704          list trav_arr:
00705          str  full_cut:
00706          str  metric:
00707          str  metr_units:
00708          str  same:
00709          str  custom_path:
00710          bool shrink:
00711          list non_shrink_arr:
00712 
00713     Returns:
```

References plot_set().

Referenced by guide_metr_usage().

Here is the call graph for this function:

Here is the caller graph for this function:



3.1.1.8 plot_sep_best_traj()

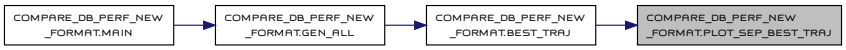
```
def compare_db_perf_new_format.plot_sep_best_traj (
                fig_num,
                cur_arr,
                filenames_db,
                legend_names,
                guide_metr,
                common_path )
```

Definition at line 466 of file compare_db_perf_new_format.py.

```
00466 def plot_sep_best_traj(fig_num, cur_arr, filenames_db, legend_names, guide_metr, common_path):
00467     pass
00468
00469
00470 def guide_metr_usage(fig_num: int, filenames_db: list, legend_names: list, guide_metr: str, common_path: str) -> int:
00471     """
00472
00473     Args:
00474          int fig_num: figure number, it should not matter, since we close all figures regularly
00475          list filenames_db: database names
00476          list legend_names: proper database description
00477          str  guide_metr: main metric for the plot
00478          str  common_path: where to store plots
00479
00480     Returns:
00481         Returns: figure number, it should not matter, since we close all figures regularly
```

Referenced by best_traj().

Here is the caller graph for this function:



3.1.1.9 plot_set()

```
int compare_db_perf_new_format.plot_set (
                int  fig_num,
                list  to_goal_arr,
                list  legend_names,
                float  max_len,
                float  max_non_init_rmsd,
                float  init_metr,
                list  bsf_arr,
                float  common_point,
                float  max_trav,
                list  trav_arr,
                str  full_cut,
                str  metric,
                str  metr_units,
                str  same,
                str  custom_path,
                bool shrink,
                list   non_shrink_arr =  None )
```

Definition at line 714 of file compare_db_perf_new_format.py.

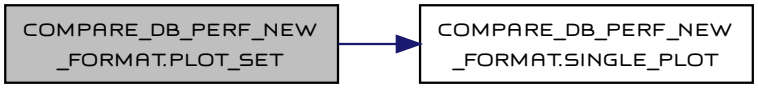
```
00714         :return: fig number
00715         return type: int
00716     """
```

```
00717     # # #### SHRINK
00718     # ax_prop = {"min_lim_x": -max_len/80, "max_lim_x": max_len+max_len/80, "min_lim_y": 0, "max_lim_y":
      max_non_init_rmsd+max_non_init_rmsd/80, "min_ax_x": 0, "max_ax_x": max_len+max_len/80, "min_ax_y": 0, "max_ax_y":
      max_non_init_rmsd+max_non_init_rmsd/80, "ax_step_x": max_len/16, "ax_step_y": max_non_init_rmsd/20}
00719     # extra_line = {"ax_type": 'hor', "val": init_rmsd, "name": "init {} ({:3.2f} {})".format(metric, init_rmsd, metr_units)}
00720     # fig_num = single_plot(fig_num, ax_prop, to_goal_arr, None,            legend_names, '.', 0.3, bsf=False, rev=False,
      extra_line=extra_line, xlab="steps (20ps each)", ylab="to goal, A", title="{} | to goal vs traveled | {} | {}".format(metric, full_cut, same),
      filename="{}_to_goal_vs_traveled_{}_{}".format(metric, full_cut, same))  # to goal vs traveled | cut
00721     #
00722     # ax_prop = {"min_lim_x": -max_len/80, "max_lim_x": max_len+max_len/80, "min_lim_y": 0, "max_lim_y":
      max_non_init_rmsd+max_non_init_rmsd/80, "min_ax_x": 0, "max_ax_x": max_len+max_len/80, "min_ax_y": 0, "max_ax_y":
      max_non_init_rmsd+max_non_init_rmsd/80, "ax_step_x": max_len/16, "ax_step_y": max_non_init_rmsd/20}
00723     # extra_line = {"ax_type": 'hor', "val": init_rmsd, "name": "init {} ({:3.2f} {})".format(metric, init_rmsd, metr_units)}
00724     # fig_num = single_plot(fig_num, ax_prop,  bsf_arr,    None,            legend_names, '-', 1,   bsf=True,  rev=False,
      extra_line=extra_line, xlab="steps (20ps each)", ylab="steps", title="{} | to goal vs best_so_far | {} | {}".format(metric, full_cut, same),
      filename="{}_to_goal_vs_best_so_far_{}_{}".format(metric, full_cut, same))  # to goal vs best_so_far | cut
00725     #
00726     # ax_prop = {"min_lim_x": max_non_init_rmsd, "max_lim_x": common_point-common_point/10, "min_lim_y": -max_len/80, "max_lim_y":
      max_len+max_len/80, "min_ax_x": common_point, "max_ax_x": max_non_init_rmsd, "min_ax_y": 0, "max_ax_y": max_len+max_len/80, "ax_step_x":
      (max_non_init_rmsd-common_point)/16, "ax_step_y": max_len/20}
00727     # extra_line = {"ax_type": 'ver', "val": init_rmsd, "name": "init {} ({:3.2f} {})".format(metric, init_rmsd, metr_units)}
00728     # fig_num = single_plot(fig_num, ax_prop,   bsf_arr,   None,            legend_names, '-',  1,   bsf=True,  rev=True,
      extra_line=extra_line, xlab="to goal, A", ylab="steps", title="{} | best_so_far vs steps | {} | {}".format(metric, full_cut, same),
      filename="{}_best_so_far_vs_steps_{}_{}".format(metric, full_cut, same))  # best_so_far vs steps | cut
00729
00730     # #### NO SHRINK
00731     custom_path = custom_path+'shrink' if shrink else custom_path+'unshrink'
00732     try:
00733         os.mkdir(custom_path)
00734     except:
00735         pass
00736     ax_prop = {"min_lim_x": -max_len/80, "max_lim_x": max_len+max_len/80, "min_lim_y": 0, "max_lim_y": max_non_init_rmsd+max_non_init_rmsd/80,
00737                "min_ax_x": 0, "max_ax_x": max_len+max_len/80, "min_ax_y": 0, "max_ax_y": max_non_init_rmsd+max_non_init_rmsd/80, "ax_step_x":
      math.floor(max_len/16), "ax_step_y": max_non_init_rmsd/20}
00738     if metr_units == 'contacts':
00739         extra_line = [
00740             {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({} {})".format(metric.upper(), int(init_metr), metr_units), "col":
      "darkmagenta"},
00741             {"ax_type": 'hor', "val": min(min(elem) for elem in to_goal_arr), "name": "The lowest {} metric ({} {})".format(metric.upper(),
      int(min(min(elem) for elem in to_goal_arr)), metr_units), "col": "darkgreen"}]
00742     else:
00743         extra_line = [
00744             {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metric.upper(), init_metr, metr_units), "col":
      "darkmagenta"},
00745             {"ax_type": 'hor', "val": min(min(elem) for elem in to_goal_arr), "name": "The lowest {} metric ({:3.2f}
      {})".format(metric.upper(), min(min(elem) for elem in to_goal_arr), metr_units), "col": "darkgreen"}]
00746     if metric == 'rmsd':
00747         extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units), "col": "midnightblue"})
00748     title = "{} | to goal vs traveled | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00749     filename = "{}_to_goal_vs_traveled_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00750     filename = os.path.join(custom_path, filename)
00751     fig_num = single_plot(fig_num, ax_prop, to_goal_arr, non_shrink_arr,  legend_names.copy(), '.', 0.3, bsf=False, rev=False,
      extra_line=extra_line, shrink=shrink, xlab="Steps (20ps each)", ylab="Distance to the goal, {}".format(metr_units), title=title,
      filename=filename)  # to goal vs traveled | cut
00752
00753     for i in range(len(to_goal_arr)):
00754         ff = legend_names[i].split('with')[1].split('ff')[0].strip()
00755         title = "{} | to goal vs traveled | {} | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00756         filename = "{}_to_goal_vs_traveled_{}_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00757         filename = os.path.join(custom_path, filename)
00758         extra_line[1]["val"] = min(to_goal_arr[i])
00759         if metr_units == 'contacts':
00760             extra_line[1]["name"] = "The lowest {} metric ({} {})".format(metric.upper(), int(min(to_goal_arr[i])), metr_units)
00761         else:
00762             extra_line[1]["name"] = "The lowest {} metric ({:3.2f} {})".format(metric.upper(), min(to_goal_arr[i]), metr_units)
00763         fig_num = single_plot(fig_num, ax_prop, [to_goal_arr[i],], [non_shrink_arr[i],] if non_shrink_arr is not None else None,
      [legend_names[i],].copy(), '.', 0.3, bsf=False, rev=False, extra_line=extra_line, shrink=shrink, xlab="Steps (20ps each)",
00764                               ylab="Distance to the goal, {}".format(metr_units), title=title, filename=filename)  # to goal vs traveled | cut
00765
00766     if shrink:
00767         ax_prop = {"min_lim_x": max_non_init_rmsd, "max_lim_x": common_point-common_point/20, "min_lim_y": -max_trav/80, "max_lim_y":
      max_trav+max_trav/80,
00768                    "min_ax_x": common_point, "max_ax_x": max_non_init_rmsd, "min_ax_y": 0, "max_ax_y": max_trav+max_trav/80, "ax_step_x":
      (max_non_init_rmsd-common_point)/20, "ax_step_y": max_trav/20}
00769         if metr_units == 'contacts':
00770             extra_line = [
00771                 {"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({} {})".format(metric.upper(), int(init_metr), metr_units),
      "col": "darkmagenta"},
00772                 {"ax_type": 'ver', "val": min(min(elem) for elem in to_goal_arr), "name": "The lowest {} metric ({} {})".format(metric.upper(),
      int(min(min(elem) for elem in to_goal_arr)), metr_units), "col": "darkgreen"}]
00773         else:
```

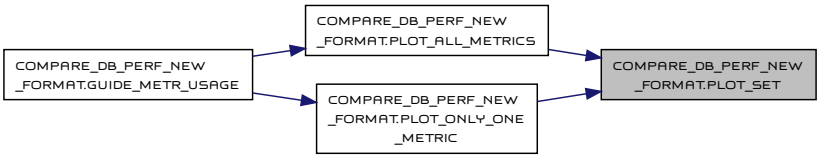
```
00774            extra_line = [
00775                {"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metric.upper(), init_metr, metr_units),
    "col": "darkmagenta"},
00776                {"ax_type": 'ver', "val": min(min(elem) for elem in to_goal_arr), "name": "The lowest {} metric ({:3.2f}
    {})".format(metric.upper(), min(min(elem) for elem in to_goal_arr), metr_units), "col": "darkgreen"}]
00777        if metric == 'rmsd':
00778            extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units), "col":
    "midnightblue"})
00779        title = "{} | traveled vs to_goal | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00780        filename = "{}_traveled_vs_to_goal_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00781        filename = os.path.join(custom_path, filename)
00782        fig_num = single_plot(fig_num, ax_prop, to_goal_arr, trav_arr,    legend_names.copy(), '.', 1,   bsf=False, rev=True,
    extra_line=extra_line, shrink=shrink, xlab="Distance to the goal, {}".format(metr_units), ylab="Past dist, {}".format(metr_units),
    title=title, filename=filename)  # traveled vs to_goal | cut
00783
00784        for i in range(len(to_goal_arr)):
00785            ff = legend_names[i].split('with')[1].split('ff')[0].strip()
00786            title = "{} | traveled vs to_goal | {} | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00787            filename = "{}_traveled_vs_to_goal_{}_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00788            filename = os.path.join(custom_path, filename)
00789            extra_line[1]["val"] = min(to_goal_arr[i])
00790            if metr_units == 'contacts':
00791                extra_line[1]["name"] = "The lowest {} metric ({} {})".format(metric.upper(), int(min(to_goal_arr[i])), metr_units)
00792            else:
00793                extra_line[1]["name"] = "The lowest {} metric ({:3.2f} {})".format(metric.upper(), min(to_goal_arr[i]), metr_units)
00794            fig_num = single_plot(fig_num, ax_prop, [to_goal_arr[i],], [trav_arr[i],], [legend_names[i],].copy(), '.', 1, bsf=False, rev=True,
    extra_line=extra_line, shrink=shrink,
00795                                  xlab="Distance to the goal, {}".format(metr_units), ylab="Past dist, {}".format(metr_units), title=title,
    filename=filename)  # traveled vs to_goal | cut
00796
00797    if not shrink:
00798        for i in range(len(non_shrink_arr)):
00799            non_shrink_arr[i].insert(0, 0)
00800    ax_prop = {"min_lim_x": -max_len / 80, "max_lim_x": max_len + max_len / 80, "min_lim_y": 0, "max_lim_y": init_metr + init_metr / 80,  #
    max_non_init_rmsd + max_non_init_rmsd / 80,
00801               "min_ax_x": 0, "max_ax_x": max_len + max_len / 80, "min_ax_y": 0, "max_ax_y": init_metr + init_metr / 80, "ax_step_x":
    math.floor(max_len / 16), "ax_step_y": init_metr / 20}
00802    if metr_units == 'contacts':
00803        extra_line = [
00804            {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({} {})".format(metric.upper(), int(init_metr), metr_units), "col":
    "darkmagenta"},
00805            {"ax_type": 'hor', "val": min(min(elem) for elem in bsf_arr), "name": "The lowest {} metric ({} {})".format(metric.upper(),
    int(min(min(elem) for elem in bsf_arr)), metr_units), "col": "darkgreen"}]
00806    else:
00807        extra_line = [
00808            {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metric.upper(), init_metr, metr_units), "col":
    "darkmagenta"},
00809            {"ax_type": 'hor', "val": min(min(elem) for elem in bsf_arr), "name": "The lowest {} metric ({:3.2f} {})".format(metric.upper(),
    min(min(elem) for elem in bsf_arr), metr_units), "col": "darkgreen"}]
00810    if metric == 'rmsd':
00811        extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units), "col": "midnightblue"})
00812    title = "{} | to goal vs best_so_far | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00813    filename = "{}_to_goal_vs_best_so_far_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00814    filename = os.path.join(custom_path, filename)
00815    fig_num = single_plot(fig_num, ax_prop, bsf_arr, non_shrink_arr, legend_names.copy(), '-', 1, bsf=True, rev=False, extra_line=extra_line,
    shrink=shrink, xlab="Steps (20ps each)", ylab="Distance to the goal, {}".format(metr_units), title=title, filename=filename)  # to goal vs
    best_so_far | cut
00816    for i in range(len(bsf_arr)):
00817        ff = legend_names[i].split('with')[1].split('ff')[0].strip()
00818        title = "{} | to goal vs best_so_far | {} | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00819        filename = "{}_to_goal_vs_best_so_far_{}_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00820        extra_line[1]["val"] = min(bsf_arr[i])
00821        if metr_units == 'contacts':
00822            extra_line[1]["name"] = "The lowest {} metric ({} {})".format(metric.upper(), int(min(bsf_arr[i])), metr_units)
00823        else:
00824            extra_line[1]["name"] = "The lowest {} metric ({:3.2f} {})".format(metric.upper(), min(bsf_arr[i]), metr_units)
00825        filename = os.path.join(custom_path, filename)
00826        fig_num = single_plot(fig_num, ax_prop, [bsf_arr[i],], [non_shrink_arr[i],] if non_shrink_arr is not None else None,
    [legend_names[i],].copy(), '-', 1, bsf=True, rev=False, extra_line=extra_line, shrink=shrink, xlab="Steps (20ps each)",
00827                              ylab="Distance to the goal, {}".format(metr_units), title=title, filename=filename)  # to goal vs best_so_far |
    cut
00828
00829    ax_prop = {"min_lim_x": max_non_init_rmsd, "max_lim_x": common_point-common_point/10, "min_lim_y": -max_len/80, "max_lim_y":
    max_len+max_len/80,
00830               "min_ax_x": common_point, "max_ax_x": max_non_init_rmsd, "min_ax_y": 0, "max_ax_y": max_len+max_len/80, "ax_step_x":
    (max_non_init_rmsd-common_point)/20, "ax_step_y": math.floor(max_len/20)}
00831
00832    if metr_units == 'contacts':
00833        extra_line = [
00834            {"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({} {})".format(metric.upper(), int(init_metr), metr_units), "col":
    "darkmagenta"},
```

```
00835             {"ax_type": 'ver', "val": min(min(elem) for elem in bsf_arr), "name": "The lowest {} metric ({} {})".format(metric.upper(),
      int(min(min(elem) for elem in bsf_arr)), metr_units), "col": "darkgreen"}]
00836     else:
00837         extra_line = [
00838             {"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metric.upper(), init_metr, metr_units), "col":
      "darkmagenta"},
00839             {"ax_type": 'ver', "val": min(min(elem) for elem in bsf_arr), "name": "The lowest {} metric ({:3.2f} {})".format(metric.upper(),
      min(min(elem) for elem in bsf_arr), metr_units), "col": "darkgreen"}]
00840     if metric == 'rmsd':
00841         extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units), "col": "midnightblue"})
00842     title = "{} | best_so_far vs steps | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00843     filename = "{}_best_so_far_vs_steps_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00844     filename = os.path.join(custom_path, filename)
00845     fig_num = single_plot(fig_num, ax_prop, bsf_arr,     non_shrink_arr, legend_names.copy(), '-',  1,   bsf=True,  rev=True,
      extra_line=extra_line, shrink=shrink, xlab="Distance to the goal, {}".format(metr_units), ylab="Steps (20 ps each)", title=title,
      filename=filename)  # best_so_far vs steps | cut
00846     for i in range(len(bsf_arr)):
00847         ff = legend_names[i].split('with')[1].split('ff')[0].strip()
00848         title = "{} | best_so_far vs steps | {} | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00849         filename = "{}_best_so_far_vs_steps_{}_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00850         extra_line[1]["val"] = min(bsf_arr[i])
00851         if metr_units == 'contacts':
00852             extra_line[1]["name"] = "The lowest {} metric ({} {})".format(metric.upper(), int(min(bsf_arr[i])), metr_units)
00853         else:
00854             extra_line[1]["name"] = "The lowest {} metric ({:3.2f} {})".format(metric.upper(), min(bsf_arr[i]), metr_units)
00855         filename = os.path.join(custom_path, filename)
00856         fig_num = single_plot(fig_num, ax_prop, [bsf_arr[i],], [non_shrink_arr[i],] if non_shrink_arr is not None else None,
      [legend_names[i],].copy(), '-', 1, bsf=True, rev=True, extra_line=extra_line, shrink=shrink,
00857                               xlab="Distance to the goal, {}".format(metr_units), ylab="Steps (20 ps each)", title=title, filename=filename)  #
      best_so_far vs steps | cut
00858
00859     return fig_num
00860
00861
```

References single_plot().
Referenced by plot_all_metrics(), and plot_only_one_metric().
Here is the call graph for this function:



Here is the caller graph for this function:



3.1.1.10 single_plot()

```
int compare_db_perf_new_format.single_plot (
        int  fig_num,
        dict  ax_prop,
        list  arr_A,
        list  arr_B,
        list  filenames_db,
        str  marker,
```

```
                float  mark_size,
                bool bsf,
                bool rev,
                bool shrink,
                str  xlab,
                str  ylab,
                str  title,
                str  filename,
                list   extra_line =  None,
                int  mdpi =  400,
                dict   second_ax =  None,
                list   sec_arr =  None )
```

Main plotting function.

```
    int  fig_num: figure number, it should not matter, since we close all figures regularly
    dict   ax_prop: axis properties
    list  arr_A: typically Y values
    list  arr_B: typically X values
    list  filenames_db: line names
    str  marker: type of the marker
    float   mark_size: size of the marker
    bool   bsf: best so far version
    bool   rev: reversed
    bool   shrink: whether to ignore x values, and just plot all y values
    str  xlab: x label
    str  ylab:  y label
    str  title: plot title
    str  filename: output filename
    list  extra_line: whether to plot extra line, if so contains its properties
    int  mdpi: plot resolution
    dict   second_ax: whether to plot second Y axis, if so this contains  dict   with properties
    list  sec_arr: value for the second axis
```

Returns

```
     :return: figure number, it should not matter, since we close all figures regularly
```

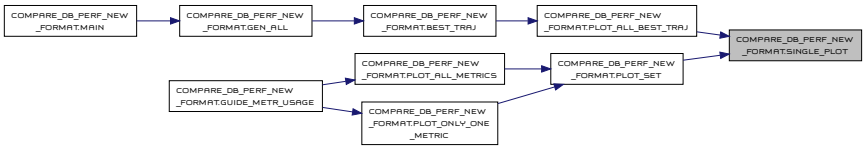
Definition at line 887 of file compare_db_perf_new_format.py.

```
00887     Returns:
00888         :return: figure number, it should not matter, since we close all figures regularly
00889     """
00890     fig_num += 1
00891     # for fname in ['angl_version_of_best_traj_angl_only_results_gromos_trp_300_2_fixed_vs_pt_energy',
00892     #  'rmsd_version_of_best_traj_rmsd_only_results_gromos_trp_300_2_fixed_vs_pt_energy',
00893     #  'rmsd_version_of_best_traj_rmsd_vs_dist',
00894     #  'xor_version_of_best_traj_rmsd_only_results_opls_trp_300_2_fixed_vs_angl',
00895     #  'xor_version_of_best_traj_rmsd_only_results_opls_trp_300_2_fixed_vs_pt_energy',
00896     #  'xor_version_of_best_traj_angl_only_results_opls_trp_300_2_fixed_vs_pt_energy',
00897     #              'rmsd_to_goal_vs_best_so_far_full_RMSD_unshrink']:
00898     #     if fname in filename:
00899     #         print('found')
00900
00901     w, h = figaspect(0.5)
00902     fig = plt.figure(fig_num, figsize=(w, h))
00903     #
00904     ax = fig.gca()
00905     fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(w, h), sharex=True, squeeze=False)
00906     plt.xlim(ax_prop["min_lim_x"], ax_prop["max_lim_x"])
00907     plt.ylim(ax_prop["min_lim_y"], ax_prop["max_lim_y"])
00908
00909     major_xticks = np.arange(ax_prop["min_ax_x"], ax_prop["max_ax_x"], ax_prop["ax_step_x"])
00910     major_yticks = np.arange(ax_prop["min_ax_y"], ax_prop["max_ax_y"], ax_prop["ax_step_y"])
00911
00912     if ax_prop["ax_step_y"] is not None:
00913         if major_yticks[-1] > ax_prop["max_lim_y"]:  # fix inconsistency in real numbers
00914             major_yticks[-1] = ax_prop["max_lim_y"]
00915         if ax_prop["max_lim_y"] - major_yticks[-1] > ax_prop["ax_step_y"]:  # this should not happen, but just in case..
00916             major_yticks = np.append(major_yticks, major_yticks[-1] + ax_prop["ax_step_y"])
00917         elif ax_prop["max_lim_y"] - major_yticks[-1] > 0.7*ax_prop["ax_step_y"]:
00918             major_yticks = np.append(major_yticks, ax_prop["max_lim_y"])
00919
00920     if ax_prop["ax_step_x"] is not None:
00921         if ax_prop["max_lim_x"] - major_xticks[-1] > ax_prop["ax_step_x"]:  # this should not happen, but just in case..
00922             print('2', filename)
00923             major_xticks = np.append(major_xticks, int(major_xticks[-1] + ax_prop["ax_step_x"]) if isinstance(ax_prop["ax_step_x"], int) else
       (major_xticks[-1] + ax_prop["ax_step_x"]))
00924         elif ax_prop["max_lim_x"] - major_xticks[-1] > 0.7 * ax_prop["ax_step_x"]:
```

```
00925             print('1', filename)
00926             major_xticks = np.append(major_xticks, int(ax_prop["max_lim_x"]) if isinstance(ax_prop["ax_step_x"], int) else
      ax_prop["max_lim_x"])
00927 
00928         if arr_B is not None and abs(arr_B[0][-1] - major_xticks[-1]) < 0.5 * ax_prop["ax_step_x"]:
00929             major_xticks[-1] = arr_B[0][-1]
00930         elif abs(max(len(elem) for elem in arr_A) - major_xticks[-1]) < 0.5 * ax_prop["ax_step_x"]:
00931             major_xticks[-1] = max(len(elem) for elem in arr_A)
00932 
00933     if major_xticks is not None:
00934         ax[0][0].set_xticks(major_xticks)
00935     if major_yticks is not None:
00936         ax[0][0].set_yticks(major_yticks)
00937     # if minor_xticks is not None:
00938     #     ax.set_xticks(minor_xticks, minor=True)
00939     # if minor_yticks is not None:
00940     #     ax.set_yticks(minor_yticks, minor=True)
00941     top_ax = ax[0][0]
00942     if second_ax is not None:
00943         ax2 = ax[0][0].twinx()
00944         major_yticks2 = np.arange(second_ax["min_ax_y"], second_ax["max_ax_y"], second_ax["ax_step_y"])
00945 
00946         if major_yticks2[-1] > second_ax["max_lim_y"]:  # fix inconsistency in real numbers
00947             major_yticks2[-1] = second_ax["max_lim_y"]
00948 
00949         if second_ax["max_lim_y"] - major_yticks2[-1] > second_ax["ax_step_y"]:
00950             major_yticks2 = np.append(major_yticks2, major_yticks2[-1] + second_ax["ax_step_y"])
00951         elif second_ax["max_lim_y"] - major_yticks2[-1] > 0.7*second_ax["ax_step_y"]:
00952             major_yticks2 = np.append(major_yticks2, second_ax["max_lim_y"])
00953 
00954         ax2.set_yticks(major_yticks2)
00955         ax2.tick_params(direction='out', length=6, width=1, grid_alpha=0.5)
00956         # ax[0].right_ax.set_ylim(second_ax["min_lim_y"], second_ax["max_lim_y"])
00957         ax2.set_ylim(second_ax["min_lim_y"], second_ax["max_lim_y"])
00958         ax2.plot(range(len(sec_arr)), sec_arr, color='r', alpha=0.75)
00959         ax2.set_ylabel(second_ax["label"] if second_ax["label"][-2] != ',' else second_ax["label"][0:-2])
00960         top_ax = ax2
00961 
00962 
00963 
00964     ax[0][0].tick_params(direction='out', length=6, width=1, grid_alpha=0.5)
00965     ax[0][0].grid(which='both', linestyle='dotted')
00966     plt.xticks(rotation=30)
00967     plt.subplots_adjust(top=0.95, bottom=0.16, left=0.09, right=0.90)
00968 
00969     lines_b = []
00970     for i, bsf_trav_to_goal in enumerate(arr_A):
00971         if not shrink:  # use provided array arr_B
00972             if rev:
00973                 line_b, = ax[0][0].plot(arr_A[i], arr_B[i], marker, markersize=mark_size, alpha=0.75)
00974             else:
00975                 line_b, = ax[0][0].plot(arr_B[i], arr_A[i], marker, markersize=mark_size, alpha=0.75)
00976         else:  # generate array from 0 to len(arr_A)
00977             if rev:
00978                 if bsf:
00979                     line_b, = ax[0][0].plot(arr_A[i], range(len(arr_A[i])), marker, markersize=mark_size, alpha=0.75)
00980                 else:
00981                     line_b, = ax[0][0].plot(arr_A[i], arr_B[i], marker, markersize=mark_size, alpha=0.75)
00982             else:
00983                 line_b, = ax[0][0].plot(range(len(arr_A[i])), arr_A[i], marker, markersize=mark_size, alpha=0.75)
00984         lines_b.append(line_b)
00985 
00986     if extra_line is not None:
00987         for el in extra_line:
00988             if el["ax_type"] == 'ver':
00989                 straight_line = ax[0][0].axvline(x=el["val"], color=el["col"], linestyle='--', alpha=0.75)  #
00990             elif el["ax_type"] == 'hor':
00991                 straight_line = ax[0][0].axhline(y=el["val"], color=el["col"], linestyle='--', alpha=0.75)
00992             else:
00993                 raise Exception('Wrong ax type')
00994             lines_b.append(straight_line)
00995             filenames_db.append(el["name"])
00996         if el["ax_type"] == 'ver':
00997             if not rev:
00998                 ax[0][0].annotate('Folding direction', xytext=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
      ax_prop["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 5 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 * ax_prop["ax_step_y"]),
      arrowprops={'arrowstyle': '->', 'lw': 1.3, 'color': 'mediumblue'}, va='center')  # -->
00999             else:
01000                 ax[0][0].annotate('Folding direction', xytext=(ax_prop["max_ax_x"] - 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
      ax_prop["ax_step_y"]), xy=(ax_prop["max_ax_x"] - 5 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 * ax_prop["ax_step_y"]),
      arrowprops={'arrowstyle': '->', 'lw': 1.3, 'color': 'mediumblue'}, va='center')  # -->
```

```
01001         else:
01002             if not rev:
01003                 if second_ax is not None:
01004                     ax2.annotate('Folding direction', xytext=(ax_prop["min_ax_x"] + 3.5 * ax_prop["ax_step_x"], second_ax["max_lim_y"] - 1 *
       second_ax["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 3.5 * ax_prop["ax_step_x"], second_ax["max_lim_y"] - 4 * second_ax["ax_step_y"]),
       arrowprops={'arrowstyle': '->', 'lw': 1.3, 'color': 'mediumblue'}, ha='center')  # <--
01005                 else:
01006                     ax[0][0].annotate('Folding direction', xytext=(ax_prop["min_ax_x"] + 3.5 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
       ax_prop["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 3.5 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 4 * ax_prop["ax_step_y"]),
       arrowprops={'arrowstyle': '->', 'lw': 1.3, 'color': 'mediumblue'}, ha='center')  # <--
01007             else:
01008                 pass # does not exist
01009                 # ax.annotate('folding direction', xytext=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
       ax_prop["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 4 * ax_prop["ax_step_y"]),
       arrowprops={'arrowstyle': '->', 'lw': 1.5, 'color': 'mediumblue'}, ha='center')  # -->
01010
01011     if second_ax is not None:
01012         lines_b.append(ax[0][0].plot([], [], marker, color='r', markersize=mark_size)[0])
01013         filenames_db.append(second_ax["line_name"])
01014
01015     ax[0][0].set_xlabel(xlab)
01016     ax[0][0].set_ylabel(ylab if ylab[-2] != ',' else ylab[0:-2])
01017     top_ax.legend(lines_b, filenames_db)
01018     plt.title(title)
01019     try:
01020         plt.savefig(filename, dpi=mdpi, transparent=True, bbox_inches='tight', pad_inches=0.02)
01021     except:
01022         plt.show()
01023     plt.close('all')
01024     return fig_num
01025
01026
```

Referenced by plot_all_best_traj(), and plot_set().
Here is the caller graph for this function:



3.2 compute_corr_between_metr Namespace Reference

Functions

- def main ()
- def myr (y, f)
- def myr_rev (y, f)
- def fill_stat_dict (filenames_db, legend_names, guide_metr)

Variables

- main_dict = dict ()
- full_dict = dict ()

3.2.1 Function Documentation

3.2.1.1 fill_stat_dict()

```
def compute_corr_between_metr.fill_stat_dict (
                filenames_db,
                legend_names,
                guide_metr )
```

Definition at line 347 of file compute_corr_between_metr.py.

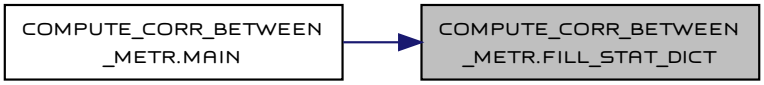
```
00347 def fill_stat_dict(filenames_db, legend_names, guide_metr):
00348     global main_dict, full_dict
00349     con_arr = [lite.connect(db_name, check_same_thread=False, isolation_level=None) for db_name in filenames_db]
00350     cur_arr = [con.cursor() for con in con_arr]
00351
00352     print('Working with ', filenames_db, ' guide metr: ', guide_metr)
00353     qry = "select a.name from main_storage a where a.{0}_goal_dist= ( select min(b.{0}_goal_dist) from main_storage b)".format(guide_metr)
00354     result_arr = [cur.execute(qry) for cur in cur_arr]
```

```
00355     fetched_one_arr = [res.fetchone() for res in result_arr]
00356     names = [all_res[0] for all_res in fetched_one_arr]
00357     spnames = [name.split('_') for name in names]
00358     all_prev_names_s = [['\'{}\"'.format('_'.join(spname[:i])) for i in range(1, len(spname)+1)] for spname in spnames]
00359     long_lines = [", ".join(all_prev_names) for all_prev_names in all_prev_names_s]
00360     qrys = ["select a.rmsd_goal_dist, a.angl_goal_dist, a.andh_goal_dist, a.and_goal_dist, a.xor_goal_dist, a.rmsd_tot_dist, a.angl_tot_dist,
      a.andh_tot_dist, a.and_tot_dist, a.xor_tot_dist, a.name, a.hashed_name from main_storage a where a.name in ( {1} ) order by
      a.id".format(guide_metr, long_line) for long_line in long_lines]
00361     result_arr = list()
00362     for i, cur in enumerate(cur_arr):
00363         result_arr.append(cur.execute(qrys[i]))
00364     fetched_all_arr = [res.fetchall() for res in result_arr]
00365 
00366     rmsd_dist_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00367     angl_dist_arr = [[dist[1] for dist in goal_dist] for goal_dist in fetched_all_arr]
00368     andh_dist_arr = [[dist[2] for dist in goal_dist] for goal_dist in fetched_all_arr]
00369     and_dist_arr = [[dist[3] for dist in goal_dist] for goal_dist in fetched_all_arr]
00370     xor_dist_arr = [[dist[4] for dist in goal_dist] for goal_dist in fetched_all_arr]
00371 
00372     goal_dist = [rmsd_dist_arr, angl_dist_arr, andh_dist_arr, and_dist_arr, xor_dist_arr]
00373     metrics = ['rmsd', 'angl', 'andh', 'and', 'xor']
00374     # metr_units = {'rmsd': 'Å', 'angl': 'n/a', 'andh': 'contacts', 'and': 'contacts', 'xor': 'contacts'}
00375 
00376     # with open('correlation.tex', 'a+') as tex_table:
00377 
00378 
00379     print('Guide metric {}'.format(guide_metr))
00380 
00381     for j in range(len(goal_dist[0])):  # iterate over dbs
00382         for i, dist_arr in enumerate(goal_dist):  # iterate over metric
00383             # tex_table.writelines(['\n\\begin{table}[h]\n', '\centering\n', '\sisetup{table-align-text-post=false}\n',
      '\\begin{tabular}{@{}|l|l|\
00384             #
          |S[table-format=3.5]|\
00385             #
          |S[table-format=3.5]|\
00386             #
          |S[table-format=3.5]|\
00387             #
          |@{}}\n']) 
00388             # tex_table.write(
00389             #     '{} & {} & {} & {} & {}  \\\\ \\hline\n'.format('{metric_x}', '{metric_y}', '{corr_xy}', '{det_xy}', '{det_yx}'))
00390 
00391             prot_name, ff = legend_names[j].split(' ')
00392             rn = None
00393             if '_' in ff:
00394                 ff, rn = ff.split('_')
00395             path_to_ener = "/home/vanya/Documents/Phillips/GMDA/Latest_results"
00396             path_to_ener1 = os.path.join(path_to_ener, prot_name)
00397             if rn is not None:
00398                 path_to_ener1 = os.path.join(path_to_ener1, "run_{}".format(rn))
00399 
00400             print('Prot: {} ff: {} run: {}'.format(prot_name, ff, rn if rn is not None else 'n/a'))
00401 
00402             # Reduced correlation matrices
00403             a = np.asarray(goal_dist[metrics.index(guide_metr)][j])
00404             a = (a - a.min()) / (a.max() - a.min())
00405             b = np.asarray(goal_dist[i][j])
00406             b = (b - b.min()) / (b.max() - b.min())
00407             main_dict[filenames_db[j]][guide_metr][metrics[i]][0] = np.corrcoef(a, b)[0][1]
00408             main_dict[filenames_db[j]][guide_metr][metrics[i]][1] = r2_score(a, b)
00409             main_dict[filenames_db[j]][guide_metr][metrics[i]][2] = r2_score(b, a)
00410 
00411             if metrics.index(guide_metr) == i:
00412                 loc_len = len(goal_dist[i][j])
00413 
00414                 path_to_ener2 = os.path.join(path_to_ener1, ff, 'PT_energy')
00415                 np_ener_file = os.path.join(path_to_ener2, '{}_correct_index_energy.npy'.format(guide_metr))
00416                 ener_arr = np.load(np_ener_file).swapaxes(0, 1)[1]
00417                 ener_arr = ener_arr[-loc_len:]  # trim, so we have same number of steps
00418                 a = np.asarray(ener_arr)
00419                 a = (a - a.min()) / (a.max() - a.min())
00420                 b = np.asarray(goal_dist[i][j])
00421                 b = (b - b.min()) / (b.max() - b.min())
00422 
00423                 if main_dict[filenames_db[j]][metrics[i]]['pt'][0] != 0:
00424                     print('warning here')
00425                 if main_dict[filenames_db[j]][metrics[i]]['pt'][1] != 0:
00426                     print('warning here')
00427                 if main_dict[filenames_db[j]][metrics[i]]['pt'][2] != 0:
00428                     print('warning here')
```

```
00429
00430                 main_dict[filenames_db[j]][guide_metr]['pt'][0] = np.corrcoef(a, b)[0][1]
00431                 main_dict[filenames_db[j]][guide_metr]['pt'][1] = r2_score(a, b)
00432                 main_dict[filenames_db[j]][guide_metr]['pt'][2] = r2_score(b, a)
00433
00434
00435             # Full correlation matrices
00436
00437             for k in range(len(goal_dist)):
00438                 # if i != k:
00439                 a = np.asarray(goal_dist[i][j])
00440                 a = (a - a.min()) / (a.max() - a.min())
00441                 b = np.asarray(goal_dist[k][j])
00442                 b = (b - b.min()) / (b.max() - b.min())
00443                 full_dict[filenames_db[j]][guide_metr][metrics[i]][metrics[k]][0] = np.corrcoef(a, b)[0][1]
00444                 full_dict[filenames_db[j]][guide_metr][metrics[i]][metrics[k]][1] = r2_score(a, b)
00445                 full_dict[filenames_db[j]][guide_metr][metrics[i]][metrics[k]][2] = r2_score(b, a)
00446
00447             loc_len = len(goal_dist[i][j])
00448
00449             path_to_ener2 = os.path.join(path_to_ener1, ff, 'PT_energy')
00450             np_ener_file = os.path.join(path_to_ener2, '{}_correct_index_energy.npy'.format(guide_metr))
00451             ener_arr = np.load(np_ener_file).swapaxes(0, 1)[1]
00452             ener_arr = ener_arr[-loc_len:]  # trim, so we have same number of steps
00453             a = np.asarray(ener_arr)
00454             a = (a - a.min()) / (a.max() - a.min())
00455             b = np.asarray(goal_dist[i][j])
00456             b = (b - b.min()) / (b.max() - b.min())
00457
00458             full_dict[filenames_db[j]][guide_metr][metrics[i]]['pt'][0] = np.corrcoef(a, b)[0][1]
00459             full_dict[filenames_db[j]][guide_metr][metrics[i]]['pt'][1] = r2_score(a, b)
00460             full_dict[filenames_db[j]][guide_metr][metrics[i]]['pt'][2] = r2_score(b, a)
00461
00462
00463
```

Referenced by main().

Here is the caller graph for this function:



3.2.1.2 main() def compute_corr_between_metr.main ( )

Definition at line 16 of file compute_corr_between_metr.py.

```
00016 def main():
00017     global main_dict, full_dict
00018     batch_arr = list()
00019
00020     # #########   TRP   ######################
00021     filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_amber_trp_300_2_fixed.sqlite3', 'results_charm_trp_300_fixed.sqlite3',
       'results_charm_trp_300_2_fixed.sqlite3', 'results_gromos_trp_300_fixed.sqlite3', 'results_gromos_trp_300_2_fixed.sqlite3',
       'results_opls_trp_300_fixed.sqlite3', 'results_opls_trp_300_2_fixed.sqlite3']
00022     legend_names = ['TRP amber_1', 'TRP amber_2', 'TRP charm_1', 'TRP charm_2', 'TRP gromos_1', 'TRP gromos_2', 'TRP opls_1', 'TRP opls_2']
00023     common_path = '../trp_all_compar'
00024     batch_arr.append((filenames_db, legend_names, common_path))
00025     for fname in filenames_db:
00026         main_dict[fname] = dict ()
00027         for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00028             main_dict[fname][metr] = {'rmsd': [0, 0, 0], 'angl': [0, 0, 0], 'andh': [0, 0, 0], 'and': [0, 0, 0], 'xor': [0, 0, 0], 'pt': [0, 0,
       0]}
00029         full_dict[fname] = dict ()
00030         for g_metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00031             full_dict[fname][g_metr] = dict ()
00032             for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00033                 full_dict[fname][g_metr][metr] = {'rmsd': [0, 0, 0], 'angl': [0, 0, 0], 'andh': [0, 0, 0], 'and': [0, 0, 0], 'xor': [0, 0, 0],
       'pt': [0, 0, 0]}
00034
00035
```

```
00036     # # ##################  VIL  ######################
00037
00038     filenames_db = ['results_amber_vil_300.sqlite3', 'results_charm_vil_300.sqlite3', 'results_gromos_vil_300.sqlite3',
      'results_opls_vil_300.sqlite3']
00039     legend_names = ['VIL amber', 'VIL charm', 'VIL gromos', 'VIL opls']
00040     common_path = '../vil_all_compar'
00041     batch_arr.append((filenames_db, legend_names, common_path))
00042     for fname in filenames_db:
00043         main_dict[fname] = dict ()
00044         for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00045             main_dict[fname][metr] = {'rmsd': [0, 0, 0], 'angl': [0, 0, 0], 'andh': [0, 0, 0], 'and': [0, 0, 0], 'xor': [0, 0, 0], 'pt': [0, 0,
      0]}
00046         full_dict[fname] = dict ()
00047         for g_metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00048             full_dict[fname][g_metr] = dict ()
00049             for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00050                 full_dict[fname][g_metr][metr] = {'rmsd': [0, 0, 0], 'angl': [0, 0, 0], 'andh': [0, 0, 0], 'and': [0, 0, 0], 'xor': [0, 0, 0],
      'pt': [0, 0, 0]}
00051
00052
00053     # # ##################  GB1  ######################
00054     # #
00055     filenames_db = ['results_amber_gb1_300.sqlite3', 'results_charm_gb1_300.sqlite3', 'results_gromos_gb1_300.sqlite3',
      'results_opls_gb1_300.sqlite3']
00056     legend_names = ['GB1 amber', 'GB1 charm', 'GB1 gromos', 'GB1 opls']
00057     common_path = '../gb1_all_compar'
00058     batch_arr.append((filenames_db, legend_names, common_path))
00059     for fname in filenames_db:
00060         main_dict[fname] = dict ()
00061         for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00062             main_dict[fname][metr] = {'rmsd': [0, 0, 0], 'angl': [0, 0, 0], 'andh': [0, 0, 0], 'and': [0, 0, 0], 'xor': [0, 0, 0], 'pt': [0, 0,
      0]}
00063         full_dict[fname] = dict ()
00064         for g_metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00065             full_dict[fname][g_metr] = dict ()
00066             for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00067                 full_dict[fname][g_metr][metr] = {'rmsd': [0, 0, 0], 'angl': [0, 0, 0], 'andh': [0, 0, 0], 'and': [0, 0, 0], 'xor': [0, 0, 0],
      'pt': [0, 0, 0]}
00068
00069
00070
00071     for filenames_db, legend_names, common_path in batch_arr:
00072         for guide_metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00073             fill_stat_dict(filenames_db, legend_names, guide_metr)
00074
00075     with open('correlation.tex', 'w') as tex_table:
00076         # for db_name in main_dict.keys():
00077         #     tex_table.writelines(['\n\\begin{table}[h]\n', '\centering\n', '\sisetup{table-align-text-post=false}\n',
00078         #                           '\\begin{tabular}{@{}|l\
00079         #                           |S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]\
00080         #                           |S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]\
00081         #                           |S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]\
00082         #                           |S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]\
00083         #                           |S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]\
00084         #                           |S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]\
00085         #                           |@{}}\n \\hline\n'])
00086         #     tex_table.write(' \multirow{2}{*}{metric\_y} & \\multicolumn{3}{c@{}|}{rmsd} & \\multicolumn{3}{c@{}|}{angl} &
      \\multicolumn{3}{c@{}|}{andh} & \\multicolumn{3}{c@{}|}{and} & \\multicolumn{3}{c@{}|}{xor} & \\multicolumn{3}{c@{}|}{pot ener} \\\\
      \\cline{2-19}\n')
00087         #     tex_table.write(' & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} \\\\
      \\hline\n'.format('{cor\_xy}', '{d\_xy}', '{d\_yx}', '{cor\_xy}', '{d\_xy}', '{d\_yx}', '{cor\_xy}', '{d\_xy}', '{d\_yx}', '{cor\_xy}',
      '{d\_xy}', '{d\_yx}', '{cor\_xy}', '{d\_xy}', '{d\_yx}', '{cor\_xy}', '{d\_xy}', '{d\_yx}'))
00088         #     for gm in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00089         #                         if gm == 'andh':
00090         #                             tw = 'and_h'
00091         #                 else:
00092         #                     tw = gm
00093         #                 tex_table.write('{} '.format(tw.upper()))
00094         #         for chm in ['rmsd', 'angl', 'andh', 'and', 'xor', 'pt']:
00095         #             val1 = main_dict[db_name][gm][chm][0]
00096         #             val2 = main_dict[db_name][gm][chm][1]
00097         #             val3 = main_dict[db_name][gm][chm][2]
00098         #             if abs(val1) > 99.999:
00099         #                 tex_table.write(' & {{$<-99$}} ')
00100         #             elif abs(val1) > 10.0:
00101         #                 tex_table.write(' & {{${}$}} '.format(int(round(val1))))
00102         #             else:
00103         #                 tex_table.write(' & {:3.2f} '.format(val1))
00104         #
00105         #             if abs(val2) > 99.999:
00106         #                 tex_table.write(' & {{$<-99$}} ')
```

```
00107         #             elif abs(val2) > 10.0:
00108         #                 tex_table.write(' & {{${}$}} '.format(int(round(val2))))
00109         #             else:
00110         #                 tex_table.write(' & {:3.2f} '.format(val2))
00111         #
00112         #             if abs(val3) > 99.999:
00113         #                 tex_table.write(' & {{$<-99$}} ')
00114         #             elif abs(val3) > 10.0:
00115         #                 tex_table.write(' & {{${}$}} '.format(int(round(val3))))
00116         #             else:
00117         #                 tex_table.write(' & {:3.2f} '.format(val3))
00118         #             # tex_table.write(' & {:3.2f} & {:3.2f} & {:3.2f} '.format(main_dict[db_name][gm][chm][0],
      main_dict[db_name][gm][chm][1], main_dict[db_name][gm][chm][2]))
00119         #         tex_table.write('\\\\ \\hline\n')
00120         # tex_table.writelines(['\\end {tabular}\n', '\\caption{{{}}}\n'.format('DB: {}'.format(db_name.translate(str.maketrans({"_":
      r"\_"})))), '\\end {table}\n'])
00121         # tex_table.write('\n\n\n')
00122
00123
00124 #         ############ CORR ONLY  ############################
00125
00126         for db_name in main_dict.keys():
00127             tex_table.writelines(['\n\\begin{table}[t]\n', '\sisetup{table-align-text-post=false}\n',
00128                                   '\\begin{tabular}{@{}|l|S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]|S[table-format=2.2]
      |S[table-format=2.2] |S[table-format=2.2]|@{}}\n\\rowcolor{lightgray}\n'])
00129             tex_table.write(' {} & {\\glsentryshort{rmsd}} & {\\glsentryshort{angl}} & {\\glsentryshort{andh}} & {\\glsentryshort{and}} &
      {\\glsentryshort{xor}} & {Potential energy} \\\\ \\hline \n')
00130             # tex_table.write(' {} & {RMSD} & {ANGL} & {AND\_H} & {AND} & {XOR} & {Potential energy} \\\\ \\hline \n')
00131             # tex_table.write(' & {} & {} & {} & {} & {} & {}  \\\\ \\hline\n'.format('{cor\_xy}', '{cor\_xy}', '{cor\_xy}', '{cor\_xy}',
      '{cor\_xy}', '{cor\_xy}'))
00132             for gm in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00133                 if gm == 'andh':
00134                     tw = '\\glsentryshort{andh}'
00135                 else:
00136                     tw = '\\glsentryshort{{{}}}'.format(gm)
00137                 tex_table.write('{} '.format(tw))
00138                 for chm in ['rmsd', 'angl', 'andh', 'and', 'xor', 'pt']:
00139                     val1 = main_dict[db_name][gm][chm][0]
00140                     if abs(val1) > 99.999:
00141                         tex_table.write(' & {{$<-99$}} ')
00142                     elif abs(val1) > 10.0:
00143                         tex_table.write(' & {{${}$}} '.format(int(round(val1))))
00144                     else:
00145                         tex_table.write(' & {:3.2f} '.format(val1))
00146
00147                 tex_table.write('\\\\ \\hline\n')
00148             db_name1 = db_name.split('.')[0]
00149             pr_1 = db_name1.split('_')[2]
00150             ff_2 = db_name1.split('_')[1]
00151             if pr_1 == 'trp':
00152                 if '2' in db_name:
00153                     tex_table.writelines(['\\end {tabular}\n', '\\label {{cor_{}}}\n'.format(db_name1),
00154                                           '\\caption{{{}}}\n'.format(
00155                                               'orrelation coefficients among metrics and potential energy for the second simulation of
      \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field. Rows simultaneously represent the best trajectory according to the
      listed metric and correlation between this metric and other metrics and potential energy.'.format(
00156                                                   pr_1, ff_2)), '\\end {table}\n'])
00157                 else:
00158                     tex_table.writelines(['\\end {tabular}\n', '\\label {{cor_{}}}\n'.format(db_name1),
00159                                           '\\caption{{{}}}\n'.format(
00160                                               'orrelation coefficients among metrics and potential energy for the first simulation of
      \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field. Rows simultaneously represent the best trajectory according to the
      listed metric and correlation between this metric and other metrics and potential energy.'.format(
00161                                                   pr_1, ff_2)), '\\end {table}\n'])
00162             else:
00163                 tex_table.writelines(['\\end {tabular}\n', '\\label {{cor_{}}}\n'.format(db_name1),
00164                                       '\\caption{{{}}}\n'.format(
00165                     'orrelation coefficients among metrics and potential energy for simulation of \\glsentryshort{{{}}} protein with
      \\glsentryshort{{{}}} force field. Rows simultaneously represent the best trajectory according to the listed metric and correlation between
      this metric and other metrics and potential energy.'.format(pr_1, ff_2)), '\\end {table}\n'])
00166
00167             tex_table.write('\n\n\n')
00168
00169
00170
00171 #         ##################### DET ONLY  ############################
00172         tex_table.write('\\begin{landscape}')
00173         for db_name in main_dict.keys():
00174             tex_table.writelines(['\n\\begin{table}\n', '\sisetup{table-align-text-post=false}\n',
00175                                   '\\begin{tabular}{@{}|l|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2]
      |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2]
```

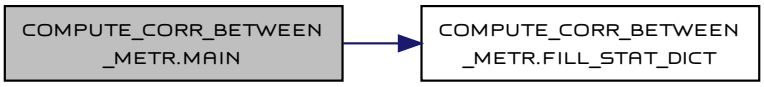
```
|S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2]
|S[table-format=3.2] |S[table-format=3.2]|@{}}\n\\rowcolor{lightgray}\n'])
00176           tex_table.write('\multirow{2}{*}{} & \multicolumn{2}{c@{}|}{\glsentryshort{rmsd}} & \multicolumn{2}{c@{}|}{\glsentryshort{angl}} &
\multicolumn{2}{c@{}|}{\glsentryshort{andh}} & \multicolumn{2}{c@{}|}{\glsentryshort{and}} & \multicolumn{2}{c@{}|}{\glsentryshort{xor}} &
\multicolumn{2}{c@{}|}{Potential energy} \\\\ \cline{2-13}\n')
00177           tex_table.write(' & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} \\\\ \hline\n'.format('{$r^2_{xy}$}',
'{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}',
'{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}'))
00178           for gm in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00179               if gm == 'andh':
00180                   tw = '\\glsentryshort{andh}'
00181               else:
00182                   tw = '\\glsentryshort{{{}}}'.format(gm)
00183               tex_table.write('{} '.format(tw))
00184               for chm in ['rmsd', 'angl', 'andh', 'and', 'xor', 'pt']:
00185                   val2 = main_dict[db_name][gm][chm][1]
00186                   val3 = main_dict[db_name][gm][chm][2]
00187
00188                   if abs(val2) > 99.999:
00189                       tex_table.write(' & {{$<-99$}} ')
00190                   elif abs(val2) > 10.0:
00191                       tex_table.write(' & {{${}$}} '.format(int(round(val2))))
00192                   else:
00193                       tex_table.write(' & {:3.2f} '.format(val2))
00194
00195                   if abs(val3) > 99.999:
00196                       tex_table.write(' & {{$<-99$}} ')
00197                   elif abs(val3) > 10.0:
00198                       tex_table.write(' & {{${}$}} '.format(int(round(val3))))
00199                   else:
00200                       tex_table.write(' & {:3.2f} '.format(val3))
00201                   # tex_table.write(' & {:3.2f} & {:3.2f} & {:3.2f} '.format(main_dict[db_name][gm][chm][0], main_dict[db_name][gm][chm][1],
main_dict[db_name][gm][chm][2]))
00202               tex_table.write('\\\\ \\hline\n')
00203
00204           db_name1 = db_name.split('.')[0]
00205           pr_1 = db_name1.split('_')[2]
00206           ff_2 = db_name1.split('_')[1]
00207           if pr_1 == 'trp':
00208               if '2' in db_name:
00209                   tex_table.writelines(['\\end {tabular}\n', '\\label {{det_{}}}\n'.format(db_name1),
00210                                         '\\caption{{{}}}\n'.format(
00211                                             'Determination coefficients among metrics and potential energy for the second simulation of
\\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field. Rows simultaneously represent the best trajectory according to the
listed metric and determination between this metric and other metrics and potential energy.'.format(
00212                                                 pr_1, ff_2)), '\\end {table}\n'])
00213               else:
00214                   tex_table.writelines(['\\end {tabular}\n', '\\label {{det_{}}}\n'.format(db_name1),
00215                                         '\\caption{{{}}}\n'.format(
00216                                             'Determination coefficients among metrics and potential energy for the first simulation of
\\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field. Rows simultaneously represent the best trajectory according to the
listed metric and determination between this metric and other metrics and potential energy.'.format(
00217                                                 pr_1, ff_2)), '\\end {table}\n'])
00218           else:
00219               tex_table.writelines(['\\end {tabular}\n', '\\label {{det_{}}}\n'.format(db_name1),
00220                                     '\\caption{{{}}}\n'.format(
00221                   'Determination coefficients among metrics and potential energy for simulation of \\glsentryshort{{{}}} protein with
\\glsentryshort{{{}}} force field. Rows simultaneously represent the best trajectory according to the listed metric and determination between
this metric and other metrics and potential energy.'.format(pr_1, ff_2)), '\\end {table}\n'])
00222           tex_table.write('\n\n\n')
00223       tex_table.write('\\end{landscape}')
00224
00225   with open('full_correlation.tex', 'w') as tex_table:
00226
00227       #         ############ CORR ONLY  ############################
00228
00229       for db_name in main_dict.keys():
00230           for guid_m in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00231               tex_table.writelines(['\n\\begin{table}[t]\n', '\sisetup{table-align-text-post=false}\n',
00232                                     '\\begin{tabular}{@{}|l|S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]|S[table-format=2.2]
|S[table-format=2.2] |S[table-format=2.2]|@{}}\n\\rowcolor{lightgray}\n'])
00233               tex_table.write(' {} & {\\glsentryshort{rmsd}} & {\\glsentryshort{angl}} & {\\glsentryshort{andh}} & {\\glsentryshort{and}} &
{\\glsentryshort{xor}} & {Potential energy} \\\\ \\hline \n')
00234               # tex_table.write(' {} & {RMSD} & {ANGL} & {AND\_H} & {AND} & {XOR} & {Potential energy} \\\\ \\hline \n')
00235               # tex_table.write(' & {} & {} & {} & {} & {} & {}  \\\\ \\hline\n'.format('{cor\_xy}', '{cor\_xy}', '{cor\_xy}',
'{cor\_xy}', '{cor\_xy}', '{cor\_xy}'))
00236               for gm in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00237                   if gm == 'andh':
00238                       tw = '\\glsentryshort{andh}'
00239                   else:
00240                       tw = '\\glsentryshort{{{}}}'.format(gm)
```

```
00241                     tex_table.write('{} '.format(tw))
00242                     for chm in ['rmsd', 'angl', 'andh', 'and', 'xor', 'pt']:
00243                         try:
00244                             val1 = full_dict[db_name][guid_m][gm][chm][0]
00245                         except:
00246                             a = 8
00247                         if abs(val1) > 99.999:
00248                             tex_table.write(' & {{$<-99$}} ')
00249                         elif abs(val1) > 10.0:
00250                             tex_table.write(' & {{${}$}} '.format(int(round(val1))))
00251                         else:
00252                             tex_table.write(' & {:3.2f} '.format(val1))
00253 
00254                     tex_table.write('\\\\ \\hline\n')
00255                 db_name1 = db_name.split('.')[0]
00256                 pr_1 = db_name1.split('_')[2]
00257                 ff_2 = db_name1.split('_')[1]
00258                 if pr_1 == 'trp':
00259                     if '2' in db_name:
00260                         tex_table.writelines(['\\end {tabular}\n', '\\label {{cor_{}_{}}}\n'.format(guid_m, db_name1),
00261                                               '\\caption{{{}}}\n'.format(
00262                                                   'Correlation coefficients among metrics and potential energy for the second simulation of \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field for \\glsentryshort{{{}}} guide metric.'.format(
00263                                                       pr_1, ff_2, guid_m)), '\\end {table}\n'])
00264                     else:
00265                         tex_table.writelines(['\\end {tabular}\n', '\\label {{cor_{}_{}}}\n'.format(guid_m, db_name1),
00266                                               '\\caption{{{}}}\n'.format(
00267                                                   'Correlation coefficients among metrics and potential energy for the first simulation of \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field for \\glsentryshort{{{}}} guide metric.'.format(
00268                                                       pr_1, ff_2, guid_m)), '\\end {table}\n'])
00269                 else:
00270                     tex_table.writelines(['\\end {tabular}\n', '\\label {{cor_{}_{}}}\n'.format(guid_m, db_name1),
00271                                           '\\caption{{{}}}\n'.format(
00272                                               'Correlation coefficients among metrics and potential energy for simulation of \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field for \\glsentryshort{{{}}} guide metric.'.format(
00273                                                   pr_1, ff_2, guid_m)), '\\end {table}\n'])
00274                 tex_table.write('\n\n\n')
00275 
00276         #         ################### DET ONLY  ############################
00277         tex_table.write('\\begin{landscape}')
00278         for db_name in main_dict.keys():
00279             for guid_m in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00280                 tex_table.writelines(['\n\\begin{table}\n', '\sisetup{table-align-text-post=false}\n',
00281                                       '\\begin{tabular}{@{}|l|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|@{}}\n\\rowcolor{lightgray}\n'])
00282                 tex_table.write('\multirow{2}{*}{} & \multicolumn{2}{c@{}|}{\glsentryshort{rmsd}} & \multicolumn{2}{c@{}|}{\glsentryshort{angl}} & \multicolumn{2}{c@{}|}{\glsentryshort{andh}} & \multicolumn{2}{c@{}|}{\glsentryshort{and}} & \multicolumn{2}{c@{}|}{\glsentryshort{xor}} & \multicolumn{2}{c@{}|}{Potential energy} \\\\ \cline{2-13}\n')
00283                 tex_table.write(' & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} \\\\ \\hline\n'.format('{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}'))
00284                 for gm in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00285                     if gm == 'andh':
00286                         tw = '\\glsentryshort{andh}'
00287                     else:
00288                         tw = '\\glsentryshort{{{}}}'.format(gm)
00289                     tex_table.write('{} '.format(tw))
00290                     for chm in ['rmsd', 'angl', 'andh', 'and', 'xor', 'pt']:
00291                         val2 = full_dict[db_name][guid_m][gm][chm][1]
00292                         val3 = full_dict[db_name][guid_m][gm][chm][2]
00293 
00294                         if abs(val2) > 99.999:
00295                             tex_table.write(' & {{$<-99$}} ')
00296                         elif abs(val2) > 10.0:
00297                             tex_table.write(' & {{${}$}} '.format(int(round(val2))))
00298                         else:
00299                             tex_table.write(' & {:3.2f} '.format(val2))
00300 
00301                         if abs(val3) > 99.999:
00302                             tex_table.write(' & {{$<-99$}} ')
00303                         elif abs(val3) > 10.0:
00304                             tex_table.write(' & {{${}$}} '.format(int(round(val3))))
00305                         else:
00306                             tex_table.write(' & {:3.2f} '.format(val3))
00307                         # tex_table.write(' & {:3.2f} & {:3.2f} & {:3.2f} '.format(main_dict[db_name][gm][chm][0], main_dict[db_name][gm][chm][1], main_dict[db_name][gm][chm][2]))
00308                     tex_table.write('\\\\ \\hline\n')
00309 
00310                 db_name1 = db_name.split('.')[0]
```

```
00311                 pr_1 = db_name1.split('_')[2]
00312                 ff_2 = db_name1.split('_')[1]
00313                 if pr_1 == 'trp':
00314                     if '2' in db_name:
00315                         tex_table.writelines(['\\end {tabular}\n', '\\label {{det_{}_{}}}\n'.format(guid_m, db_name1),
00316                                               '\\caption{{{}}}\n'.format(
00317                                                   'Determination coefficients among metrics and potential energy for the second simulation of
      \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field for \\glsentryshort{{{}}} guide metric.'.format(
00318                                                       pr_1, ff_2, guid_m)), '\\end {table}\n'])
00319                     else:
00320                         tex_table.writelines(['\\end {tabular}\n', '\\label {{det_{}_{}}}\n'.format(guid_m, db_name1),
00321                                               '\\caption{{{}}}\n'.format(
00322                                                   'Determination coefficients among metrics and potential energy for the first simulation of
      \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field for \\glsentryshort{{{}}} guide metric.'.format(
00323                                                       pr_1, ff_2, guid_m)), '\\end {table}\n'])
00324                 else:
00325                     tex_table.writelines(['\\end {tabular}\n', '\\label {{det_{}_{}}}\n'.format(guid_m, db_name1),
00326                                           '\\caption{{{}}}\n'.format(
00327                                               'Determination coefficients among metrics and potential energy for simulation of
      \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field for \\glsentryshort{{{}}} guide metric.'.format(
00328                                                   pr_1, ff_2, guid_m)), '\\end {table}\n'])
00329                 tex_table.write('\n\n\n')
00330         tex_table.write('\\end{landscape}')
00331
00332
00333
00334
```

References fill_stat_dict().

Here is the call graph for this function:



3.2.1.3 myr() def compute_corr_between_metr.myr (y, f)

Definition at line 335 of file compute_corr_between_metr.py.

```
00335 def myr(y, f):
00336     SSres = sum(map(lambda x: (x[0] - x[1]) ** 2, zip(y, f)))
00337     SStot = sum([(x - np.mean(y)) ** 2 for x in y])
00338     return 1-(SSres/SStot)
00339
00340
```

3.2.1.4 myr_rev() def compute_corr_between_metr.myr_rev (y, f)

Definition at line 341 of file compute_corr_between_metr.py.

```
00341 def myr_rev(y, f):
00342     SSres = sum(map(lambda x: (x[0] - x[1]) ** 2, zip(y, f)))
00343     SStot = sum([(x - np.mean(f)) ** 2 for x in f])
00344     return 1-(SSres/SStot)
00345
00346
```

3.2.2 Variable Documentation

3.2.2.1 full_dict compute_corr_between_metr.full_dict = dict ()

Definition at line 14 of file compute_corr_between_metr.py.

3.2.2.2 main_dict compute_corr_between_metr.main_dict = dict ()

Definition at line 13 of file compute_corr_between_metr.py.

3.3 compute_sincos_dist Namespace Reference

Functions

- def compute_sincos_dist (num_el, filename_nat='sincos_goal.dat', filename_check='sincos_bb_300.dat')

3.3.1 Function Documentation

3.3.1.1 compute_sincos_dist() def compute_sincos_dist.compute_sincos_dist (num_el, filename_nat = 'sincos_goal.dat', filename_check = 'sincos_bb_300.dat')

Definition at line 8 of file compute_sincos_dist.py.

```
00008 def compute_sincos_dist(num_el, filename_nat = 'sincos_goal.dat', filename_check = 'sincos_bb_300.dat'):
00009     with open(filename_nat, 'rb') as file:
00010         initial_1d_array = np.frombuffer(file.read(), dtype=np.float64 , count=-1)
00011     nat_arr = np.reshape(initial_1d_array, (-1, num_el*2))
00012     with open(filename_check, 'rb') as file:
00013         initial_1d_array = np.frombuffer(file.read(), dtype=np.float64 , count=-1)
00014     check_arr = np.reshape(initial_1d_array, (-1, num_el*2))
00015     del initial_1d_array
00016 
00017     res_arr = [None]*check_arr.shape[0]
00018     for i in range(check_arr.shape[0]):
00019         res_arr[i] = np.sum(abs(check_arr[i] - nat_arr))
00020     # res_arr = [res_arr[i*2] + res_arr[i*2+1] for i in range(len(res_arr)/2)]
00021 
00022     max_val = max(res_arr)
00023     min_val = min(res_arr)
00024     fig_num = 0
00025     mdpi = 400
00026     major_xticks = None
00027     minor_xticks = None
00028     major_yticks = None
00029     minor_yticks = None
00030     w, h = figaspect(0.5)
00031     fig = plt.figure(fig_num, figsize=(w, h))
00032     plt.xlim(0, len(res_arr))
00033     ax = fig.gca()
00034     major_xticks = np.arange(0, len(res_arr) + len(res_arr) / 10, len(res_arr) / 10)
00035     major_yticks = np.arange(min_val, max_val + max_val / 16, (max_val - min_val) / 16)
00036     if major_xticks is not None:
00037         ax.set_xticks(major_xticks)
00038     if minor_xticks is not None:
00039         ax.set_xticks(minor_xticks, minor=True)
00040     if major_yticks is not None:
00041         ax.set_yticks(major_yticks)
00042     if minor_yticks is not None:
00043         ax.set_yticks(minor_yticks, minor=True)
00044     plt.grid(which='both')
00045     lines = []
00046 
00047     line, = plt.plot(range(len(res_arr)), res_arr, '-', markersize=1)
00048     lines.append(line)
00049     ax.legend(lines, 'full cont')
00050     plt.xlabel("frame")
00051     plt.ylabel("sin/cos")
00052     plt.title('sin/cos (difference, error) for 20ns gb1 simulatoin and goal at 300K (lower is better)')
00053     plt.savefig('sincos_20ns_300.png', dpi=mdpi)
00054 
00055 compute_sincos_dist(110, 'sincos_goal.dat')
```

3.4 concat_all_xtc Namespace Reference

Functions

- def get_all_xtc (past_dir)

Variables

- int elem_at_once = 128
- def all_xtc = get_all_xtc('./past/')
- int tot_iter = 0
- int cur_name = 0
- new_names = list()
- def cur_files = all_xtc[tot_iter:tot_iter+elem_at_once]
- f
- o

- n
- new_names1 = list()
- new_names2 = list()
- new_names3 = list()

3.4.1 Function Documentation

3.4.1.1 get_all_xtc() def concat_all_xtc.get_all_xtc (past_dir)

Definition at line 6 of file concat_all_xtc.py.

```
00006 def get_all_xtc(past_dir):
00007     filenames_found = [f.split("/")[-1] for f in os.listdir(past_dir)]
00008     filenames_found_important = [f for f in filenames_found if f.split('.')[1] == 'xtc']
00009     del filenames_found
00010     print('Found files: {} with .xtc'.format(len(filenames_found_important)))
00011     return filenames_found_important
00012
```

3.4.2 Variable Documentation

3.4.2.1 all_xtc def concat_all_xtc.all_xtc = get_all_xtc('./past/')

Definition at line 15 of file concat_all_xtc.py.

3.4.2.2 cur_files concat_all_xtc.cur_files = all_xtc[tot_iter:tot_iter+elem_at_once]

Definition at line 25 of file concat_all_xtc.py.

3.4.2.3 cur_name int concat_all_xtc.cur_name = 0

Definition at line 22 of file concat_all_xtc.py.

3.4.2.4 elem_at_once int concat_all_xtc.elem_at_once = 128

Definition at line 13 of file concat_all_xtc.py.

3.4.2.5 f concat_all_xtc.f

Definition at line 28 of file concat_all_xtc.py.

3.4.2.6 n concat_all_xtc.n

Definition at line 28 of file concat_all_xtc.py.

3.4.2.7 new_names concat_all_xtc.new_names = list()

Definition at line 23 of file concat_all_xtc.py.

3.4.2.8 new_names1 concat_all_xtc.new_names1 = list()

Definition at line 34 of file concat_all_xtc.py.

3.4.2.9 new_names2 concat_all_xtc.new_names2 = list()

Definition at line 51 of file concat_all_xtc.py.

3.4.2.10 new_names3 concat_all_xtc.new_names3 = list()

Definition at line 68 of file concat_all_xtc.py.

3.4.2.11 o concat_all_xtc.o

Definition at line 28 of file concat_all_xtc.py.

3.4.2.12 tot_iter int concat_all_xtc.tot_iter = 0

Definition at line 21 of file concat_all_xtc.py.

3.5 convert_bad_db Namespace Reference

Functions

- def get_db_con (db_name='fixed_db', tot_seeds=4)

Variables

- string in_db = "results_opls_trp_300"
- con_bad = lite.connect(in_db+'.sqlite3', check_same_thread=False, isolation_level=None)
- cur_bad = con_bad.cursor()
- string qry = "SELECT rmsd_goal_dist, rmsd_prev_dist, rmsd_tot_dist, angl_goal_dist, angl_prev_dist, angl_tot_dist, andh_goal_dist," \
- res = cur_bad.execute(qry)
- res_first = res.fetchone()
- res_arr = res.fetchall()
- log_res = res.fetchall()
- vis_res = res.fetchall()
- good_arr = list()
- elem = res_first
- def con_fixed = get_db_con(in_db+'_fixed')
- def cur_good = con_fixed.cursor()

3.5.1 Function Documentation

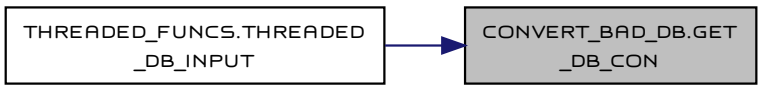
3.5.1.1 get_db_con() def convert_bad_db.get_db_con (db_name = 'fixed_db', tot_seeds = 4)

Definition at line 11 of file convert_bad_db.py.

```
00011 def get_db_con(db_name ='fixed_db', tot_seeds=4):
00012     counter = 0
00013     # db_path = '/dev/shm/GMDApy'
00014     db_path = os.getcwd()
00015     full_path = os.path.join(db_path, db_name + '.sqlite3')
00016 
00017     con = lite.connect(full_path, check_same_thread=False, isolation_level=None)
00018 
00019     cur = con.cursor()
00020     cur.execute("""CREATE TABLE main_storage (
00021         id               INTEGER   PRIMARY KEY AUTOINCREMENT,
00022 
00023         rmsd_goal_dist   FLOAT    NOT NULL,
00024         rmsd_prev_dist   FLOAT    NOT NULL,
00025         rmsd_tot_dist    FLOAT    NOT NULL,
00026 
00027         angl_goal_dist   FLOAT    NOT NULL,
00028         angl_prev_dist   FLOAT    NOT NULL,
00029         angl_tot_dist    FLOAT    NOT NULL,
00030 
00031         andh_goal_dist   INTEGER   NOT NULL,
00032         andh_prev_dist   INTEGER   NOT NULL,
00033         andh_tot_dist    INTEGER   NOT NULL,
00034 
00035         and_goal_dist    INTEGER   NOT NULL,
00036         and_prev_dist    INTEGER   NOT NULL,
00037         and_tot_dist     INTEGER   NOT NULL,
00038 
00039         xor_goal_dist    INTEGER   NOT NULL,
00040         xor_prev_dist    INTEGER   NOT NULL,
00041         xor_tot_dist     INTEGER   NOT NULL,
00042 
00043         curr_gc          INTEGER   NOT NULL,
00044         Timestamp        DATETIME DEFAULT (CURRENT_TIMESTAMP),
00045         hashed_name      CHAR (32) NOT NULL UNIQUE,
00046         name             TEXT
00047         );""")
00048     con.commit()
00049     cur.execute("""CREATE TABLE visited (
00050         vid        INTEGER   PRIMARY KEY AUTOINCREMENT, \
00051         id        REFERENCES main_storage (id),
00052         cur_gc    INTEGER,
00053         Timestamp DATETIME DEFAULT (CURRENT_TIMESTAMP)
00054     );""")
00055     con.commit()
00056 
00057     add_ind_q = 'CREATE INDEX viz_id_idx ON visited (id);'
00058     cur.execute(add_ind_q)
00059     con.commit()
00060 
00061     # id         REFERENCES main_storage (id), \
00062     init_query = 'CREATE TABLE log ( \
00063         lid        INTEGER   PRIMARY KEY AUTOINCREMENT, \
00064         operation  INTEGER, \
00065         id         INTEGER, \
00066         src        CHAR (8), \
```

```
00067         dst        CHAR(8), \
00068         cur_metr   CHAR(5), \
00069         gc         INTEGER , \
00070         mul        FLOAT, \
00071         bsfr        FLOAT, \
00072         bsfn        FLOAT, \
00073         bsfh        FLOAT, \
00074         bsfa        FLOAT, \
00075         bsfx        FLOAT, \
00076         Timestamp  DATETIME DEFAULT (CURRENT_TIMESTAMP)'  # no this is not an error
00077     for i in range(tot_seeds):
00078         init_query += ", \
00079         dist_from_prev_{0} FLOAT, \
00080         dist_to_goal_{0}   FLOAT ".format(i+1)
00081     init_query += ');'
00082 
00083     cur.execute(init_query)
00084     con.commit()
00085     add_ind_q = 'CREATE INDEX log_id_idx ON log (id);'
00086     cur.execute(add_ind_q)
00087     con.commit()
00088 
00089     cur.execute('PRAGMA mmap_size=-64000')  # 32M
00090     cur.execute('PRAGMA journal_mode =  OFF')
00091     cur.execute('PRAGMA synchronous = OFF')
00092     cur.execute('PRAGMA temp_store = MEMORY')
00093     cur.execute('PRAGMA threads = 32')
00094 
00095     return con
00096 
```

Referenced by threaded_funcs.threaded_db_input().
Here is the caller graph for this function:



3.5.2 Variable Documentation

3.5.2.1 con_bad

convert_bad_db.con_bad = lite.connect(in_db+'.sqlite3', check_same_thread=False, isolation_level=None)
Definition at line 99 of file convert_bad_db.py.

3.5.2.2 con_fixed

def convert_bad_db.con_fixed = get_db_con(in_db+'_fixed')
Definition at line 137 of file convert_bad_db.py.

3.5.2.3 cur_bad

convert_bad_db.cur_bad = con_bad.cursor()
Definition at line 102 of file convert_bad_db.py.

3.5.2.4 cur_good

def convert_bad_db.cur_good = con_fixed.cursor()
Definition at line 138 of file convert_bad_db.py.

3.5.2.5 elem

convert_bad_db.elem = res_first
Definition at line 122 of file convert_bad_db.py.

3.5.2.6 good_arr

convert_bad_db.good_arr = list()
Definition at line 121 of file convert_bad_db.py.

3.5.2.7 in_db

string convert_bad_db.in_db = "results_opls_trp_300"
Definition at line 97 of file convert_bad_db.py.

3.5.2.8 log_res

convert_bad_db.log_res = res.fetchall()
Definition at line 114 of file convert_bad_db.py.

3.5.2.9 qry

string convert_bad_db.qry = "SELECT rmsd_goal_dist, rmsd_prev_dist, rmsd_tot_dist, angl_goal_dist, angl_prev_dist, angl_tot_dist, andh_goal_dist," \
Definition at line 104 of file convert_bad_db.py.

3.5.2.10 res

convert_bad_db.res = cur_bad.execute(qry)
Definition at line 108 of file convert_bad_db.py.

3.5.2.11 res_arr

convert_bad_db.res_arr = res.fetchall()
Definition at line 110 of file convert_bad_db.py.

3.5.2.12 res_first

convert_bad_db.res_first = res.fetchone()
Definition at line 109 of file convert_bad_db.py.

3.5.2.13 vis_res

convert_bad_db.vis_res = res.fetchall()
Definition at line 117 of file convert_bad_db.py.

3.6 db_proc Namespace Reference

Functions

- tuple get_db_con (int tot_seeds=4)

  Creates the database with structure that fits exact number of seeds.
- NoReturn log_error (lite.Connection con, str type, int id)

  Writes an error message into the log table.
- int get_id_for_hash (lite.Connection con, str h_name)

  Searches main storage for id with given hash.
- tuple get_corr_vid_for_id (lite.Connection con, int max_id, list prev_ids, float last_gc)

  Used for recovery procedure.
- int get_corr_lid_for_id (lite.Connection con, int next_id, int vid_ts, int last_vis_id)

  Used for recovery procedure.
- list get_all_hashed_names (lite.Connection con)

  Fetches all hashes from the main_storage.
- NoReturn insert_into_main_stor (lite.Connection con, dict node_info, int curr_gc, str digest_name, str name)

  Inserts main information into the DB.
- NoReturn insert_into_visited (lite.Connection con, str hname, int gc)

  Inserts node processing event.
- NoReturn insert_into_log (lite.Connection con, str operation, str hname, str src, str dst, list bsf, int gc, float mul, list prev_arr, list goal_arr, str cur_metr_name)

  Inserts various information, like new best_so_far events, insertions into the open queue, etc.
- NoReturn copy_old_db (list main_dict_keys, list last_visited, str next_in_oq, float last_gc)

  Used during the recovery procedure.

3.6.1 Function Documentation

3.6.1.1 copy_old_db()

```
NoReturn db_proc.copy_old_db (
        list main_dict_keys,
        list last_visited,
        str next_in_oq,
        float last_gc )
```

Used during the recovery procedure.

list main_dict_keys: all hash values from the main_dict - storage of all metric information
list last_visited: several (3) recent values from the visited queue
str next_in_oq: next hash (id) in the open queue, used for double check
float last_gc: last greedy counter observed in the information from the pickle

Returns

Conditionally copies data from the previous DB into a new one as a part of the restore process.

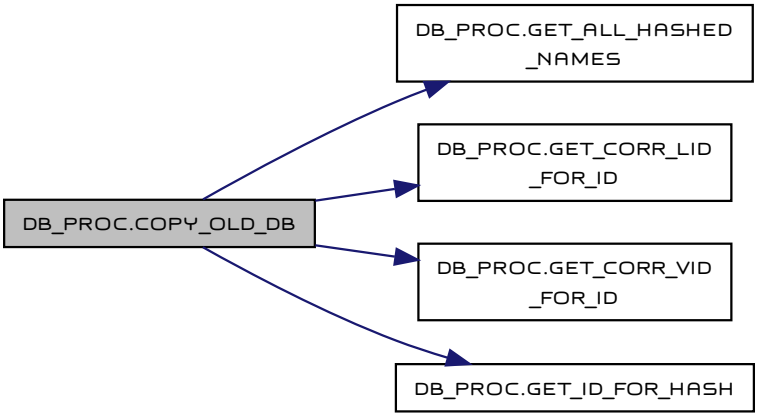
Definition at line 449 of file db_proc.py.

```
00449     """
00450     counter = 0
00451     db_path = os.getcwd()
00452     # db_name = 'results_{}.sqlite3'.format(counter)
00453     full_path = os.path.join(db_path, 'results_{}.sqlite3'.format(counter))
00454 
00455     while os.path.exists(full_path):
00456         prev_db = full_path
00457         counter += 1
00458         full_path = os.path.join(db_path, 'results_{}.sqlite3'.format(counter))
00459 
00460     #  yes, prev_db - the last one which exists
00461     cur_con = lite.connect(prev_db, check_same_thread=False, isolation_level=None)
00462 
00463     current_db_cur = cur_con.cursor()
00464 
00465     current_db_cur.execute("DELETE FROM log")
00466     current_db_cur.execute("DELETE FROM visited")
00467     current_db_cur.execute("DELETE FROM main_storage")
00468     cur_con.commit()
00469 
00470     prev_db_con = lite.connect(os.path.join(db_path, 'results_{}.sqlite3'.format(counter - 2)), check_same_thread=False, isolation_level=None)
00471 
00472     hashes = get_all_hashed_names(prev_db_con)
00473     for hash_hame in hashes:
00474         if hash_hame[0] in main_dict_keys:
00475             break
00476 
00477     max_id = get_id_for_hash(prev_db_con, hash_hame[0])
00478     prev_ids = [get_id_for_hash(prev_db_con, last_visited[0][2]), get_id_for_hash(prev_db_con, last_visited[1][2]),
       get_id_for_hash(prev_db_con, last_visited[2][2])]
00479     next_id = get_id_for_hash(prev_db_con, next_in_oq)
00480     # del last_visited, next_in_oq
00481     max_vid, vid_ts, last_vis_id = get_corr_vid_for_id(prev_db_con, max_id, prev_ids, last_gc)
00482     max_lid = get_corr_lid_for_id(prev_db_con, next_id, vid_ts, last_vis_id)
00483 
00484     prev_db_con.close()
00485     del prev_db_con, hash_hame, hashes, main_dict_keys
00486 
00487     current_db_cur.execute("ATTACH DATABASE ? AS prev_db", ('results_{}.sqlite3'.format(counter-2),))  # -1 - cur, -2 - prev
00488 
00489     current_db_cur.execute("INSERT INTO main.main_storage SELECT * FROM prev_db.main_storage WHERE prev_db.main_storage.id <= ?", (max_id,))
00490     cur_con.commit()
00491     current_db_cur.execute("INSERT INTO main.visited SELECT * FROM prev_db.visited WHERE prev_db.visited.vid <= ?", (max_vid,))
00492     cur_con.commit()
00493     current_db_cur.execute("INSERT INTO main.log SELECT * FROM prev_db.log WHERE prev_db.log.lid <= ?", (max_lid,))
00494     cur_con.commit()
00495 
00496 #
00497 # def sync_state_with_db(state):
00498 #     counter = 0
00499 #     db_path = os.getcwd()
00500 #     db_name = 'results_{}.sqlite3'.format(counter)
00501 #     full_path = os.path.join(db_path, 'results_{}.sqlite3'.format(counter))
00502 #
00503 #     while os.path.exists(full_path):
00504 #         prev_db = full_path
00505 #         counter += 1
00506 #         full_path = os.path.join(db_path, 'results_{}.sqlite3'.format(counter))
00507 #
00508 #     #  yes, prev_db - last one which exists
00509 #     cur_con = lite.connect(prev_db, check_same_thread=False, isolation_level=None)
00510 #
00511 #     current_db_cur = cur_con.cursor()
00512 #
00513 #     current_db_cur.execute("DELETE FROM log")
00514 #     # get_conn
00515 #     # get indexes
00516 #     # drop all log with
00517 #     # drop all vis with
00518 #     # drop all main with
00519 #     # vacuum
00520 #     return True
```

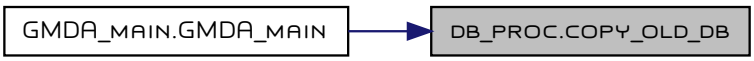
References get_all_hashed_names(), get_corr_lid_for_id(), get_corr_vid_for_id(), and get_id_for_hash().

Referenced by GMDA_main.GMDA_main().

Here is the call graph for this function:



Here is the caller graph for this function:



3.6.1.2 get_all_hashed_names()

```
list db_proc.get_all_hashed_names (
        lite.Connection con )
```

Fetches all hashes from the main_storage.

lite.Connection con: DB connection

Returns

:return: list of all hashes in the main_storage :rtype: list

Definition at line 286 of file db_proc.py.

```
00286     """
00287     qry = "SELECT hashed_name FROM main_storage order by id desc"
00288     cur = con.cursor()
00289     result = cur.execute(qry)
00290     rows = result.fetchall()
00291     return rows
00292
00293
```

Referenced by copy_old_db().

Here is the caller graph for this function:

3.6.1.3 get_corr_lid_for_id()

```
int db_proc.get_corr_lid_for_id (
        lite.Connection con,
        int  next_id,
        int  vid_ts,
        int  last_vis_id )
```

Used for recovery procedure.
Tries to find matching sequence of nodes in the log table

lite.Connection con: DB connection
int next_id: next id we expect to see in the log, used for double check
int vid_ts: visited timestampt
int last_vis_id: last visited id

Returns

:return: the latest valid log_id

Definition at line 227 of file db_proc.py.

```
00227     """
00228     qry = "SELECT lid, CAST(strftime('%s', Timestamp) AS INT) FROM log WHERE id='{}' AND src='WQ' AND dst='VIZ' order by
       lid".format(last_vis_id)
00229     cur = con.cursor()
00230     result = cur.execute(qry)
00231     rows = result.fetchall()
00232     if len(rows) > 1:
00233         # find the smallest dist between vid_ts and all ts
00234         dist = abs(rows[0][1] - vid_ts)
00235         good_lid = int(rows[0][0])
00236         i = 1
00237         while i < len(rows):
00238             if abs(rows[i][1] - vid_ts) <= dist:
00239                 dist = abs(rows[i][1] - vid_ts)
00240                 good_lid = int(rows[i][0])
00241             i += 1
00242     else:
00243         good_lid = int(rows[0][0])
00244
00245     # so now we have good_lid which is very close, but may be not exact
00246
00247     qry = "SELECT lid, operation, id, src, dst FROM log WHERE lid > {} order by lid limit 4".format(good_lid)
00248     result = cur.execute(qry)
00249     rows = result.fetchall()
00250     i = 0
00251     if (rows[i][1] == 'current' and rows[i][4] == 'WQ') or rows[i][1] == 'skip':
00252         good_lid += 1
00253         i += 1
00254         if rows[i][1] == 'prom_O':
00255             good_lid += 1
00256             i += 1
00257
00258     if rows[i][1] == 'result' and rows[i][4] == 'VIZ' and int(rows[i][2]) == next_id:
00259         print("Log table ID computed perfectly.")
00260
00261     return good_lid
00262
00263
00264 # I am not using it
00265 # def get_max_id_from_main(con):
00266 #     qry = "SELECT max(id) FROM main_storage"
00267 #     cur = con.cursor()
00268 #     result = cur.execute(qry)
00269 #     row = result.fetchone()
00270 #     if row is not None:
```

```
00271 #         num = int(row[0])
00272 #     else:
00273 #         num = None
00274 #     return num
00275
00276
```

Referenced by copy_old_db().
Here is the caller graph for this function:

3.6.1.4 get_corr_vid_for_id()

```
tuple db_proc.get_corr_vid_for_id (
        lite.Connection con,
        int  max_id,
        list  prev_ids,
        float  last_gc )
```

Used for recovery procedure.
Tries to find matching sequence of nodes in the visited table

lite.Connection con: DB connection
int max_id: maximum value of the id (defined by previous search as the common latest id)
list prev_ids: several ids that should match
float last_gc: extra check, whether greed counters also match

Returns

:return: last common visited id, timestamp, and id :rtype: tuple

Definition at line 194 of file db_proc.py.

```
00194     """
00195     qry = "SELECT vid, id, CAST(strftime('%s', Timestamp) AS INT), cur_gc FROM visited WHERE id<'{}' AND id in ({}, {}, {}) order by vid
       desc".format(max_id, prev_ids[0], prev_ids[1], prev_ids[2])
00196     cur = con.cursor()
00197     result = cur.execute(qry)
00198     rows = result.fetchall()
00199     i = 0
00200     while i+2 < len(rows):  # 3 for next version
00201         if rows[i][0] - rows[i+1][0] == 1 and rows[i+1][0] - rows[i+2][0] == 1:
00202             break
00203         i += 1
00204     if i+2 >= len(rows):
00205         raise Exception("Sequence of events from pickle dump not found in DB")
00206     last_good_vid = rows[i][0]
00207     last_good_ts = rows[i][2]
00208     last_good_id = rows[i][1]
00209     if last_gc != int(rows[i][3]):
00210         raise Exception('Everything looked good, but greed counters did not match.\n Check manually and comment this exception if you are sure
       that this is normal.\n')
00211
00212     return last_good_vid, last_good_ts, last_good_id
00213
00214
```

Referenced by copy_old_db().
Here is the caller graph for this function:

3.6.1.5 get_db_con()

```
tuple db_proc.get_db_con (
        int tot_seeds = 4 )
```

Creates the database with structure that fits exact number of seeds.
Filename for DB is generated as next number after the highest consequent found. If there is results_0.sqlite3, then next will be results_1.sqlite3 if it did not exist.

int tot_seeds: number of seeds used in the current run
:type tot_seeds: int

Returns

:return: database connection and name

Connection to the new database and it's name.
Definition at line 36 of file db_proc.py.

```
00036     """
00037     counter = 0
00038     # db_path = '/dev/shm/GMDApy'
00039     db_path = os.getcwd()
00040     db_name = 'results_{}.sqlite3'.format(counter)
00041     full_path = os.path.join(db_path, 'results_{}.sqlite3'.format(counter))
00042     while os.path.exists(full_path):
00043         counter += 1
00044         full_path = os.path.join(db_path, 'results_{}.sqlite3'.format(counter))
00045 
00046     con = lite.connect(full_path, check_same_thread=False, isolation_level=None)
00047 
00048     cur = con.cursor()
00049     cur.execute("""CREATE TABLE main_storage (
00050         id               INTEGER   PRIMARY KEY AUTOINCREMENT,
00051 
00052         rmsd_goal_dist   FLOAT    NOT NULL,
00053         rmsd_prev_dist   FLOAT    NOT NULL,
00054         rmsd_tot_dist    FLOAT    NOT NULL,
00055 
00056         angl_goal_dist   FLOAT    NOT NULL,
00057         angl_prev_dist   FLOAT    NOT NULL,
00058         angl_tot_dist    FLOAT    NOT NULL,
00059 
00060         andh_goal_dist   INTEGER   NOT NULL,
00061         andh_prev_dist   INTEGER   NOT NULL,
00062         andh_tot_dist    INTEGER   NOT NULL,
00063 
00064         and_goal_dist    INTEGER   NOT NULL,
00065         and_prev_dist    INTEGER   NOT NULL,
00066         and_tot_dist     INTEGER   NOT NULL,
00067 
00068         xor_goal_dist    INTEGER   NOT NULL,
00069         xor_prev_dist    INTEGER   NOT NULL,
00070         xor_tot_dist     INTEGER   NOT NULL,
00071 
00072         curr_gc          INTEGER   NOT NULL,
00073         Timestamp        DATETIME DEFAULT (CURRENT_TIMESTAMP),
00074         hashed_name      CHAR (32) NOT NULL UNIQUE,
00075         name             TEXT
00076         );""")
00077     con.commit()
00078     cur.execute("""CREATE TABLE visited (
00079         vid        INTEGER   PRIMARY KEY AUTOINCREMENT, \
00080         id        REFERENCES main_storage (id),
00081         cur_gc    INTEGER,
00082         Timestamp DATETIME DEFAULT (CURRENT_TIMESTAMP)
00083     );""")
00084     con.commit()
00085 
00086     add_ind_q = 'CREATE INDEX viz_id_idx ON visited (id);'
00087     cur.execute(add_ind_q)
00088     con.commit()
00089 
00090     # id         REFERENCES main_storage (id), \
00091     init_query = 'CREATE TABLE log ( \
00092         lid        INTEGER   PRIMARY KEY AUTOINCREMENT, \
00093         operation  INTEGER, \
00094         id         INTEGER, \
00095         src        CHAR (8), \
00096         dst        CHAR(8), \
00097         cur_metr   CHAR(5), \
00098         gc         INTEGER , \
00099         mul        FLOAT, \
00100         bsfr        FLOAT, \
```

```
00101         bsfn        FLOAT, \
00102         bsfh        FLOAT, \
00103         bsfa        FLOAT, \
00104         bsfx        FLOAT, \
00105         Timestamp  DATETIME DEFAULT (CURRENT_TIMESTAMP)'  # no this is not an error
00106     for i in range(tot_seeds):
00107         init_query += ", \
00108         dist_from_prev_{0} FLOAT, \
00109         dist_to_goal_{0}   FLOAT ".format(i+1)
00110     init_query += ');'
00111
00112     cur.execute(init_query)
00113     con.commit()
00114     add_ind_q = 'CREATE INDEX log_id_idx ON log (id);'
00115     cur.execute(add_ind_q)
00116     con.commit()
00117
00118     cur.execute('PRAGMA mmap_size=-64000')  # 32M
00119     cur.execute('PRAGMA journal_mode =  OFF')
00120     cur.execute('PRAGMA synchronous = OFF')
00121     cur.execute('PRAGMA temp_store = MEMORY')
00122     cur.execute('PRAGMA threads = 32')
00123
00124     return con, db_name
00125
00126
```

Searches main storage for id with given hash.

lite.Connection con: DB connection
str h_name: hashname to use during the search

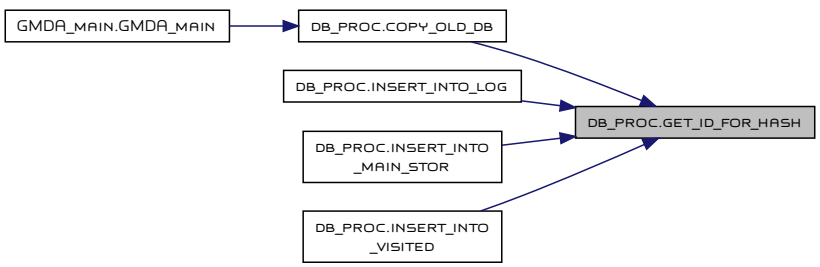
Returns

:return: id or None if not found

Definition at line 167 of file db_proc.py.

```
00167     """
00168     con.commit()
00169     qry = "SELECT id FROM main_storage WHERE hashed_name='{}'".format(h_name)
00170     cur = con.cursor()
00171     result = cur.execute(qry)
00172     row = result.fetchone()
00173     if row is not None:
00174         num = int(row[0])
00175     else:
00176         num = None
00177     # if not isinstance(num, int):
00178     #     print("ID was not found in main stor")
00179     return num
00180
00181
```

Referenced by copy_old_db(), insert_into_log(), insert_into_main_stor(), and insert_into_visited().
Here is the caller graph for this function:



3.6.1.6 insert_into_log()

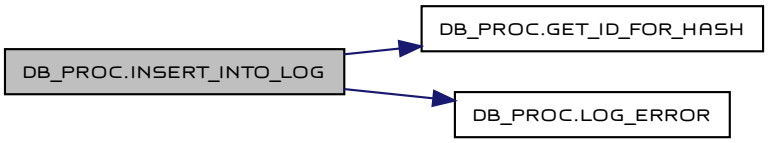
```
NoReturn  db_proc.insert_into_log (
            lite.Connection con,
```

```
                str  operation,
                str  hname,
                str  src,
                str  dst,
                list  bsf,
                int  gc,
                float  mul,
                list  prev_arr,
                list  goal_arr,
                str  cur_metr_name )
Inserts various information, like new best_so_far events, insertions into the open queue, etc.

    lite.Connection con: DB connection
    str  operation: result, current, prom_0, skip
    str  hname: hash name, same as MD filenames
    str  src: from WQ (open queue)
    str  dst: to VIZ (visited)
    list  bsf: all best_so_far values for each metric
    int  gc: greedy counter - affects events like seed change
    float   mul: greedy multiplier - controls greediness
    list  prev_arr: distance from the previous node
    list  goal_arr: distance to the goal
    str  cur_metr_name: name of the current metric


Returns

     Stores data in the DB in a log table.

Definition at line 382 of file db_proc.py.
00382     Stores data in the DB in a log table.
00383     """
00384     src = 'None' if src == '' else src
00385     dst = 'None' if dst == '' else dst
00386     nid = get_id_for_hash(con, hname)
00387     nid = 'None' if nid is None else nid
00388     columns = 'operation, id, src, dst, cur_metr, bsfr, bsfn, bsfh, bsfa, bsfx, gc, mul, '
00389 
00390     if not isinstance(goal_arr, (list,)):  # short version for skip operation
00391         columns += 'dist_from_prev_1, dist_to_goal_1'
00392         final_str = ', '.join('"{}"'.format(elem) if isinstance(elem, str) else str(elem)
00393                               for elem in (operation, nid, src, dst, cur_metr_name, bsf[0], bsf[1], bsf[2], bsf[3],
00394                                            bsf[4], gc, mul, prev_arr, goal_arr))
00395     else:
00396         nseeds = len(prev_arr)  # long version for append operation
00397         columns += ', '.join(('dist_from_prev_{0}'.format(i+1) for i in range(nseeds))) + ', '
00398         columns += ', '.join(('dist_to_goal_{0}'.format(i+1) for i in range(nseeds)))
00399         prev_arr_str = ', '.join((str(elem) for elem in prev_arr))
00400         goal_arr_str = ', '.join((str(elem) for elem in goal_arr))
00401         final_str = ', '.join('"{}"'.format(elem) if isinstance(elem, str) else str(elem)
00402                               for elem in (operation, nid, src, dst, cur_metr_name, bsf[0], bsf[1], bsf[2], bsf[3], bsf[4], gc, mul))
00403         final_str += ", ".join((", prev_arr_str, goal_arr_str))
00404 
00405     qry = 'INSERT INTO log({}) VALUES ({})'.format(columns, final_str)
00406     cur = con.cursor()
00407     try:
00408         cur.execute(qry)
00409         con.commit()
00410     except Exception as e:
00411         print(e, '\nqry: ', operation, hname, src, dst, bsf, gc, mul, prev_arr, goal_arr)
00412         print('Extra info: ', qry)
00413         print('Type of function : {}'.format('Short' if not isinstance(goal_arr, (list,)) else 'Long'))
00414         log_error(con, 'LOG', nid)
00415 
00416 
00417 # def prep_insert_into_log(con, operation, name, src, dst, bsf, gc, mul, prev_arr, goal_arr):
00418 #     src = 'None' if src == '' else src
00419 #     nid = get_id_for_name(con, name)
00420 #     columns = 'operation, id, src, dst, bsf, gc, mul, '
00421 #
00422 #     if isinstance(goal_arr, (float, int)):  # short version
00423 #         columns += 'dist_from_prev_1, dist_to_goal_1'
00424 #         final_str = ', '.join('"{}"'.format(elem) if isinstance(elem, str) else str(elem)
00425 #                               for elem in (operation, nid, src, dst, bsf, gc, mul, prev_arr, goal_arr))
00426 #     else:
00427 #         nseeds = len(prev_arr)
00428 #         columns += ', '.join(('dist_from_prev_{0}, dist_to_goal_{0}'.format(i+1) for i in range(nseeds)))
00429 #         prev_arr_str = ', '.join((str(elem) for elem in prev_arr))
00430 #         goal_arr_str = ', '.join((str(elem) for elem in goal_arr))
```

```
00431 #         final_str = ', '.join('"{}"'.format(elem) if isinstance(elem, str) else str(elem)
00432 #                               for elem in (operation, nid, src, dst, bsf, gc, mul))
00433 #         final_str += ", ".join((", prev_arr_str, goal_arr_str))
00434 #
00435 #     return final_str
00436
00437
```

References get_id_for_hash(), and log_error().

Here is the call graph for this function:



3.6.1.7 insert_into_main_stor()

```
NoReturn db_proc.insert_into_main_stor (
                lite.Connection con,
                dict  node_info,
                int  curr_gc,
                str  digest_name,
                str  name )
```

Inserts main information into the DB.

lite.Connection con: DB connection
dict node_info: all metric values associated with the node
int curr_gc: current greedy counter
str digest_name: hash name for the path, same as filenames for MD simulations
str name: path from the origin separated by _

Returns

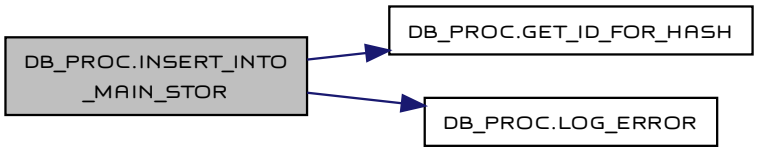
Stores data in the DB in a main_storage table.

Definition at line 306 of file db_proc.py.

```
00306     """
00307     # con = lite.connect('results_8.sqlite3', timeout=300, check_same_thread=False, isolation_level=None)
00308     # qry = "INSERT OR IGNORE INTO main_storage(rmsd_goal_dist, rmsd_prev_dist, rmsd_tot_dist, angl_goal_dist,
00309     # angl_prev_dist, angl_tot_dist," \
00310     qry = "INSERT INTO main_storage(rmsd_goal_dist, rmsd_prev_dist, rmsd_tot_dist, angl_goal_dist, angl_prev_dist, angl_tot_dist," \
00311           "                         andh_goal_dist, andh_prev_dist, andh_tot_dist, and_goal_dist, and_prev_dist, and_tot_dist," \
00312           "                         xor_goal_dist, xor_prev_dist, xor_tot_dist, curr_gc, hashed_name, name) " \
00313           "VALUES (?, ?, ?, ?, ?, ?,   ?, ?, ?, ?, ?, ?,   ?, ?, ?, ?, ?, ?)"
00314     cur = con.cursor()
00315     try:
00316         cur.execute(qry, [str(elem) for elem in (node_info['RMSD_to_goal'], node_info['RMSD_from_prev'], node_info['RMSD_dist_total'],
00317                           node_info['ANGL_to_goal'], node_info['ANGL_from_prev'], node_info['ANGL_dist_total'],
00318                           node_info['AND_H_to_goal'], node_info['AND_H_from_prev'], node_info['AND_H_dist_total'],
00319                           node_info['AND_to_goal'], node_info['AND_from_prev'], node_info['AND_dist_total'],
00320                           node_info['XOR_to_goal'], node_info['XOR_from_prev'], node_info['XOR_dist_total'],
00321                           curr_gc, digest_name, name)])
00322         con.commit()
00323     except Exception as e:
00324         nid = get_id_for_hash(con, digest_name)
00325         log_error(con, 'MAIN', nid)
00326         qry = "SELECT * FROM main_storage WHERE id=?"
00327         cur = con.cursor()
00328         result = cur.execute(qry, nid)
00329         row = result.fetchone()
00330         print('Original elment in MAIN:', row)
00331         qry = "SELECT * FROM log WHERE id=?"
00332         cur = con.cursor()
00333         result = cur.execute(qry, nid)
00334         rows = result.fetchall()
00335         print('Printing all I found in the log about this ID:')
```

```
00336         for row in rows:
00337             print(row)
00338         print('Error element message: ', e, '\nqry: ', node_info, curr_gc, digest_name, name)
00339 
00340 
```

References get_id_for_hash(), and log_error().
Here is the call graph for this function:



3.6.1.8 insert_into_visited()

```
NoReturn db_proc.insert_into_visited (
                lite.Connection con,
                str  hname,
                int  gc )
```

Inserts node processing event.

lite.Connection con: DB connection
str hname: hashname, same as MD filenames
int gc: greedy counter

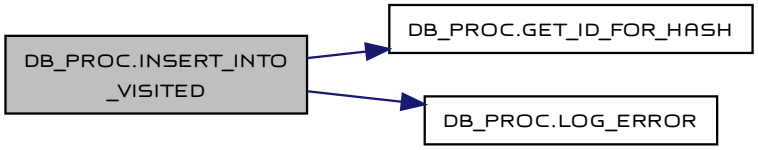
Returns

Stores data in the DB in a visited table.

Definition at line 352 of file db_proc.py.

```
00352     """
00353     nid = get_id_for_hash(con, hname)
00354     qry = 'INSERT INTO visited( id, cur_gc ) VALUES (?, ?)'
00355     cur = con.cursor()
00356     try:
00357         cur.execute(qry, (nid, gc))
00358         con.commit()
00359     except Exception as e:
00360         print(e, '\nqry: ', hname, gc)
00361         log_error(con, 'VIZ', nid)
00362 
00363 
```

References get_id_for_hash(), and log_error().
Here is the call graph for this function:



3.6.1.9 log_error()

```
NoReturn db_proc.log_error (
                lite.Connection con,
                str  type,
                int  id )
```

Writes an error message into the log table.

con: current DB connection
type: error type
id: id associated with the error

Returns

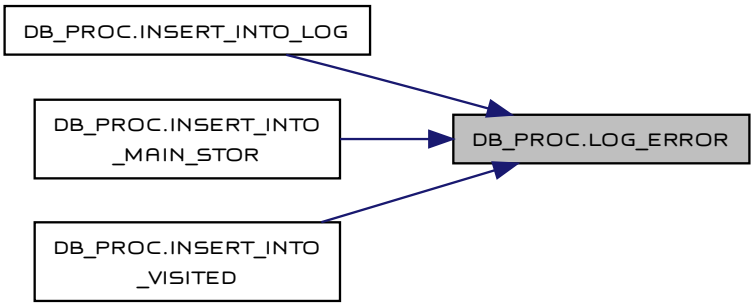
Adds one row in the log table.

Definition at line 137 of file db_proc.py.

```
00137     """
00138     qry = 'INSERT INTO log (id, operation, dst)  VALUES ({}, "ERROR", "{}")'.format(id, type)
00139     try:
00140         con.cursor().execute(qry)
00141         con.commit()
00142     except Exception as e:
00143         print(e)
00144         print('Error in "log_error": {}'.format(qry))
00145
00146
00147 # def get_id_for_name(con, name):
00148 #     con.commit()
00149 #     qry = "SELECT id FROM main_storage WHERE name='{}'".format(name)
00150 #     cur = con.cursor()
00151 #     result = cur.execute(qry)
00152 #     num = int(result.fetchone()[0])
00153 #     if not isinstance(num, int):
00154 #         raise Exception("ID was not found in main stor")
00155 #     return num
00156
00157
```

Referenced by insert_into_log(), insert_into_main_stor(), and insert_into_visited().

Here is the caller graph for this function:



3.7 fix_filenames Namespace Reference

Variables

- files = os.walk('.').__next__()[2]
- int counter = 0

3.7.1 Variable Documentation

3.7.1.1 counter

int fix_filenames.counter = 0

Definition at line 7 of file fix_filenames.py.

3.7.1.2 files fix_filenames.files = os.walk('.').__next__()[2]

Definition at line 5 of file fix_filenames.py.

3.8 gen_mdp Namespace Reference

Functions

- str get_mdp (int seed, int temp, str name='default')

  Generates text for .mdp file with simulation settings.

3.8.1 Function Documentation

3.8.1.1 get_mdp() str gen_mdp.get_mdp (int seed, int temp, str name = 'default')

Generates text for .mdp file with simulation settings.

int seed: seed to be used for initial velocities generation
int temp: temperature of the experiment
str name: name of the experiment inside the .mdp file

Returns

:return: string with .mdp text :rtype: str

Definition at line 22 of file gen_mdp.py.

```
00022     """
00023     calibration_mdp = "\
00024 ; Run parameters\n\
00025 integrator  = md        ; leap-frog integrator\n\
00026 nsteps      = 10000     ; 2 * 10000 = 20 ps\n\
00027 dt          = 0.002     ; 2 fs\n\
00028 ld-seed     = {2:d}     ; \n\
00029 ; Output control\n\
00030 nstxout     = 0     ; save coordinates every 0.0 ps\n\
00031 nstvout     = 0     ; save velocities every 0.0 ps\n\
00032 nstenergy   = 10000 ; save energies every 0.0 ps\n\
00033 nstlog      = 0     ; update log file every 0.0 ps\n\
00034 nstxout-compressed  = 10000 ; save coordinates every 0.0 ps\n\
00035 energygrps  = Protein SOL\n\
00036 ; Bond parameters\n\
00037 continuation            = no        ; first dynamics run\n\
00038 constraint_algorithm    = lincs     ; holonomic constraints \n\
00039 constraints             = h-bonds   ; all bonds (even heavy atom-H bonds) constrained\n\
00040 lincs_iter              = 1         ; accuracy of LINCS\n\
00041 lincs_order             = 4         ; also related to accuracy\n\
00042 ; Neighborsearching\n\
00043 cutoff-scheme   = Verlet\n\
00044 ns_type         = grid      ; search neighboring grid cells\n\
00045 nstlist         = 10        ; 20 fs, largely irrelevant with Verlet\n\
00046 rcoulomb        = 1.0       ; short-range electrostatic cutoff (in nm)\n\
00047 rvdw            = 1.0       ; short-range van der Waals cutoff (in nm)\n\
00048 ; Electrostatics\n\
00049 coulombtype     = PME   ; Particle Mesh Ewald for long-range electrostatics\n\
00050 pme_order       = 4     ; cubic interpolation\n\
00051 fourierspacing  = 0.16  ; grid spacing for FFT\n\
00052 ; Temperature coupling is on\n\
00053 tcoupl      = V-rescale             ; modified Berendsen thermostat\n\
00054 tc-grps     = Protein Non-Protein   ; two coupling groups - more accurate\n\
00055 tau_t       = 0.1     0.1           ; time constant, in ps\n\
00056 ref_t       = {1:d}       {1:d}           ; reference temperature, one for each group, in K\n\
00057 ; Pressure coupling is off\n\
00058 pcoupl      = no        ; no pressure coupling in NVT\n\
00059 ; Periodic boundary conditions\n\
00060 pbc     = xyz           ; 3-D PBC\n\
00061 ; Dispersion correction\n\
00062 DispCorr    = EnerPres  ; account for cut-off vdW scheme\n\
00063 ; Velocity generation\n\
00064 gen-vel     = yes       ; assign velocities from Maxwell distribution\n\
00065 gen-temp    = {1:d}     ; temperature for Maxwell distribution\n\
00066 gen-seed    = {2:d}     ; generate a random seed".format(name, temp, seed)
00067     return calibration_mdp
```

Referenced by helper_funcs.get_seed_dirs().

Here is the caller graph for this function:

3.9 generate_REMD_dirs Namespace Reference

Functions

- def gen_dirs ()
- def get_mdp_str_ener_gr (str name, float temp, int seed, int steps)
- def get_mdp_str_gpu (str name, float temp, int seed, int steps)

3.9.1 Function Documentation

3.9.1.1 gen_dirs() def generate_REMD_dirs.gen_dirs ( )

Definition at line 7 of file generate_REMD_dirs.py.

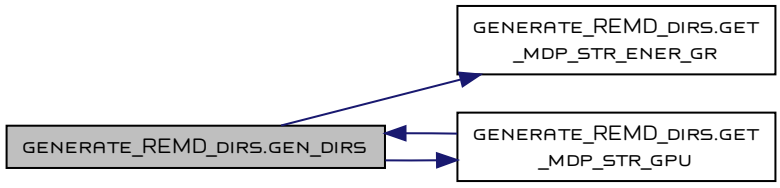
```
00007 def gen_dirs():
00008     root_dir = 'REMD_profiles'
00009     cur_prot = 'TRP'
00010     tot_steps = 31250000  # trp 100 000
00011     # tot_steps = 166670000  # vil 100 000
00012     # tot_steps = 250000000  # gb1 800 000
00013
00014     full_path = os.path.join(root_dir, cur_prot)
00015     ffs = ['amber', 'charm', 'gromos', 'opls']
00016
00017     trp_profile_1 = [300.00, 302.87, 305.77, 308.69, 311.63, 314.59, 317.57, 320.58, 323.62, 326.67, 329.75, 332.86, 335.98, 339.13, 342.31,
00018                 345.51, 348.74, 351.99, 355.26, 358.56, 361.90, 365.25, 368.63, 372.04, 375.48, 378.93, 382.42, 385.94, 389.48, 393.05,
00019                 396.65, 400.00]  # amber, charm, opls
00020     trp_profile_2 = [300.00, 302.90, 305.83, 308.78, 311.76, 314.76, 317.78, 320.82, 323.89, 326.98, 330.10, 333.25, 336.41, 339.61, 342.82,
00021                 346.07, 349.34, 352.63, 355.95, 359.30, 362.67, 366.07, 369.50, 372.94, 376.42, 379.92, 383.46, 387.02, 390.62, 394.23,
00022                 397.89, 400.00]  # gromos
00023
00024     vil_profile_1 = [300.00, 303.07, 306.17, 309.30, 312.46, 315.64,
00025 318.85, 322.09, 325.35, 328.63, 331.95, 335.28, 338.65, 342.05, 345.48, 348.93,
00026 352.42, 355.93, 359.48, 363.05, 366.65, 370.29, 373.95, 377.64, 381.37, 385.13,
00027 388.91, 392.73, 396.59, 400.00
00028 ]  # amber, charm, opls
00029     vil_profile_2 = [300.00, 303.15, 306.32, 309.52, 312.75, 316.01, 319.29,
00030 322.58, 325.92, 329.29, 332.68, 336.11, 339.57, 343.05, 346.57, 350.11, 353.69,
00031 357.29, 360.93, 364.59, 368.29, 372.02, 375.79, 379.58, 383.41, 387.27, 391.17,
00032 395.10, 399.06, 400.00
00033 ]  # gromos
00034
00035     gb1_profile_1 = [300.00, 302.57, 305.16, 307.76, 310.39, 313.03,
00036                      315.69, 318.37, 321.07, 323.78, 326.52, 329.27, 332.05, 334.84, 337.62, 340.45,
00037                      343.30, 346.17, 349.07, 351.98, 354.91, 357.86, 360.84, 363.83, 366.84, 369.88,
00038                      372.94, 376.01, 379.11, 382.22, 385.37, 388.53, 391.72, 394.93, 398.16, 400.00
00039 ]  # amber, charm, opls
00040     gb1_profile_2 = [300.00, 302.57, 305.15, 307.76, 310.38, 313.03, 315.69,
00041                      318.37, 321.07, 323.78, 326.52, 329.27, 332.05, 334.84, 337.62, 340.45, 343.30,
00042                      346.17, 349.06, 351.98, 354.91, 357.86, 360.84, 363.83, 366.84, 369.88, 372.94,
00043                      376.01, 379.11, 382.23, 385.37, 388.54, 391.70, 394.91, 398.14, 400.00
00044 ]  # gromos
00045
00046     profile_1 = trp_profile_1
00047     profile_2 = trp_profile_2
00048
00049     temperartures = [
00050         profile_1,
00051         profile_1,
00052         profile_2,
00053         profile_1
00054     ]
00055
00056     try:
00057         os.mkdir(root_dir)
00058     except:
00059         print('Failed to create directory {}.'.format(root_dir))
```

```
00060
00061     try:
00062         os.mkdir(full_path)
00063     except:
00064         print('Failed to create directory {}.'.format(full_path))
00065
00066     gpu_flag = True
00067
00068     for i, ff in enumerate(ffs):
00069         work_dir = os.path.join(full_path, ff)
00070         try:
00071             os.mkdir(work_dir)
00072         except:
00073             print('Failed to create directory {}.'.format(os.path.join(full_path, ff)))
00074         for j, temp in enumerate(temperartures[i]):
00075             if gpu_flag:
00076                 mdp_content = get_mdp_str_gpu(name='REMD {}@{}'.format(cur_prot, ff), temp=temp, seed=1, steps=tot_steps)
00077             else:
00078                 mdp_content = get_mdp_str_ener_gr(name='REMD {}@{}'.format(cur_prot, ff), temp=temp, seed=1, steps=tot_steps)
00079             temp_dir = os.path.join(work_dir, '{}_{}_{}'.format(cur_prot, ff, j+1))
00080             try:
00081                 os.mkdir(temp_dir)
00082             except:
00083                 pass
00084             with open(os.path.join(temp_dir, 'md.mdp'), 'w') as mdp_file:
00085                 mdp_file.write(mdp_content)
00086
00087         # cp2(os.path.join(conf_files_dir, 'prot.ndx'), work_dir)
00088         # if ff == 'charm':
00089         #     cp2(os.path.join(conf_files_dir, 'charmm36-nov2018.ff'), work_dir)
00090
00091
00092 def get_mdp_str_ener_gr(name: str, temp: float, seed: int, steps: int):
00093     """
00094
00095          str  name:
00096          float temp:
00097          int seed:
00098          int steps:
```

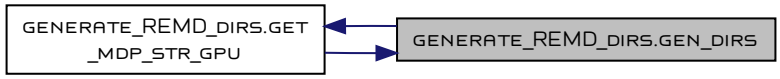
References get_mdp_str_ener_gr(), and get_mdp_str_gpu().

Referenced by get_mdp_str_gpu().

Here is the call graph for this function:



Here is the caller graph for this function:



3.9.1.2 get_mdp_str_ener_gr() def generate_REMD_dirs.get_mdp_str_ener_gr (

```
                str  name,
                float  temp,
                int  seed,
                int  steps )
```

Definition at line 99 of file generate_REMD_dirs.py.

```
00099     """
00100     mdp_str = "\
00101         ; Run parameters\n\
00102         integrator  = md        ; leap-frog integrator\n\
00103         nsteps      = {3:d}     ; 2 * 10000 = 20 ps\n\
00104         dt          = 0.002     ; 2 fs\n\
00105         ld-seed     = {2:d}     ; \n\
00106         ; Output control\n\
00107         nstxout     = 0     ; save coordinates every 0.0 ps\n\
00108         nstvout     = 0     ; save velocities every 0.0 ps\n\
00109         nstenergy   = 10000 ; save energies every 0.0 ps\n\
00110         nstlog      = 10000     ; update log file every 0.0 ps\n\
00111         nstxout-compressed  = 10000 ; save coordinates every 0.0 ps\n\
00112         energygrps  = Protein SOL\n\
00113         ; Bond parameters\n\
00114         continuation            = no        ; first dynamics run\n\
00115         constraint_algorithm    = lincs     ; holonomic constraints \n\
00116         constraints             = h-bonds   ; all bonds (even heavy atom-H bonds) constrained\n\
00117         lincs_iter              = 1         ; accuracy of LINCS\n\
00118         lincs_order             = 4         ; also related to accuracy\n\
00119         ; Neighborsearching\n\
00120         cutoff-scheme   = Verlet\n\
00121         ns_type         = grid      ; search neighboring grid cells\n\
00122         nstlist         = 10        ; 20 fs, largely irrelevant with Verlet\n\
00123         rcoulomb        = 1.0       ; short-range electrostatic cutoff (in nm)\n\
00124         rvdw            = 1.0       ; short-range van der Waals cutoff (in nm)\n\
00125         ; Electrostatics\n\
00126         coulombtype     = PME   ; Particle Mesh Ewald for long-range electrostatics\n\
00127         pme_order       = 4     ; cubic interpolation\n\
00128         fourierspacing  = 0.16  ; grid spacing for FFT\n\
00129         ; Temperature coupling is on\n\
00130         tcoupl      = V-rescale             ; modified Berendsen thermostat\n\
00131         tc-grps     = Protein Non-Protein   ; two coupling groups - more accurate\n\
00132         tau_t       = 0.1     0.1           ; time constant, in ps\n\
00133         ref_t       = {1:f}       {1:f}           ; reference temperature, one for each group, in K\n\
00134         ; Pressure coupling is off\n\
00135         pcoupl      = no        ; no pressure coupling in NVT\n\
00136         ; Periodic boundary conditions\n\
00137         pbc     = xyz           ; 3-D PBC\n\
00138         ; Dispersion correction\n\
00139         DispCorr    = EnerPres  ; account for cut-off vdW scheme\n\
00140         ; Velocity generation\n\
00141         gen-vel     = yes       ; assign velocities from Maxwell distribution\n\
00142         gen-temp    = {1:f}     ; temperature for Maxwell distribution\n\
00143         gen-seed    = {2:d}     ; generate a random seed".format(name, temp, seed, steps)
00144     return mdp_str
00145
00146
00147 def get_mdp_str_gpu(name: str, temp: float, seed: int, steps: int):
00148     """
00149
00150          str  name:
00151          float temp:
00152          int seed:
00153          int steps:
```

Referenced by gen_dirs().

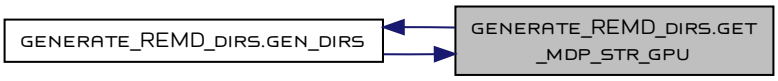
Here is the caller graph for this function:

3.9.1.3 get_mdp_str_gpu() def generate_REMD_dirs.get_mdp_str_gpu (

```
                str  name,
                float  temp,
```

```
                int  seed,
                int  steps )
Definition at line 154 of file generate_REMD_dirs.py.
00154     """
00155     mdp_str = "\
00156         ; Run parameters\n\
00157         integrator  = md        ; leap-frog integrator\n\
00158         nsteps      = {3:d}     ; 2 * 10000 = 20 ps\n\
00159         dt          = 0.002     ; 2 fs\n\
00160         ld-seed     = {2:d}     ; \n\
00161         ; Output control\n\
00162         nstxout     = 0     ; save coordinates every 0.0 ps\n\
00163         nstvout     = 0     ; save velocities every 0.0 ps\n\
00164         nstenergy   = 0 ; save energies every 0.0 ps\n\
00165         nstlog      = 10000     ; update log file every 0.0 ps\n\
00166         nstxout-compressed  = 10000 ; save coordinates every 0.0 ps\n\
00167         ; Bond parameters\n\
00168         continuation            = no        ; first dynamics run\n\
00169         constraint_algorithm    = lincs     ; holonomic constraints \n\
00170         constraints             = h-bonds   ; all bonds (even heavy atom-H bonds) constrained\n\
00171         lincs_iter              = 1         ; accuracy of LINCS\n\
00172         lincs_order             = 4         ; also related to accuracy\n\
00173         ; Neighborsearching\n\
00174         cutoff-scheme   = Verlet\n\
00175         ns_type         = grid      ; search neighboring grid cells\n\
00176         nstlist         = 10        ; 20 fs, largely irrelevant with Verlet\n\
00177         rcoulomb        = 1.0       ; short-range electrostatic cutoff (in nm)\n\
00178         rvdw            = 1.0       ; short-range van der Waals cutoff (in nm)\n\
00179         ; Electrostatics\n\
00180         coulombtype     = PME   ; Particle Mesh Ewald for long-range electrostatics\n\
00181         pme_order       = 4     ; cubic interpolation\n\
00182         fourierspacing  = 0.16  ; grid spacing for FFT\n\
00183         ; Temperature coupling is on\n\
00184         tcoupl      = V-rescale             ; modified Berendsen thermostat\n\
00185         tc-grps     = Protein Non-Protein   ; two coupling groups - more accurate\n\
00186         tau_t       = 0.1     0.1           ; time constant, in ps\n\
00187         ref_t       = {1:f}       {1:f}           ; reference temperature, one for each group, in K\n\
00188         ; Pressure coupling is off\n\
00189         pcoupl      = no        ; no pressure coupling in NVT\n\
00190         ; Periodic boundary conditions\n\
00191         pbc     = xyz           ; 3-D PBC\n\
00192         ; Dispersion correction\n\
00193         DispCorr    = EnerPres  ; account for cut-off vdW scheme\n\
00194         ; Velocity generation\n\
00195         gen-vel     = yes       ; assign velocities from Maxwell distribution\n\
00196         gen-temp    = {1:f}     ; temperature for Maxwell distribution\n\
00197         gen-seed    = {2:d}     ; generate a random seed".format(name, temp, seed, steps)
00198     return mdp_str
00199
00200
References gen_dirs().
Referenced by gen_dirs().
Here is the call graph for this function:
```

Here is the caller graph for this function:



3.10 generate_total_best_tables Namespace Reference

Functions

- def main ()
- def plot_tables (list filenames_db, str out_file, list table_names)

3.10.1 Function Documentation

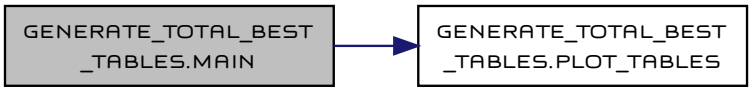
3.10.1.1 main() def generate_total_best_tables.main ( )

Definition at line 49 of file generate_total_best_tables.py.

```
00049 def main():
00050
00051     # ############ TRP #########
00052     # filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_amber_trp_300_2_fixed.sqlite3', 'results_charm_trp_300_fixed.sqlite3',
      'results_charm_trp_300_2_fixed.sqlite3', 'results_gromos_trp_300_fixed.sqlite3', 'results_gromos_trp_300_2_fixed.sqlite3',
      'results_opls_trp_300_fixed.sqlite3', 'results_opls_trp_300_2_fixed.sqlite3']
00053     # table_names = ['amber trp 1', 'amber trp 2', 'charm trp 1', 'charm trp 2', 'gromos trp 1', 'gromos trp 2', 'opls trp 1', 'opls trp 2']
00054     # outfile = 'all_trp_all'
00055     # plot_tables(filenames_db, outfile, table_names)
00056
00057     # filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_charm_trp_300_fixed.sqlite3', 'results_gromos_trp_300_fixed.sqlite3',
      'results_opls_trp_300_fixed.sqlite3']
00058     # table_names = ['amber trp 1', 'charm trp 1', 'gromos trp 1', 'opls trp 1']
00059     # outfile = 'all_trp_1'
00060     # plot_tables(filenames_db, outfile, table_names)
00061
00062     # filenames_db = ['results_amber_trp_300_2_fixed.sqlite3', 'results_charm_trp_300_2_fixed.sqlite3',
      'results_gromos_trp_300_2_fixed.sqlite3', 'results_opls_trp_300_2_fixed.sqlite3']
00063     # table_names = ['amber trp 2', 'charm trp 2', 'gromos trp 2', 'opls trp 2']
00064     # outfile = 'all_trp_2'
00065     # plot_tables(filenames_db, outfile, table_names)
00066
00067     # filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_amber_trp_300_2_fixed.sqlite3']
00068     # table_names = ['amber trp 1', 'amber trp 2']
00069     # outfile = 'amber_trp'
00070     # plot_tables(filenames_db, outfile, table_names)
00071
00072     # filenames_db = ['results_charm_trp_300_fixed.sqlite3', 'results_charm_trp_300_2_fixed.sqlite3']
00073     # table_names = ['charm trp 1', 'charm trp 2']
00074     # outfile = 'charm_trp'
00075     # plot_tables(filenames_db, outfile, table_names)
00076
00077     # filenames_db = ['results_gromos_trp_300_fixed.sqlite3', 'results_gromos_trp_300_2_fixed.sqlite3']
00078     # table_names = ['gromos trp 1', 'gromos trp 2']
00079     # outfile = 'gromos_trp'
00080     # plot_tables(filenames_db, outfile, table_names)
00081
00082     # filenames_db = ['results_opls_trp_300_fixed.sqlite3', 'results_opls_trp_300_2_fixed.sqlite3']
00083     # table_names = ['opls trp 1', 'opls trp 2']
00084     # outfile = 'opls_trp'
00085     # plot_tables(filenames_db, outfile, table_names)
00086
00087
00088     # # ############ VIL #########
00089     # filenames_db = ['results_amber_vil_300.sqlite3', 'results_charm_vil_300.sqlite3', 'results_gromos_vil_300.sqlite3',
      'results_opls_vil_300.sqlite3']
00090     # table_names = ['amber vil', 'charm vil', 'gromos vil', 'opls vil']
00091     # outfile = 'all_vil'
00092     # plot_tables(filenames_db, outfile, table_names)
00093
00094
00095     # ############ GB1 #########
00096     filenames_db = ['results_amber_gb1_300.sqlite3', 'results_charm_gb1_300.sqlite3', 'results_gromos_gb1_300.sqlite3',
      'results_opls_gb1_300.sqlite3']
00097     table_names = ['amber gb1', 'charm gb1', 'gromos gb1', 'opls gb1']
00098     outfile = 'all_gb1'
00099     plot_tables(filenames_db, outfile, table_names)
00100
00101
00102 def plot_tables(filenames_db: list, out_file: str, table_names: list):
00103     """
00104
00105     Args:
00106         list filenames_db:
00107         str  out_file:
00108         list table_names:
```

References plot_tables().

Here is the call graph for this function:



3.10.1.2 plot_tables()

```
def generate_total_best_tables.plot_tables (
        list  filenames_db,
        str  out_file,
        list  table_names )
```

Definition at line 109 of file generate_total_best_tables.py.

```
00109     """
00110     out_file = '{}.tex'.format(out_file)
00111     con_arr = [lite.connect(db_name, check_same_thread=False, isolation_level=None) for db_name in filenames_db]
00112     cur_arr = [con.cursor() for con in con_arr]
00113     metrics = ["RMSD", "ANGL", "AND_H", "AND", "XOR"]
00114     metrics_tab = ["RMSD", "ANGL", "AND\\_H", "AND", "XOR"]
00115     allowed_faild = [20, 10, 5, 5, 10]
00116 
00117     total_promotions = list()
00118     prom_during_metric = list()
00119     total_steps_during_metric = list()
00120     for db_name in filenames_db:
00121         con = lite.connect(db_name, check_same_thread=False, isolation_level=None)
00122         cur = con.cursor()
00123         qry = "select count(1) from log where operation='prom_0' "  # total
00124         result = cur.execute(qry)
00125         total_promotions.append(result.fetchone()[0])
00126         personal_res = list()
00127         personal_total_steps = list()
00128         for partial_metr in metrics:
00129             qry = "select count(1) from log where operation='prom_0' and cur_metr='{}'".format(partial_metr)
00130             result = cur.execute(qry)
00131             personal_res.append(result.fetchone()[0])
00132         prom_during_metric.append(personal_res)
00133         for partial_metr in metrics:
00134             qry = "select count(1) from log where dst='VIZ' and cur_metr='{}'".format(partial_metr)
00135             result = cur.execute(qry)
00136             personal_total_steps.append(result.fetchone()[0])
00137         total_steps_during_metric.append(personal_total_steps)
00138         del personal_res
00139         con.close()
00140     del result, qry, partial_metr, db_name, cur, con, con_arr, cur_arr, personal_total_steps
00141     a = 8
00142     # for i in range(len(total_promotions)):
00143     #     print('{}\t{}\t{}\t{}\t{}\t{}\t{}\t{}'.format('metr', 'per_s', 'tot_s', 'pe/to', 'pe/al   ', 'to/al', 'to/al ', 'fa'))
00144     #     for j in range(len(prom_during_metric[i])):
00145     #         print('{}:\t{:d}  \t{:d}\t{:3.2f}\t{:3.2f} \t{:3.2f}\t{:3.2f}\t{:d}'.format(metrics[j], prom_during_metric[i][j],
       total_steps_during_metric[i][j], 100*prom_during_metric[i][j]/total_steps_during_metric[i][j], 100 * total_steps_during_metric[i][j] /
       total_promotions[i], 100 * prom_during_metric[i][j] / total_promotions[i], 100 * prom_during_metric[i][j] / sum(total_promotions),
       allowed_faild[j]))
00146     #     print('t: {}\t{}'.format(total_promotions[i], sum(total_steps_during_metric[i])))
00147     #     print()
00148 
00149 
00150     with open(out_file, 'w') as tex_table:
00151         tex_table.writelines(['\\begin{table}[h]\n', '\centering\n', '\sisetup{table-align-text-post=false}\n',
       '\\begin{tabular}{@{}|l|S[table-format=2.0]\
00152 
       |S[table-format=3.0]\
00153 
       |S[table-format=6]\
00154 
       |S[table-format=3.3]\
00155 
       |S[table-format=3.2]\
```

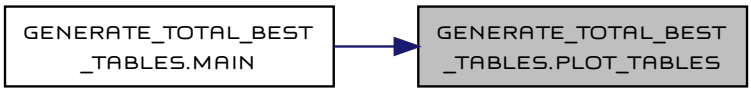
```
00156
    |S[table-format=3.3]\
00157
    |S[table-format=1.2]\
00158
    |@{}}\n’])
00159       for i in range(len(total_promotions)):
00160           tex_table.write(”)
00161
00162           tex_table.write(’\multicolumn{{8}}{{c}}{{{}}} \\\\ \n’.format(table_names[i]))
00163           tex_table.write(’\\hline\n’)
00164           tex_table.write(’{} & {} & {} & {} & {} & {} & {} & {} \\\\ \n’.format(          ”,      ’{allowed}’, ’{percent}’, ’{metric}’,
    ’{percent}’, ’{promotions}’, ’{percent of}’, ’{promotions}’))
00165           tex_table.write(’{} & {} & {} & {} & {} & {} & {} & {} \\\\ \\hline\n’.format(’{metric}’, ’{fails}’, ’{allowed}’, ’{total
    steps}’, ’{steps}’, ’{per metric}’, ’{promotions}’, ’{per 1000 steps}’))
00166           for j in range(len(prom_during_metric[i])):
00167               tex_table.write(’{:s} & {:d} & {:3.0f}\\si{{\\percent}} & {:d} & {:3.2f}\\si{{\\percent}} & {} & {:3.2f}\\si{{\\percent}}  &
    {:3.2f} \\\\ \\hline\n’.format(
00168                   metrics_tab[j],
00169                   allowed_faild[j],
00170                   100*allowed_faild[j]/sum(allowed_faild),
00171                   total_steps_during_metric[i][j],
00172                   100*total_steps_during_metric[i][j]/sum(total_steps_during_metric[i]),
00173                   prom_during_metric[i][j],
00174                   100 * prom_during_metric[i][j]/sum(prom_during_metric[i]),
00175                   1000 * prom_during_metric[i][j]/total_steps_during_metric[i][j]))
00176           tex_table.write(’{:s} & {:d} & {:3.0f}\\si{{\\percent}} & {:d} & {:3.2f}\\si{{\\percent}} & {} & {:3.2f}\\si{{\\percent}} &
    {:3.2f}\\\\ \\hline \\hline \n’.format(’total’, sum(allowed_faild), 100, sum(total_steps_during_metric[i]), 100, sum(prom_during_metric[i]),
    100, 1000 * sum(prom_during_metric[i])/sum(total_steps_during_metric[i])))
00177       tex_table.writelines([’\\end {tabular}\n’, ’\\caption{{{}}}\n’.format(’{}’.format(’, ’.join(table_names))), ’\\end {table}\n’])
00178
00179       tex_table.write(’\n\n\n’)
00180
00181
00182       total_steps_during_metric_comb = [sum(x) for x in zip(*total_steps_during_metric)]
00183       prom_during_metric_comb = [sum(x) for x in zip(*prom_during_metric)]
00184
00185       tex_table.writelines([’\\begin{table}[h]\n’, ’\centering\n’, ’\sisetup{table-align-text-post=false}\n’,
    ’\\begin{tabular}{@{}|l|S[table-format=2.0]\
00186
    |S[table-format=3.0]\
00187
    |S[table-format=6]\
00188
    |S[table-format=3.3]\
00189
    |S[table-format=3.2]\
00190
    |S[table-format=3.3]\
00191
    |S[table-format=1.2]\
00192
    |@{}}\n’, ’\\hline\n’])
00193       tex_table.write(’{} & {} & {} & {} & {} & {} & {} & {} \\\\ \n’.format(”, ’{allowed}’, ’{percent}’, ’{metric}’, ’{percent}’,
    ’{promotions}’, ’{percent of}’, ’{promotions}’))
00194       tex_table.write(’{} & {} & {} & {} & {} & {} & {} & {} \\\\ \\hline\n’.format(’{metric}’, ’{fails}’, ’{allowed}’, ’{total steps}’,
    ’{steps}’, ’{per metric}’, ’{promotions}’, ’{per 1000 steps}’))
00195       for j in range(len(prom_during_metric_comb)):
00196           tex_table.write(’{:s} & {:d} & {:3.0f}\\si{{\\percent}} & {:d} & {:3.2f}\\si{{\\percent}} & {} & {:3.2f}\\si{{\\percent}}  &
    {:3.2f} \\\\ \\hline\n’.format(
00197               metrics_tab[j],
00198               allowed_faild[j],
00199               100*allowed_faild[j]/sum(allowed_faild),
00200               total_steps_during_metric_comb[j],
00201               100*total_steps_during_metric_comb[j]/sum(total_steps_during_metric_comb),
00202               prom_during_metric_comb[j],
00203               100 * prom_during_metric_comb[j]/sum(prom_during_metric_comb),
00204               1000 * prom_during_metric_comb[j]/total_steps_during_metric_comb[j]))
00205       tex_table.write(’{:s} & {:d} & {:3.0f}\\si{{\\percent}} & {:d} & {:3.2f}\\si{{\\percent}} & {} & {:3.2f}\\si{{\\percent}} &
    {:3.2f}\\\\ \\hline \\hline \n’.format(’total’, sum(allowed_faild), 100, sum(total_steps_during_metric_comb), 100,
    sum(prom_during_metric_comb), 100, 1000 * sum(prom_during_metric_comb)/sum(total_steps_during_metric_comb)))
00206       tex_table.writelines([’\\end {tabular}\n’, ’\\caption{{{}}}\n’.format(’Summary of ({})’.format(’, ’.join(table_names))), ’\\end
    {table}\n’])
00207
00208
00209
00210       tex_table.write(’\n\n\n’)
00211       norm_coef = [min(allowed_faild)/elem   for elem in allowed_faild]
00212       allowed_faild = [elem * norm_coef[k] for k, elem in enumerate(allowed_faild)]
00213
```

```
00214         tex_table.writelines(['\\begin{table}[h]\n', '\centering\n', '\sisetup{table-align-text-post=false}\n',
      '\\begin{tabular}{@{}|l|S[table-format=2.0]\
00215
      |S[table-format=3.0]\
00216
      |S[table-format=6]\
00217
      |S[table-format=3.3]\
00218
      |S[table-format=3.2]\
00219
      |S[table-format=3.3]\
00220
      |S[table-format=1.2]\
00221
      |@{}}\n'])
00222
00223         for i in range(len(total_promotions)):
00224             total_steps_during_metric[i] = [elem * norm_coef[k] for k, elem in enumerate(total_steps_during_metric[i])]
00225             prom_during_metric[i] = [elem * norm_coef[k] for k, elem in enumerate(prom_during_metric[i])]
00226
00227             tex_table.write('\multicolumn{{8}}{{c}}{{{}}} \\\\ \n'.format(table_names[i]))
00228             tex_table.write('\\hline\n')
00229             tex_table.write('{} & {} & {} & {} & {} & {} & {} & {} \\\\ \n'.format('', '{allowed}', '{percent}', '{metric}', '{percent}',
      '{promotions}', '{percent of}', '{promotions}'))
00230             tex_table.write(
00231                 '{} & {} & {} & {} & {} & {} & {} & {} \\\\ \\hline\n'.format('{metric}', '{fails}', '{allowed}', '{total steps}', '{steps}',
      '{per metric}', '{promotions}', '{per 1000 steps}'))
00232             for j in range(len(prom_during_metric[i])):
00233                 tex_table.write('{:s} & {:3.0f} & {:2.0f}\\si{{\\percent}} & {:3.0f} & {:3.2f}\\si{{\\percent}} & {} &
      {:3.2f}\\si{{\\percent}} & {:3.2f}  \\\\ \\hline\n'.format(
00234                     metrics_tab[j],
00235                     allowed_faild[j],
00236                     100*allowed_faild[j]/sum(allowed_faild),
00237                     total_steps_during_metric[i][j],
00238                     100*total_steps_during_metric[i][j]/sum(total_steps_during_metric[i]),
00239                     prom_during_metric[i][j],
00240                     100 * prom_during_metric[i][j]/sum(prom_during_metric[i]),
00241                     1000 * prom_during_metric[i][j]/total_steps_during_metric[i][j]))
00242             tex_table.write('{:s} & {:3.0f} & {:2.0f}\\si{{\\percent}} & {:2.0f} & {}\\si{{\\percent}} & {} & {}\\si{{\\percent}} &
      {:3.2f}\\\\ \\hline \\hline \n'.format('total', sum(allowed_faild), 100, sum(total_steps_during_metric[i]), 100, sum(prom_during_metric[i]),
      100, 1000 * sum(prom_during_metric[i])/sum(total_steps_during_metric[i])))
00243         tex_table.writelines(['\\end {tabular}\n', '\\caption{{{}}}\n'.format('Normalized ' + '{}'.format(', '.join(table_names))), '\\end
      {table}\n'])
00244
00245         total_steps_during_metric_comb = [sum(x) for x in zip(*total_steps_during_metric)]
00246         prom_during_metric_comb = [sum(x) for x in zip(*prom_during_metric)]
00247
00248         tex_table.write('\n\n\n')
00249
00250         tex_table.writelines(['\\begin{table}[h]\n', '\centering\n', '\sisetup{table-align-text-post=false}\n',
      '\\begin{tabular}{@{}|l|S[table-format=2.0]\
00251
      |S[table-format=3.0]\
00252
      |S[table-format=6]\
00253
      |S[table-format=3.3]\
00254
      |S[table-format=3.2]\
00255
      |S[table-format=3.3]\
00256
      |S[table-format=1.2]\
00257                                                                                                                                                |@{}}\n',
      '\\hline\n'])
00258         tex_table.write('{} & {} & {} & {} & {} & {} & {} & {} \\\\ \n'.format('', '{allowed}', '{percent}', '{metric}', '{percent}',
      '{promotions}', '{percent of}', '{promotions}'))
00259         tex_table.write('{} & {} & {} & {} & {} & {} & {} & {} \\\\ \\hline\n'.format('{metric}', '{fails}', '{allowed}', '{total steps}',
      '{steps}', '{per metric}', '{promotions}', '{per 1000 steps}'))
00260         for j in range(len(prom_during_metric_comb)):
00261             tex_table.write('{:s} & {:3.0f} & {:2.0f}\\si{{\\percent}} & {:3.0f} & {:3.2f}\\si{{\\percent}} & {} & {:3.2f}\\si{{\\percent}}
      & {:3.2f}  \\\\ \\hline\n'.format(
00262                 metrics_tab[j],
00263                 allowed_faild[j],
00264                 100*allowed_faild[j]/sum(allowed_faild),
00265                 total_steps_during_metric_comb[j],
00266                 100*total_steps_during_metric_comb[j]/sum(total_steps_during_metric_comb),
00267                 prom_during_metric_comb[j],
00268                 100 * prom_during_metric_comb[j]/sum(prom_during_metric_comb),
00269                 1000 * prom_during_metric_comb[j]/total_steps_during_metric_comb[j]))
```

```
00270         tex_table.write('{:s} & {:3.0f} & {:2.0f}\\si{{\\percent}} & {:2.0f} & {}\\si{{\\percent}} & {} & {}\\si{{\\percent}} & {:3.2f}\\\\\
      \\hline \\hline \n'.format('total', sum(allowed_faild), 100, sum(total_steps_during_metric_comb), 100, sum(prom_during_metric_comb), 100,
      1000 * sum(prom_during_metric_comb)/sum(total_steps_during_metric_comb)))
00271         tex_table.writelines(['\\end {tabular}\n', '\\caption{{{}}}\n'.format('Normalized ' + 'summary of ({})'.format(',
      '.join(table_names))), '\\end {table}\n'])
00272
00273
00274
00275
00276         #     tex_table.writelin('\\hline')
00277         #     qry = "select count(1) from log where operation='prom_0' "
00278         #     result_arr = cur.execute(qry) cur_arr
00279         #     total_prom = [res.fetchone() for res in result_arr]
00280         #     for partial_metr in ["RMSD", "ANGL", "AND_H", "AND", "XOR"]:
00281         #         qry = "select count(1) from log where operation='prom_0' and cur_metr='{}'".format(partial_metr)
00282         #         result_arr = [cur.execute(qry) for cur in cur_arr]
00283         #         fetched_one_arr = [res.fetchone() for res in result_arr]
00284         #         tex_table.writelin('\\hline')
00285         #     tex_table.writelin('\\hline')
00286         #
00287         # tex_table.writelines(['\\caption\{{}\}'.format('some caption here'), '\\end {tabular}', '\\end {table}'])
00288
00289
```

Referenced by main().

Here is the caller graph for this function:



3.11 GMDA_main Namespace Reference

Functions

- list queue_rebuild (list process_queue, list open_queue_to_rebuild, dict node_info, float cur_mult, str new_metr_name, bool sep_proc=True)

  Resorts the queue according to the new metric.
- int get_atom_num (str ndx_file)

  Computes number of atoms in the particular index file.
- tuple parse_hostnames (int seednum, str hostfile='hostfile')

  Spreads the load among the hosts found in the hostfile.
- tuple compute_on_local_machine (list cpu_map, list seed_list, str cur_name, str past_dir, str work_dir, dict seed_dirs, str topol_↩ file_init, str ndx_file_init, list prev_runs_files, str old_name_digest)

  This version is optimised for usage on one machine with tMPI (see GROMACS docs).
- tuple compute_with_mpi (list seed_list, str cur_name, str past_dir, str work_dir, dict seed_dirs, str topol_file_init, str ndx_file_↩ init, list prev_runs_files, str old_name_digest, int tot_seeds, list hostnames, list ncores, bool sched=False, int ntomp=1)

  This version is optimised for usage on more than one machine with tMPI and/or MPI.
- bool check_in_queue (list queue, str elem_hash)

  Checks whether elements with provided hash exists in the queue.
- list second_chance (list open_queue, list visited_queue, str best_so_far_name, str cur_metric, dict main_dict, int node_max_att, str cur_metric_name, str best_so_far, float tol_error, float greed_mult)

  Typically executed during the seed change.
- list check_dupl (str name_to_check, list visited_queue)

  This function is just a detector of duplicates.
- NoReturn GMDA_main (list prev_runs_files, str past_dir, mp.JoinableQueue print_queue, mp.JoinableQueue db_input_queue, mp.JoinableQueue copy_queue, mp.JoinableQueue rm_queue, int tot_seeds=4)

  This is the main loop.

Variables

- int MAX_ITEMS_TO_HANDLE = 50000

3.11.1 Function Documentation

3.11.1.1 check_dupl()

```
list GMDA_main.check_dupl (
        str  name_to_check,
        list  visited_queue )
```

This function is just a detector of duplicates.
Main source of dupplicates is when the algorithme gives the second chance to the same seed, but does not use it. This function checks whether specific name was used recently

name_to_check: name that is about to be sampled
visited_queue: all previously used names

Returns

:return: True if name was used recently, otherwise False

Definition at line 514 of file GMDA_main.py.

```
00514     """
00515     arr = [name[2] for name in visited_queue]
00516     if name_to_check in arr:
00517         print("Duplicate found in {} last elements, index: {}\nelem:{}".format(len(arr), arr.index(name_to_check), name_to_check))
00518         return True
00519     return False
00520
00521
```

Referenced by GMDA_main().
Here is the caller graph for this function:

3.11.1.2 check_in_queue()

```
bool GMDA_main.check_in_queue (
        list  queue,
        str  elem_hash )
```

Checks whether elements with provided hash exists in the queue.

list queue: specific queue to check
str elem_hash: name to find in the queue

Returns

:return: True if element found, False otherwise :rtype: bool

Definition at line 433 of file GMDA_main.py.

```
00433     """
00434     for elem in queue:
00435         if elem[2] == elem_hash:
00436             return True
00437     return False
00438
00439
```

Referenced by second_chance().
Here is the caller graph for this function:

3.11.1.3 compute_on_local_machine()

```
tuple GMDA_main.compute_on_local_machine (
        list cpu_map,
        list seed_list,
        str cur_name,
        str past_dir,
        str work_dir,
        dict seed_dirs,
        str topol_file_init,
        str ndx_file_init,
        list prev_runs_files,
        str old_name_digest )
```

This version is optimised for usage on one machine with tMPI (see GROMACS docs).
Performs check whether requested simulation was completed in the past. If so (and all requested files exist), we skip the computation, otherwise we start the sequence of events that prepare and run the simulation in the separate process. I was playing with better core distribution, but it did not work well, since GROMACS may complain when you assign odd number of cores, or when 14 cores does not work, but 12 and 16 are fine. What I know fo sure that powers of 2 work the best until 128 cores, but we do not have so many cores on one machine. Two machines are worse than one (yes, 64+64 is slower than 64, same with 32+32) - maybe Infiniband can help, but we do not have one. Additionally, I commented prev_runs - it just uses more RAM without giving any significant speedup.

list cpu_map: number of cores for particular task (seed)
list seed_list: list of current seeds
str cur_name: name of the current node (prior path constructed from seed names s_0_1_4)
str past_dir: path to the directory with prior computations
str work_dir: path to the directory where seed dirs reside
dict seed_dirs: dict which contains physical path to the directory where simulation with particular seed is performed
str topol_file_init: .top - topology of the initial (unfolded) conformation
str ndx_file_init: .ndx - index of the protein atoms of the unfolded conformation
list prev_runs_files: information about all previously generated files in ./past directory
str old_name_digest: digest of the current name

Returns

:return: array of PIDs to join them later and allow some more parallel computation, hash names, simulation names. :rtype: tuple

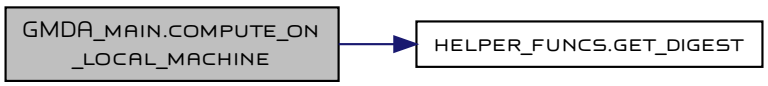
Returns PIDs and new filenames. PIDs - to join processes later.

Definition at line 271 of file GMDA_main.py.

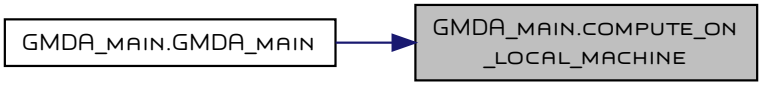
```
00271     Returns: PIDs and new filenames. PIDs - to join processes later.
00272     """
00273     files_for_trjcat = list()
00274     recent_filenames = list()
00275     pid_arr = list()
00276     # global extra_past
00277     # recent_n2d = dict ()
00278     # recent_d2n = dict ()
00279     for i, exec_group in enumerate(cpu_map):
00280         saved_cores = 0
00281         for cur_group_sched in exec_group:
00282             cores, seed_2_process = cur_group_sched
00283             seed_2_process = seed_list[seed_2_process]
00284             new_name = '{}_{}'.format(cur_name, seed_2_process)
00285             seed_digest_filename = get_digest(new_name)
00286             # recent_n2d[new_name] = seed_digest_filename
00287             # recent_d2n[seed_digest_filename] = new_name
00288             xtc_filename = '{}.xtc'.format(seed_digest_filename)
00289             gro_filename = '{}.gro'.format(seed_digest_filename)
00290 
00291             files_for_trjcat.append(os.path.join(past_dir, xtc_filename))
00292             # if xtc_filename in prev_runs_files and gro_filename in prev_runs_files:
00293             # # if os.path.exists(os.path.join('./past', xtc_filename)) and os.path.exists(os.path.join('./past', gro_filename)):
00294             #     saved_cores += cores  # not fair, but short TODO: write better logic for cores remapping
00295             #     recent_filenames.append(xtc_filename)
00296             #     recent_filenames.append(gro_filename)
00297             #     continue
00298             # else:
00299             if not (os.path.exists(os.path.join(past_dir, xtc_filename)) and os.path.exists(os.path.join(past_dir, gro_filename))): #\
00300                 # and not (os.path.exists(os.path.join(extra_past, xtc_filename)) and os.path.exists(os.path.join(extra_past, gro_filename))):
00301                 md_process = None
00302                 md_process = mp.Process(target=make_a_step,
00303                                         args=(work_dir, seed_2_process, seed_dirs, topol_file_init, ndx_file_init,
00304                                               seed_digest_filename, old_name_digest, past_dir, cores + saved_cores))
00305                 md_process.start()
00306                 # print('Process started :{} pid:{} alive:{} ecode:{} with next param: s:{}, pd:{}, cor:{}'.format(md_process.name,
00307                 # md_process.pid, md_process.is_alive(), md_process.exitcode, seed_2_process, past_dir, cores+saved_cores))
00308                 pid_arr.append(md_process)
00309                 # make_a_step(work_dir, seed_2_process, seed_dirs, seed_list, topol_file, ndx_file, name_2_digest_map,
00310                 # cur_job_name, past_dir, cores+saved_cores)
00311                 saved_cores = 0
00312                 # print('md_process{} '.format(seed_2_process), end="")
```

```
00313                 # recent_filenames.append(xtc_filename)
00314                 # recent_filenames.append(gro_filename)
00315         if i is not len(cpu_map) - 1:  # if it is not the last portion of threads then wait for completion
00316             [proc.join() for proc in pid_arr]
00317 
00318     # combine prev_step and goal to compute two dist in one pass
00319     # rm_queue.join()  # make sure that queue is empty (all files were deleted)
00320 
00321     # Test code for multiprocessing check. There was a problem with python3.4 and old sqlite (too many parallel
00322     # connections when reusing past results).
00323     # [proc.join(timeout=90) for proc in pid_arr]
00324     # if len(pid_arr):
00325     #     print('Proc arr is not empty:', end=' ')
00326     #     while True:
00327     #         proc_stil_running = 0
00328     #         for cur_group_sched in pid_arr:
00329     #             print('waiting for name:{} pid:{} alive:{} ecode:{}'.format(cur_group_sched.name,
00330     #             cur_group_sched.pid, cur_group_sched.is_alive(), cur_group_sched.exitcode))
00331     #             cur_group_sched.join(timeout=40)
00332     #             if cur_group_sched.exitcode is not None:
00333     #                 proc_stil_running += 1
00334     #         if proc_stil_running == len(pid_arr):
00335     #             print('Done.')
00336     #             break
00337 
00338     # if len(pid_arr):
00339     #     print('j{} '.format(len(pid_arr)), end="")
00340     return pid_arr, files_for_trjcat, recent_filenames, None, None  # recent_n2d, recent_d2n
00341 
00342 
```

References helper_funcs.get_digest().
Referenced by GMDA_main().
Here is the call graph for this function:



Here is the caller graph for this function:



3.11.1.4 compute_with_mpi()

```
tuple  GMDA_main.compute_with_mpi (
        list  seed_list,
        str  cur_name,
        str  past_dir,
        str  work_dir,
        dict  seed_dirs,
        str  topol_file_init,
        str  ndx_file_init,
        list  prev_runs_files,
        str  old_name_digest,
        int  tot_seeds,
        list  hostnames,
```

```
                list  ncores,
                bool  sched =  False,
                int  ntomp =  1 )
```

This version is optimised for usage on more than one machine with tMPI and/or MPI.
If you use scheduler and know exactly how many cores each machine has - supply correct hostfile and use tMPI on each machine with OMP. If you use scheduler without option to choose specific machine - use version without scheduler or local version (depends on your cluster implementation).
Performs check whether requested simulation was completed in the past. If so (and all requested files exist), we skip the computation, otherwise we start the sequence of events that prepare and run the simulation in the separate process. I was playing with better core distribution, but it did not work well, since GROMACS may complain when you assign odd number of cores, or when 14 cores does not work, but 12 and 16 are fine. What I know fo sure that powers of 2 work the best until 128 cores, but we do not have so many cores on one machine. Two machines are worse than one (yes, 64+64 is slower than 64, same with 32+32) - maybe InfiniBand can help, but we do not have one. Additionally, I commented prev_runs - it just uses more RAM without giving any significant speedup.

```
    list  seed_list:   list  of current seeds
    str  cur_name: name of the current node (prior path constructed from seed names s_0_1_4)
    str  past_dir: path to the directory with prior computations
    strwork_dir: path to the directory where seed dirs reside
    dict   seed_dirs:  dict   which contains physical path to the directory where simulation with particular seed is performed
    str  topol_file_init:  .top - topology of the initial (unfolded) conformation
    str  ndx_file_init: .ndx - index of the protein atoms of the initial (unfolded) conformation
    list  prev_runs_files: information about all previously generated files in ./past directory
    str  old_name_digest: digest of the current name
    int  tot_seeds: total number of seeds, controversial optimisation.
    list  hostnames: correct names/IPs of the hosts
    int  ncores: number of cores on each host
    bool   sched: secelts proper make_a_step version
    int  ntomp: how many OMP threads use during the MD simulation (2-4 is the optimal value on 32-64 core hosts)
```

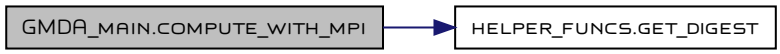
Returns

:return: array of PIDs to join them later and allow some more parallel computation, hash names, simulation names. :rtype: tuple

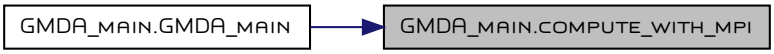
PIDs and new filenames. PIDs - to join processes later.
Definition at line 377 of file GMDA_main.py.

```
00377
00378     PIDs and new filenames. PIDs - to join processes later.
00379     """
00380     # if os.path.exists(os.path.join(os.getcwd(), 'local.comp')):
00381     #     hostnames = [('Perseus', )]*tot_seeds
00382     gc.collect()
00383     files_for_trjcat = list()
00384     recent_filenames = list()
00385     pid_arr = list()
00386     # recent_n2d = dict ()
00387     # recent_d2n = dict ()
00388     for i in range(tot_seeds):
00389         seed_2_process = seed_list[i]
00390         new_name = '{}_{}'.format(cur_name, seed_2_process)
00391         seed_digest_filename = get_digest(new_name)
00392         # recent_n2d[new_name] = seed_digest_filename
00393         # recent_d2n[seed_digest_filename] = new_name
00394         xtc_filename = '{}.xtc'.format(seed_digest_filename)
00395         gro_filename = '{}.gro'.format(seed_digest_filename)
00396
00397         # if os.path.exists(os.path.join(extra_past, xtc_filename)) and os.path.exists(os.path.join(extra_past, gro_filename)):
00398         #     files_for_trjcat.append(os.path.join(extra_past, xtc_filename))
00399         # else:
00400         files_for_trjcat.append(os.path.join(past_dir, xtc_filename))
00401
00402         # if xtc_filename not in prev_runs_files or gro_filename not in prev_runs_files:
00403         if not (os.path.exists(os.path.join(past_dir, xtc_filename)) and os.path.exists(os.path.join(past_dir, gro_filename))): # \
00404                 # and not (os.path.exists(os.path.join(extra_past, xtc_filename)) and os.path.exists(os.path.join(extra_past, gro_filename))):
00405             # make_a_step2(work_dir, seed_2_process, seed_dirs, topol_file_init, ndx_file_init, seed_digest_filename, old_name_digest,
00406             # past_dir, hostnames[i], ncores[i])
00407             if sched:
00408                 md_process = mp.Process(target=make_a_step3,
00409                                         args=(work_dir, seed_2_process, seed_dirs, topol_file_init, ndx_file_init,
00410                                               seed_digest_filename, old_name_digest, past_dir, int(ncores/tot_seeds), ntomp))
00411             else:
00412                 md_process = mp.Process(target=make_a_step2,
00413                                         args=(work_dir, seed_2_process, seed_dirs, topol_file_init, ndx_file_init,
00414                                               seed_digest_filename, old_name_digest, past_dir, hostnames[i], ncores[i]))
00415             md_process.start()
00416             pid_arr.append(md_process)
00417         recent_filenames.append(xtc_filename)
00418         recent_filenames.append(gro_filename)
00419
00420     return pid_arr, files_for_trjcat, recent_filenames, None, None  # recent_n2d, recent_d2n
```

```
00421
00422
References helper_funcs.get_digest().
Referenced by GMDA_main().
Here is the call graph for this function:
```




```
Here is the caller graph for this function:
```



3.11.1.5 get_atom_num()

```
int GMDA_main.get_atom_num (
        str  ndx_file )
```

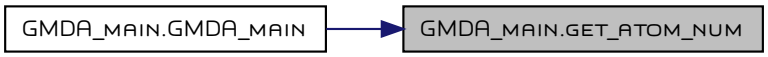
Computes number of atoms in the particular index file.

str ndx_file: .ndx - index of the protein atoms of the current conformation.

Returns

:return: number of atoms in the .ndx file. :rtype: int

```
Definition at line 210 of file GMDA_main.py.
00210     """
00211     with open(ndx_file, 'r') as index_file:
00212         index_file.readline()  # first line is the comment - skip it
00213         indices = index_file.read().strip()
00214     elems = indices.split()
00215     atom_num = len(elems)
00216     return atom_num
00217
00218
Referenced by GMDA_main().
Here is the caller graph for this function:
```



3.11.1.6 GMDA_main()

```
NoReturn  GMDA_main.GMDA_main (
        list  prev_runs_files,
        str  past_dir,
        mp.JoinableQueue print_queue,
```

```
                mp.JoinableQueue db_input_queue,
                mp.JoinableQueue copy_queue,
                mp.JoinableQueue rm_queue,
                int  tot_seeds =  4 )
```

This is the main loop.
Note that it has many garbage collector calls - it can slightly reduce the performance, but also reduces total memory usage. Feel free to comment them - they do not affect the algorithm

list prev_runs_files you may see this as the list of files found before the execution.
We do not use it anymore to reduce the memory footprint.
Instead we check existence of the file separately.
str past_dir: location of all generated .gro, .xtc, metric values. Sequence of past seeds results in the unique name.
:type past_dir: str
mp.JoinableQueue print_queue: separate thread for printing operations, connected to the main process by Queue.
It helps significantly during the restart without the previously saved state:
you can query DB faster without waiting for printing operations to complete.
mp.JoinableQueue db_input_queue:
mp.JoinableQueue copy_queue: connection to the separate process that handled async copy. Should be rewriten with asyncio
mp.JoinableQueue rm_queue: connection to the separate process that handled async rm. Should be rewriten with asyncio
int tot_seeds: number of parallel seeds to be executed - very powerful knob

Returns

:return: Nothing, once stop condition is reached, looping stops and returns to the parent to join/clean other threads

Definition at line 544 of file GMDA_main.py.

```
00544         :return: Nothing, once stop condition is reached, looping stops and returns to the parent to join/clean other threads
00545     """
00546     # prev_runs_files = None  # temp action - trying to save memory
00547     print('Main process rebuild_queue_process: ', os.getpid())
00548     gc.collect()
00549     prot_dir = os.path.join(os.getcwd(), 'prot_dir')
00550     if not os.path.exists(prot_dir):
00551         os.makedirs(prot_dir)
00552     print('Prot dir: ', prot_dir)
00553     # These files has to be in prot_dir
00554     init = os.path.join(prot_dir, 'init.gro')  # initial state, will be copied into work dir, used for MD
00555     goal = os.path.join(prot_dir, 'goal.gro')  # final state, will not be used, but needed for derivation of other files
00556 
00557     topol_file_init = os.path.join(prot_dir, 'topol_unfolded.top')  # needed for MD
00558     topol_file_goal = os.path.join(prot_dir, 'topol_folded.top')  # needed for MD
00559 
00560     ndx_file_init = os.path.join(prot_dir, 'prot_unfolded.ndx')  # needed for extraction of protein data
00561     ndx_file_goal = os.path.join(prot_dir, 'prot_folded.ndx')  # needed for extraction of protein data
00562 
00563     init_bb_ndx = os.path.join(prot_dir, 'bb_unfolded.ndx')
00564     goal_bb_ndx = os.path.join(prot_dir, 'bb_folded.ndx')
00565 
00566     # These files will be generated
00567     init_xtc = os.path.join(prot_dir, 'init.xtc')  # small version, used for rmsd
00568     goal_xtc = os.path.join(prot_dir, 'goal.xtc')  # small version, used for rmsd
00569     goal_prot_only = os.path.join(prot_dir, 'goal_prot.gro')  # needed for knn_rms
00570     init_prot_only = os.path.join(prot_dir, 'init_prot.gro')  # needed for contacts
00571     # goal_bb_gro = os.path.join(prot_dir, 'goal_bb.gro')
00572     goal_bb_xtc = os.path.join(prot_dir, 'goal_bb.xtc')
00573     goal_angle_file = os.path.join(prot_dir, 'goal_angle.dat')
00574     goal_sincos_file = os.path.join(prot_dir, 'goal_sincos.dat')
00575 
00576     # cp2(os.path.join(prot_dir, 'nmr.gro'), goal)
00577     # cp2(os.path.join(prot_dir, 'md_heated.gro'), goal)
00578 
00579     # h_ndx_file = os.path.join(prot_dir, 'prot_h.ndx')
00580 
00581     # create prot_only init and goal
00582     gmx_trjconv(f=init, o=init_xtc, n=ndx_file_init)
00583     gmx_trjconv(f=goal, o=goal_xtc, n=ndx_file_goal)
00584     gmx_trjconv(f=goal, o=goal_prot_only, n=ndx_file_goal, s=goal)
00585     gmx_trjconv(f=init, o=init_prot_only, n=ndx_file_init, s=init)
00586     gmx_trjconv(f=goal, o=goal_bb_xtc, n=goal_bb_ndx, s=goal)
00587 
00588     get_bb_to_angle_mdsctk(x=goal_bb_xtc, o=goal_angle_file)
00589     get_angle_to_sincos_mdsctk(i=goal_angle_file, o=goal_sincos_file)
00590 
00591     atom_num = get_atom_num(ndx_file_init)
00592     atom_num_bb = get_atom_num(goal_bb_ndx)
00593     angl_num = 2 * int(atom_num_bb / 3) - 2  # each bb amino acid has 3 atoms, thus 3 angles, we skip 1 since it is almost always 0.
00594     # In order to make plain you need three points, this is why you loos 2 elements. Last two do not have extra atoms to form a plain
00595 
```

```
00596     with open(goal_sincos_file, 'rb') as file:
00597         initial_1d_array = np.frombuffer(file.read(), dtype=np.float64 , count=-1)
00598     goal_angles = np.reshape(initial_1d_array, (-1, angl_num*2))[0]
00599     del file, initial_1d_array
00600 
00601     cont_dist = 3.0
00602     goal_ind = get_contat_profile_mdsctk(goal_prot_only, goal_xtc, ndx_file_goal, cont_dist)[1:]  # first is total num of contacts
00603     goal_contacts = np.zeros(atom_num * atom_num, dtype=np.bool)
00604     goal_contacts[goal_ind] = True
00605     del goal_ind
00606 
00607     h_pos_goal = parse_top_for_h(topol_file_goal)
00608     h_filter_goal = np.zeros(atom_num * atom_num, dtype=np.bool)
00609     for pos in h_pos_goal:
00610         h_filter_goal[(pos - 1) * atom_num:pos * atom_num] = True
00611     del pos
00612     goal_cont_h = np.logical_and(goal_contacts, h_filter_goal)
00613 
00614     h_pos_init = parse_top_for_h(topol_file_init)
00615     h_filter_init = np.zeros(atom_num * atom_num, dtype=np.bool)
00616     for pos in h_pos_init:
00617         h_filter_init[(pos - 1) * atom_num:pos * atom_num] = True
00618     del pos
00619 
00620     # usually h_filter_init is the same as h_filter_goal since they share same force field
00621     if np.sum(np.logical_xor(h_filter_init, h_filter_goal)) > 0:
00622         print('Warning, H positions in init and goal are different')
00623     del h_pos_goal, h_pos_init
00624 
00625     cpu_pool = mp.Pool(mp.cpu_count())
00626 
00627     goal_contacts_and_sum = np.sum(goal_contacts)
00628     goal_contacts_xor_sum = get_native_contacts(goal_prot_only, [goal_xtc], ndx_file_goal, goal_contacts,
00629                                                 atom_num, cont_dist, np.logical_xor, pool=cpu_pool)[0]
00630     if goal_contacts_xor_sum != 0:
00631         raise Exception('goal.gro XOR goal.xtc is not 0 - they are different')
00632     else:
00633         del goal_contacts_xor_sum
00634     goal_contacts_and_h_sum = get_native_contacts(goal_prot_only, [goal_xtc], ndx_file_goal, goal_cont_h,
00635                                                   atom_num, cont_dist, np.logical_and, pool=cpu_pool)[0]
00636     # nat_contacts = np.sum(logic_fun(goal_contacts, init_contacts))
00637 
00638     if not os.path.exists(init_xtc) or not os.path.exists(goal_xtc) or \
00639             not os.path.exists(topol_file_init) or not os.path.exists(ndx_file_init):
00640         print('Copy initial and final state in to prot_dir')
00641         exit("Copy initial and final state in to prot_dir")
00642 
00643     work_dir = os.path.join(os.getcwd(), 'work_dir')  # either /dev/shm or os.getcwd()
00644 
00645     # counter = 0
00646     # work_dir = os.path.join('/dev/shm', 'work_dir_{}'.format(counter))  # either /dev/shm or os.getcwd()
00647     # while os.path.exists(work_dir):
00648     #     counter += 1
00649     #     work_dir = os.path.join('/dev/shm', 'work_dir_{}'.format(counter))  # either /dev/shm or os.getcwd()
00650     # del counter
00651 
00652     if not os.path.exists(work_dir):
00653         os.makedirs(work_dir)
00654     print('Work dir: ', work_dir)
00655 
00656     if not os.path.exists(past_dir):
00657         os.makedirs(past_dir)
00658 
00659     print('Past dir: ', past_dir)
00660 
00661     simulation_temp = 300
00662 
00663     print('Information about the protein:\nIt contains {} atoms and {} hydrogen contacts'
00664           '\n{} phipsi angles is going to be used as for angle distance'
00665           '\nthere are {} protein-protein contacts with distance {}A\nand {} protein-protein-h contacts with distance {}A.'
00666           '\nSimulation temp is set to {}K'
00667           ''.format(atom_num, np.sum(goal_cont_h), angl_num, goal_contacts_and_sum, cont_dist,
00668                     goal_contacts_and_h_sum, cont_dist, simulation_temp))
00669 
00670     seed_start = 0
00671     seed_list = list(range(seed_start, tot_seeds+seed_start))
00672     del seed_start
00673     seed_dirs = get_seed_dirs(work_dir, seed_list, simulation_temp)
00674     # rm_seed_dirs(seed_dirs)
00675 
00676     if os.path.exists(os.path.join(os.getcwd(), 'local.comp')):
```

```
00677         use_mpi = False
00678     else:
00679         use_mpi = True
00680 
00681     scheduler = False
00682     if scheduler:
00683         use_mpi = True
00684         core_map = 16
00685         nomp = 2
00686         hostnames = False
00687     else:
00688         nomp = False
00689         if use_mpi:
00690             hostnames, core_map = parse_hostnames(tot_seeds)
00691         else:
00692             cpu_map = create_core_mapping(nseeds=tot_seeds)
00693             hostnames = False
00694 
00695 
00696     metric_names =      ['RMSD', 'ANGL', 'AND_H', 'AND', 'XOR']
00697     metric_allowed_sc = [  20,     10,      5,      5,    10  ]
00698     alowed_metrics =    ['RMSD', 'ANGL', 'AND_H', 'AND', 'XOR']
00699     cur_metric = 0
00700     cur_metric_name = alowed_metrics[cur_metric]
00701     guiding_metric = 0  # main metric to tack global progress
00702 
00703     num_metrics = len(metric_names)
00704 
00705     an_file = 'ambient.noise'
00706     err_mult = 0.8
00707     tol_error = check_precomputed_noize(an_file, metric_names)
00708     if tol_error is None:
00709         goal_nz = os.path.join(prot_dir, 'folded_for_noise.gro')
00710         if hostnames:
00711             noize_file = gen_file_for_amb_noize(work_dir, seed_list, seed_dirs, ndx_file_goal,
00712                                                 topol_file_goal, goal_nz, hostnames, core_map)
00713         else:
00714             # noize_file = gen_file_for_amb_noize(work_dir, goal_nz, seed_list, seed_dirs, ndx_file_goal, topol_file_goal, goal_nz)
00715             noize_file = gen_file_for_amb_noize(work_dir, seed_list, seed_dirs, ndx_file_goal, topol_file_goal, goal_nz)
00716         # 0 - rmsd, 1 - angles, 2 - h_contacts, 3 - full_contacts_xor, 4 - full_contacts_and
00717     if tol_error is None or len(tol_error) < num_metrics:
00718         goal_prot_only_nz = os.path.join(prot_dir, 'goal_prot_nz.gro')
00719         gmx_trjconv(f=goal_nz, o=goal_prot_only_nz, n=ndx_file_goal, s=goal_nz)
00720         goal_angle_file_nz = os.path.join(prot_dir, 'goal_angle_nz.dat')
00721         goal_sincos_file_nz = os.path.join(prot_dir, 'goal_sincos_nz.dat')
00722         goal_bb_xtc_nz = os.path.join(prot_dir, 'goal_bb_nz.xtc')
00723         gmx_trjconv(f=goal_nz, o=goal_bb_xtc_nz, n=goal_bb_ndx, s=goal_nz)
00724         goal_xtc_nz = os.path.join(prot_dir, 'goal_nz.xtc')
00725         gmx_trjconv(f=goal_nz, o=goal_xtc_nz, n=ndx_file_goal)
00726         get_bb_to_angle_mdsctk(x=goal_bb_xtc_nz, o=goal_angle_file_nz)
00727         get_angle_to_sincos_mdsctk(i=goal_angle_file_nz, o=goal_sincos_file_nz)
00728         with open(goal_sincos_file_nz, 'rb') as file:
00729             initial_1d_array = np.frombuffer(file.read(), dtype=np.float64 , count=-1)
00730         goal_angles_nz = np.reshape(initial_1d_array, (-1, angl_num * 2))[0]
00731         del file, initial_1d_array
00732         goal_ind_nz = get_contat_profile_mdsctk(goal_prot_only, goal_xtc, ndx_file_goal, cont_dist)[1:]  # first is total num of contacts
00733         goal_contacts_nz = np.zeros(atom_num * atom_num, dtype=np.bool)
00734         goal_contacts_nz[goal_ind_nz] = True
00735         del goal_ind_nz
00736 
00737         h_pos_goal_nz = parse_top_for_h(topol_file_goal)
00738         h_filter_goal_nz = np.zeros(atom_num * atom_num, dtype=np.bool)
00739         for pos in h_pos_goal_nz:
00740             h_filter_goal_nz[(pos - 1) * atom_num:pos * atom_num] = True
00741         del h_pos_goal_nz, pos
00742         goal_cont_h_nz = np.logical_and(goal_contacts_nz, h_filter_goal_nz)
00743 
00744         goal_contacts_and_h_sum_nz = get_native_contacts(goal_prot_only_nz, [goal_xtc_nz], ndx_file_goal, goal_cont_h_nz,
00745                                                          atom_num, cont_dist, np.logical_and, pool=cpu_pool)[0]
00746         goal_contacts_and_sum_nz = np.sum(goal_contacts_nz)
00747         err_node_info = compute_init_metric(past_dir, tot_seeds, noize_file, goal_xtc_nz, goal_prot_only_nz, angl_num, goal_bb_ndx,
00748                                             goal_angles_nz, goal_prot_only_nz, ndx_file_goal, goal_cont_h_nz, atom_num, cont_dist,
00749                                             h_filter_goal_nz, goal_contacts_nz, goal_contacts_and_h_sum_nz, goal_contacts_and_sum_nz)
00750         tol_error = dict ()
00751         for metr_name in metric_names:
00752             tol_error[metr_name] = min([node['{}_to_goal'.format(metr_name)] for node in err_node_info]) * err_mult
00753         save_an_file(an_file, tol_error, metric_names)
00754         del err_node_info, metr_name
00755     del an_file
00756 
00757     print('Done measuring ambient noise for folded state at {}K.\n'
```

```
00758           'Min result for {} seeds was multiplied by {}.\n'
00759           'RMSD noise was {:0.5f}A\n'
00760           'PhiPsi angle noise was {:0.5f}\n'
00761           'Contact distance noise with AND logical function for H contacts was {:.3f}\n'
00762           'Contact distance noise with AND logical function was {:.3f}\n'
00763           'Contact distance noise with XOR logical function was {:.3f}\n'
00764           ".format(simulation_temp, tot_seeds, err_mult, tol_error['RMSD'], tol_error['ANGL'], tol_error['AND_H'],
00765                    tol_error['AND'], tol_error['XOR']))
00766       del err_mult
00767       node_info = compute_init_metric(past_dir, 1, init_xtc, goal_xtc, goal_prot_only, angl_num, init_bb_ndx, goal_angles, init_prot_only,
00768                                       ndx_file_init, goal_cont_h, atom_num, cont_dist, h_filter_init, goal_contacts,
00769                                       goal_contacts_and_h_sum, goal_contacts_and_sum)
00770
00771       print('Done measuring distance from initial state at {}K.\n'
00772             'RMSD dist: {:0.5f}A\n'
00773             'PhiPsi angle difference: {:0.5f}\n'
00774             'H contact disagreement (AND_H): {} of {}\n'
00775             'All contact disagreement (AND): {} of {}\n'
00776             'All contact disagreement (XOR): {}\n'.format(simulation_temp,
00777                                                           node_info['RMSD_to_goal'],
00778                                                           node_info['ANGL_to_goal'],
00779                                                           node_info['AND_H_to_goal'], goal_contacts_and_h_sum,
00780                                                           node_info['AND_to_goal'], goal_contacts_and_sum,
00781                                                           node_info['XOR_to_goal']))
00782       print('Unfolded to noise ratio:\n'
00783             'RMSD : {:.5f}\n'
00784             'PhiPsi angles: {:.5f}\n'
00785             'H contact (AND_H) disagreement: {:.5f}\n'
00786             'All contact (AND) disagreement: {:.5f}\n'
00787             'All contact disagreement (XOR): {:.5f}\n'.format(node_info['RMSD_to_goal'] / tol_error['RMSD'],
00788                                                               node_info['ANGL_to_goal'] / tol_error['ANGL'],
00789                                                               node_info['AND_H_to_goal']/tol_error['AND_H'],
00790                                                               node_info['AND_to_goal'] / tol_error['AND'],
00791                                                               node_info['XOR_to_goal'] / tol_error['XOR']))
00792
00793       # part of code used to study relation between contact distance and noise
00794       # f.write(
00795       #     '{} \n'.format(' '.join(str(elem) for elem in [cont_dist, node_info['AND_H_to_goal'], goal_contacts_and_h_sum,
00796       #     node_info['AND_H_to_goal'] / goal_contacts_and_h_sum, node_info['AND_to_goal'],
00797       #                                                    goal_contacts_and_sum,
00798       #                                                    node_info['AND_to_goal'] / goal_contacts_and_sum, node_info['XOR_to_goal'],
00799       #                                                    node_info['AND_H_to_goal'] / tol_error['AND_H'],
00800       #                                                    node_info['AND_to_goal'] / tol_error['AND'],
00801       #                                                    node_info['XOR_to_goal'] / tol_error['XOR']])))
00802       # print('done writing the file')
00803       # exit(22)
00804       # name_2_digest_map = dict ()
00805       # digest_2_name_map = dict ()
00806       # name_2_digest_map['s'] = get_digest('s')
00807       cur_hash_name = get_digest('s')
00808       # digest_2_name_map[name_2_digest_map['s']] = 's'
00809
00810       main_dict = dict ()
00811       main_dict[cur_hash_name] = node_info
00812
00813       open_queue = list()
00814       heapq.heappush(open_queue, (node_info['RMSD_to_goal'], 0, cur_hash_name))  # metric_val, attempts, name
00815
00816       cp2(init_xtc[:-4] + '.gro', os.path.join(past_dir, cur_hash_name + '.gro'))
00817       cp2(init_xtc[:-4] + '.xtc', os.path.join(past_dir, cur_hash_name + '.xtc'))
00818       # copy_queue.put_nowait((init_xtc[:-4] + '.gro', os.path.join(past_dir, name_2_digest_map['s'] + '.gro')))
00819       # copy_queue.put_nowait((init_xtc[:-4] + '.xtc', os.path.join(past_dir, name_2_digest_map['s'] + '.xtc')))
00820       # copy_queue.put_nowait(None)
00821
00822       visited_queue = list()
00823       skipped_counter = 0
00824
00825       combined_pg = os.path.join(work_dir, "out.xtc")
00826       temp_xtc_file = os.path.join(work_dir, "temp.xtc")
00827       # temp_xtc_file_bb = os.path.join(work_dir, "temp_bb.xtc")
00828
00829       loop_start = time.perf_counter()
00830
00831       # info_form_str = 'n:{}\db_input_thread:{:.4f}\tg:{:.4f}\ts:{}\tq:{}\tv:{}\tl:{:.2f}s\tc:{:.2f}s'
00832       info_form_str = 'o_q:{:<5}  v_q:{:<3}  s:{:<3}  grm:{:6.3f}  gan:{:6.3f}  gah:{:<4}  gad:{:<4}  gxo:{:<4}  ' \
00833                       't:{:5.2f}s  gbr:{:.4f}  gba:{:.4f}  gc:{:<2}  ns:{:3.1f}  sc:{}'
00834       #  node_info['rmds_total'], node_info['rmds_to_goal'], skipped_counter, len(open_queue), len(visited_queue),
00835       # loop_end - loop_start, best_so_far, global_best_so_far, greed_count, greed_mult, seed_change_counter,
00836       # node_info['nat_cont_to_goal']))
00837       # info_form_str.format(len(open_queue), len(visited_queue), skipped_counter, node_info['RMSD_to_goal'],
00838       # node_info['ANGL_to_goal'], node_info['AND_H_to_goal'],
```

```
00839     #                     node_info['AND_to_goal']), node_info['XOR_to_goal'], loop_end - loop_start, best_so_far[1],
00840     #                     best_so_far[0], greed_count, greed_mult, seed_change_counter)
00841     under_form_str = '{}_{}'
00842
00843     greed_mult = 1.0
00844     greed_count = 0
00845
00846     # con, dbname = get_db_con(tot_seeds)
00847     # insert_into_main_stor(con, node_info, greed_count, name_2_digest_map['s'], 's')
00848     db_input_queue.put_nowait((insert_into_main_stor, (node_info, greed_count, cur_hash_name, 's')))
00849
00850     node_max_att = 4
00851
00852     seed_change_counter = 0
00853     # change_metrics_limit = 3  # how many seed changes(20 iter per change) with no problems we have to have to change cur metricss
00854
00855     # search LMA in the code
00856     # seed_change_limit = 1000
00857     # local_minimum_counter = 0
00858     # local_minim_names = list()
00859
00860     # nmr_structure_switch = 2  # 0 for nmr, 1 for relaxed, 2 for heated
00861
00862     best_so_far = [node_info['{}_to_goal'.format(metr)] for metr in metric_names]
00863     print(best_so_far)
00864     best_so_far_name = [cur_hash_name] * num_metrics
00865     # global_best_so_far = best_so_far
00866
00867     Path(combined_pg).touch()
00868     Path(temp_xtc_file).touch()
00869     if os.path.exists('./local_min.xtc'):
00870         os.remove(('./local_min.xtc'))
00871
00872     compute_all_at_once = True
00873     counter_since_seed_changed = 0
00874
00875     recover = False  # STOP! before changing this toggle read bellow:
00876     # 1. Make backup of your pickles
00877     # 2. Remember number of the last good db - this name should always be the last one
00878     # There was no proper testing of this functionality and backups may overwrite last good state
00879     # Backups rely on time and number of steps, but if you have too fast/slow I/O - everything may go wrong. Thus do the pickle backup.
00880     if recover:  # this can (and should) be done in parallel or instead of most var initialization (much earlier)
00881         visited_queue, open_queue, main_dict = main_state_recover()
00882         prev_state = supp_state_recover()
00883         tol_error, seed_list, seed_dirs, seed_change_counter, skipped_counter, \
00884         cur_metric_name, cur_metric, counter_since_seed_changed, guiding_metric, greed_mult, \
00885         best_so_far_name, best_so_far, greed_count = prev_state
00886         del prev_state
00887         copy_old_db(list(main_dict.keys()), visited_queue[-3:].copy()[::-1], open_queue[0][2], greed_count-1)
00888
00889     # try:
00890     # aa = 0
00891     iter_from_bak = 0
00892     time_for_backup = False
00893     bak_time_check = time.perf_counter()
00894     while len(open_queue) > 0:  # and aa < 137:
00895         gc.collect()
00896         # if not aa % 10:
00897         #     # Prints out a summary of the large objects
00898         #     summary.print_(summary.summarize(muppy.get_objects()))
00899         # aa +=1
00900         new_elem = heapq.heappop(open_queue)  # tot_dist, att, name
00901         tot_dist, att, cur_hash_name = new_elem
00902         del new_elem
00903         if counter_since_seed_changed:  # you may disable this check, it was here to track nodes with the same name.
00904             if check_dupl(cur_hash_name, visited_queue[-counter_since_seed_changed:]):
00905                 continue
00906         # however, if you see nodes with the same name - check real name and if it is different - change hashing function
00907         # much
00908         counter_since_seed_changed += 1
00909
00910         node_info = main_dict[cur_hash_name]
00911         cur_name = zlib.decompress(node_info['native_name']).decode()
00912         # cur_file = os.path.join(past_dir, node_info['digest_name'])
00913
00914         visited_queue.append((tot_dist, att+1, cur_hash_name))  # TODO: trim it when size > 500 by 300, update tot_trim
00915         del tot_dist, att
00916
00917         db_input_queue.put_nowait((insert_into_visited, (cur_hash_name, greed_count)))
00918         db_input_queue.put_nowait((insert_into_log, ('result', cur_hash_name, 'WQ', 'VIZ', best_so_far, greed_count, greed_mult,
00919                                                      node_info['{}_dist_total'.format(cur_metric_name)],
```

```
00920                                               node_info['{}_to_goal'.format(cur_metric_name)], cur_metric_name)))
00921         # insert_into_visited(con, cur_name, greed_count)
00922         # insert_into_log(con, 'result', cur_name, 'WQ', 'VIZ', best_so_far, greed_count, greed_mult, node_info['{}_dist_total'.
00923         #                 format(cur_metric_name)], node_info['{}_to_goal'.format(cur_metric_name)])
00924         loop_end = time.perf_counter()
00925
00926         print_queue.put_nowait((info_form_str,
00927                                 ((len(open_queue), len(visited_queue), skipped_counter, node_info['RMSD_to_goal'],
00928                                   node_info['ANGL_to_goal'], node_info['AND_H_to_goal'], node_info['AND_to_goal'],
00929                                   node_info['XOR_to_goal'], loop_end - loop_start, best_so_far[0], best_so_far[1],
00930                                   greed_count, greed_mult, seed_change_counter))))
00931         # print(info_form_str.format(len(open_queue), len(visited_queue), skipped_counter, node_info['RMSD_to_goal'],
00932         #                           node_info['ANGL_to_goal'], node_info['AND_H_to_goal'], node_info['AND_to_goal'],
00933         #                           node_info['XOR_to_goal'], loop_end - loop_start, best_so_far[0], best_so_far[1],
00934         #                           greed_count, greed_mult, seed_change_counter))
00935
00936         # if node_info['ANGL_to_goal'] < best_so_far[1]:
00937         #     print('BSF:')
00938         #     print(best_so_far)
00939         #     print('Cur node info ANGL'.format(node_info['ANGL_to_goal']))
00940         #     print('Cur node info name'.format(cur_name))
00941         #     raise Exception('Error in best so far')
00942
00943         loop_start = time.perf_counter()
00944         if not use_mpi:
00945             pid_arr, files_for_trjcat, recent_filenames, recent_n2d, recent_d2n = compute_on_local_machine(cpu_map, seed_list, cur_name,
00946                                                                                                            past_dir, work_dir, seed_dirs,
00947                                                                                                            topol_file_init, ndx_file_init,
00948                                                                                                            prev_runs_files,
00949                                                                                                            cur_hash_name)
00950         else:
00951             pid_arr, files_for_trjcat, recent_filenames, recent_n2d, recent_d2n = compute_with_mpi(seed_list, cur_name, past_dir, work_dir,
00952                                                                                                    seed_dirs, topol_file_init,
00953                                                                                                    ndx_file_init, prev_runs_files,
00954                                                                                                    cur_hash_name, tot_seeds, hostnames,
00955                                                                                                    core_map, scheduler, nomp)
00956
00957         # update map
00958         # name_2_digest_map.update(recent_n2d)
00959         # digest_2_name_map.update(recent_d2n)
00960         # update prev files
00961         # prev_runs_files.extend(recent_filenames)
00962         if prev_runs_files:
00963             if len(prev_runs_files) <= tot_seeds*2:  # gro+xtc - two types
00964                 prev_runs_files = None
00965             else:
00966                 for file in recent_filenames:
00967                     try:
00968                         prev_runs_files.remove(file)
00969                     except Exception:
00970                         pass  # this is not an error - this behaviour is expected when you started following other route.
00971                         # print("Was not able to remove {}, list size: {}".format(file, len(prev_runs_files)))
00972                 del file
00973         del recent_filenames, recent_n2d, recent_d2n
00974
00975         os.remove(combined_pg)
00976         gmx_trjcat(f=['{}.xtc'.format(os.path.join(past_dir, cur_hash_name)), goal_xtc],
00977                    o=combined_pg, n=ndx_file_init, cat=True, vel=False, sort=False, overwrite=True)
00978
00979         [proc.join() for proc in pid_arr]
00980         del pid_arr
00981
00982         if compute_all_at_once or cur_metric < 2:
00983             os.remove(temp_xtc_file)
00984             gmx_trjcat(f=files_for_trjcat, o=temp_xtc_file, n=ndx_file_init, cat=True, vel=False, sort=False, overwrite=True)
00985
00986         new_nodes_names = [under_form_str.format(cur_name, seed_name) for seed_name in seed_list]
00987         # for i, node in enumerate(new_nodes):
00988         #     new_nodes[i]['digest_name'] = get_digest(new_nodes_names[i])
00989         #     # new_nodes[i]['native_name'] = new_nodes_names[i]
00990         #     new_nodes[i]['native_name'] = zlib.compress(new_nodes_names[i].encode(), 9)
00991         # del node, i
00992         new_nodes, metric_to_goal, metric_form_prev, metric_to_tot = compute_metric(past_dir, new_nodes_names, tot_seeds, combined_pg,
00993                                                                                     temp_xtc_file, goal_prot_only, node_info, angl_num,
00994                                                                                     init_bb_ndx, goal_angles, init_prot_only,
00995                                                                                     files_for_trjcat, ndx_file_init, goal_cont_h,
00996                                                                                     atom_num, cont_dist, h_filter_init, goal_contacts,
00997                                                                                     cur_metric, goal_contacts_and_h_sum,
00998                                                                                     goal_contacts_and_sum, prev_runs_files is not None,
00999                                                                                     cpu_pool=cpu_pool,
01000                                                                                     compute_all_at_once=compute_all_at_once)
```

```
01001         del files_for_trjcat
01002
01003         new_filtered = list()
01004         for i in range(tot_seeds):
01005             # if seed_change_counter:
01006             #     local_minim_names.append(seed_name)
01007
01008             # MAIN INSERT  new_nodes, metric_form_prev, metric_to_goal, metric_to_tot
01009             # we have two conditions to get intro the queue:
01010             # 1st - get better than the best result (obvious)
01011             # 2nd - we have to make big enough step from the previous point
01012             # AND this step should bring us closer to the goal 1/2 of just a noise
01013             if (metric_form_prev[i] > tol_error[cur_metric_name]
01014                 and metric_to_goal[i] - node_info['{}_to_goal'.format(cur_metric_name)] < tol_error[cur_metric_name] / 2) \
01015                     or metric_to_goal[i] <= best_so_far[cur_metric]:
01016                 # LMA - this approach is currently frozen since it did not show any benefits with RMSD,
01017                 # but was never adapted to multiple metrics
01018                 # if check_local_minimum(temp_xtc_file, goal_prot_only, tol_error):
01019                 # else:
01020                 #     print('point was on path to local minimum')
01021
01022                 heapq.heappush(open_queue, (greed_mult * metric_to_tot[i] + metric_to_goal[i], 0, new_nodes[i]['digest_name']))
01023                 new_filtered.append((greed_mult * metric_to_tot[i] + metric_to_goal[i], 0, new_nodes[i]['digest_name']))
01024                 # insert_into_main_stor(con, new_nodes[i], greed_count,
01025                 # name_2_digest_map[new_nodes_names[i]], new_nodes_names[i])
01026                 db_input_queue.put_nowait((insert_into_main_stor,
01027                                            (new_nodes[i], greed_count, new_nodes[i]['digest_name'], new_nodes_names[i])))
01028                 main_dict[new_nodes[i]['digest_name']] = new_nodes[i]
01029             else:
01030                 skipped_counter += 1
01031                 # insert_into_log(con, 'skip', cur_name, '', 'SKIP', best_so_far, greed_count,
01032                 # greed_mult, metric_form_prev[i], metric_form_prev[i])
01033                 db_input_queue.put_nowait((insert_into_log, ('skip', cur_hash_name, '', 'SKIP', best_so_far, greed_count,
01034                                                              greed_mult, metric_form_prev[i], metric_to_goal[i], cur_metric_name)))
01035         db_input_queue.put_nowait((insert_into_log, ('current', cur_hash_name, '', 'WQ', best_so_far, greed_count,
01036                                                      greed_mult, metric_form_prev, metric_to_goal, cur_metric_name)))
01037         del metric_to_tot, metric_form_prev, i, new_nodes_names
01038
01039         if compute_all_at_once:
01040             for metr in metric_names:
01041                 if metr != cur_metric_name:
01042                     min_val = min([node['{}_to_goal'.format(metr)] for node in new_nodes])
01043                     if best_so_far[metric_names.index(metr)] > min_val:
01044                         # print('bsf["{}"]={:.4f}, min={:.4f}'.
01045                         # format(metr, best_so_far[metric_names.index(metr)], min_val), end=' ')
01046                         best_so_far[metric_names.index(metr)] = min_val
01047                     del min_val
01048                 # else:
01049                     # print('skipping "{}"'.format(metr), end=' ')
01050             del metr
01051         # print()
01052         if best_so_far[guiding_metric] >
      new_nodes[metric_to_goal.index(min(metric_to_goal))]['{}_to_goal'.format(metric_names[guiding_metric])]:
01053             seed_change_counter = 0
01054
01055         if best_so_far[cur_metric] > min(metric_to_goal):
01056             best_so_far_new = min(metric_to_goal)
01057             best_so_far[cur_metric] = best_so_far_new
01058             best_so_far_name[cur_metric] = new_nodes[metric_to_goal.index(best_so_far_new)]['digest_name']
01059             db_input_queue.put_nowait((insert_into_log,
01060                                        ('prom_0', best_so_far_name[cur_metric], '', best_so_far, greed_count, greed_mult,
01061                                         new_nodes[metric_to_goal.index(best_so_far_new)]['{}_from_prev'.format(cur_metric_name)],
01062                                         new_nodes[metric_to_goal.index(best_so_far_new)]['{}_to_goal'.format(cur_metric_name)],
01063                                         cur_metric_name)))
01064             if guiding_metric == cur_metric or best_so_far[guiding_metric] >=
      new_nodes[metric_to_goal.index(best_so_far_new)]['{}_to_goal'.format(metric_names[guiding_metric])]:
01065                 for i in range(num_metrics):
01066                     if i != cur_metric:
01067                         best_so_far_name[i] = best_so_far_name[cur_metric]
01068                         best_so_far[i] = new_nodes[metric_to_goal.index(best_so_far_new)]['{}_to_goal'.format(metric_names[i])]
01069                 del i
01070             seed_change_counter = 0
01071
01072             # local_minim_names = list()  # search for LMA
01073             # if global_best_so_far[cur_metric] > best_so_far_new:
01074             #     global_best_so_far[cur_metric] = best_so_far_new
01075
01076             # This code is for multiple stage folding. Code has to be adapted for several metrics.
01077             # if len(visited_queue) > 1 and global_best_so_far < visited_queue[1][2]/5 and nmr_structure_switch == 1:
01078             #     print('Changing goal to nmr structure')
01079             #     cp2(os.path.join(prot_dir, 'nmr.gro'), goal)
```

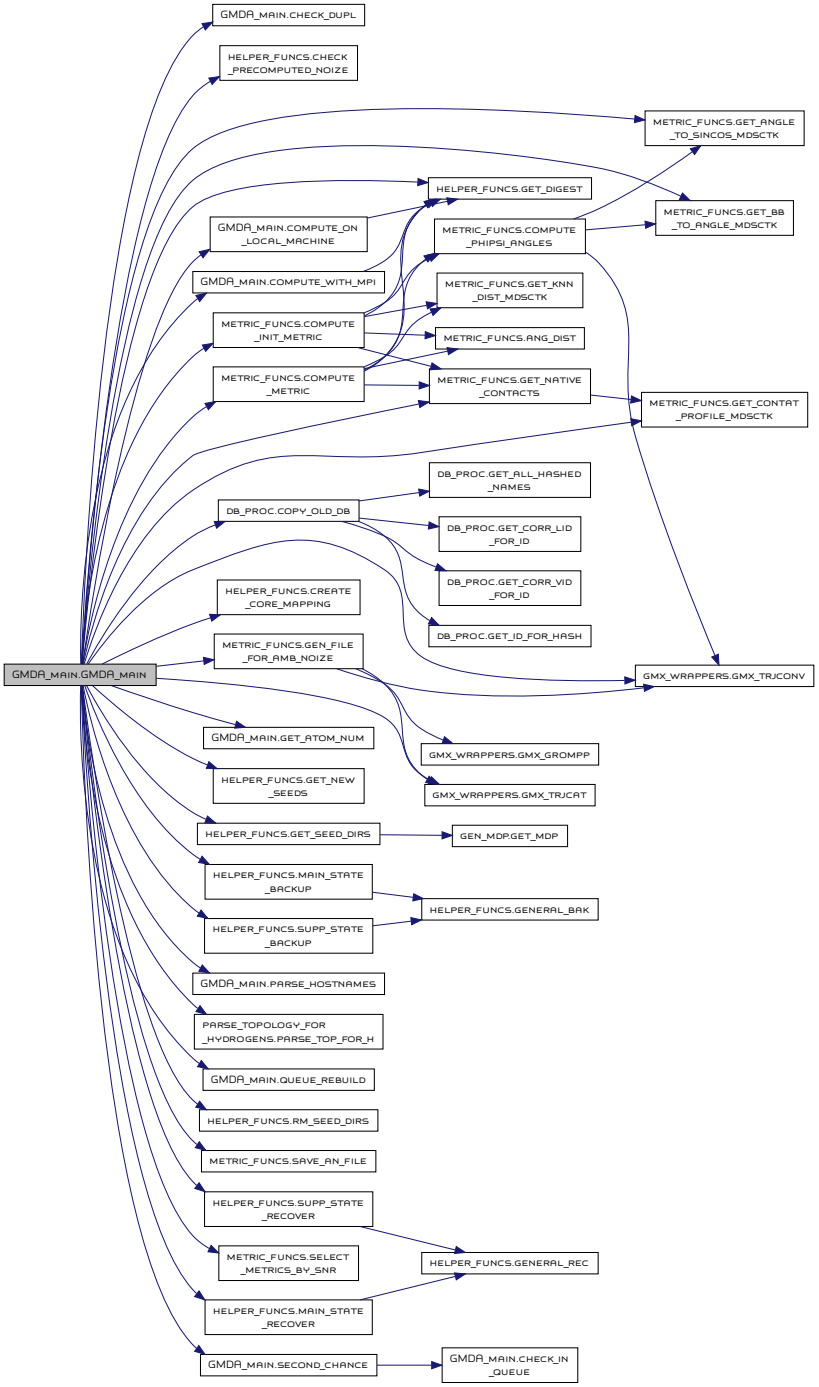
```
01080             #     gmx_trjconv(f=goal, o=goal_xtc, n=ndx_file)
01081             #     gmx_trjconv(f=goal, o=goal_prot_only, n=ndx_file, s=goal)
01082             #     open_queue = recompute_rmsd_for_openq(open_queue, goal_xtc, name_2_digest_map, past_dir,
01083             #     goal_prot_only, greed_mult)
01084             #     best_so_far = open_queue[-1][2]
01085             #     nmr_structure_switch = 0
01086             # elif len(visited_queue) > 1 and global_best_so_far < visited_queue[1][2]/3 and nmr_structure_switch == 2:
01087             #     print('Changing goal to relaxed structure')
01088             #     cp2(os.path.join(prot_dir, 'relaxed.gro'), goal)
01089             #     gmx_trjconv(f=goal, o=goal_xtc, n=ndx_file)
01090             #     gmx_trjconv(f=goal, o=goal_prot_only, n=ndx_file, s=goal)
01091             #     open_queue = recompute_rmsd_for_openq(open_queue, goal_xtc, name_2_digest_map, past_dir,
01092             #     goal_prot_only, greed_mult)
01093             #     best_so_far = open_queue[-1][2]
01094             #     nmr_structure_switch = 1
01095 
01096             # This is part of local minimum approach (LMA) search for LMA in this code
01097             # if os.path.exists('./local_minim_bas.xtc'):
01098             #     os.remove('./local_minim_bas.xtc')
01099             del best_so_far_new
01100             if greed_mult < 1.0:  # perfect place to optimize queue rebuild
01101                 greed_count = max(0, 10 * (greed_count // 10) - 8)
01102                 if 100 < greed_count < 110:
01103                     greed_count = 101
01104                 else:
01105                     greed_mult = min(1.001 - min(1.0, (greed_count // 10) / 10), 1.0)
01106                     open_queue = queue_rebuild(None, open_queue, main_dict, greed_mult, cur_metric_name, sep_proc=False)
01107             else:
01108                 greed_count = 0
01109         else:
01110             greed_count += 1
01111 
01112             if greed_count in range(10, 101, 10):
01113                 # open_queue = rebuild_queue.get(timeout=1800)[0]  # 30min
01114                 open_queue = rebuild_queue.get()[0]  # 30min
01115                 if new_filtered:
01116                     for elem in new_filtered:
01117                         heapq.heappush(open_queue, elem)
01118                 # cur_metric = metric_names.index(cur_metric_name)
01119                 del rebuild_queue
01120                 # if not isinstance(rebuild_queue_process, mp.Process):
01121                 #     a=8
01122                 rebuild_queue_process.join()
01123 
01124             elif greed_count == 121:
01125                 seeds_next = get_new_seeds(seed_list)
01126                 seed_change_counter += 1
01127                 seed_dirs_next = get_seed_dirs(work_dir, seeds_next, simulation_temp)
01128                 # previously I passed here "seed_dirs", but decided to save RAM
01129                 if seed_change_counter > metric_allowed_sc[cur_metric]:
01130                     new_metr_name = select_metrics_by_snr(new_nodes, node_info, metric_names, tol_error,
01131                                                           compute_all_at_once, alowed_metrics, cur_metric_name)
01132                     rebuild_queue = mp.Queue()
01133                     # open_queue = queue_rebuild(None, open_queue, main_dict, greed_mult, new_metr_name, sep_proc=False)
01134                     rebuild_queue_process = mp.Process(target=queue_rebuild,
01135                                                        args=(rebuild_queue, open_queue, main_dict, greed_mult, new_metr_name))
01136                     # if not isinstance(rebuild_queue_process, mp.Process):
01137                     #     a = 8
01138                     rebuild_queue_process.start()
01139                     del new_metr_name
01140                 # TODO: local minimum has to be rethought and rewritten.
01141                 #  At this point (before multiple metrics) experiments show that is does not give any benefits
01142                 # if seed_change_counter == seed_change_limit:
01143                 #     seed_change_counter = 0
01144                 #     greed_count = 112
01145                 #     open_queue = proc_local_minim(open_queue, best_so_far_name[cur_metric], tol_error, ndx_file_init,
01146                 #     name_2_digest_map, goal_prot_only, local_minim_names)
01147                 #     local_minim_names = list()
01148                 #     best_so_far[cur_metric] = (init_distance[cur_metric] + best_so_far[cur_metric])/2
01149                 #     local_minimum_counter += 1
01150                 #     continue
01151         del metric_to_goal
01152 
01153         if greed_count in range(9, 100, 10):
01154             rebuild_queue = mp.Queue()
01155             greed_mult = min(1.001 - (greed_count+1) / 100, 1.0)
01156             rebuild_queue_process = mp.Process(target=queue_rebuild, args=(rebuild_queue, open_queue, main_dict,
01157                                                                            greed_mult, cur_metric_name))
01158             rebuild_queue_process.start()
01159         elif greed_count == 122:
01160             greed_count = 102
```

```
01161             if seed_change_counter > metric_allowed_sc[cur_metric]:
01162                 print('Switching metric from {} to '.format(cur_metric_name), end="")
01163                 open_queue, cur_metric_name = rebuild_queue.get()  # 30min
01164                 # open_queue, cur_metric_name = rebuild_queue.get(timeout=1800)  # 30min
01165                 print(cur_metric_name)
01166                 cur_metric = metric_names.index(cur_metric_name)
01167                 del rebuild_queue
01168                 rebuild_queue_process.join()
01169                 extra_elem_q = queue_rebuild(None, new_filtered, main_dict, greed_mult, cur_metric_name, sep_proc=False)
01170                 for elem in extra_elem_q:
01171                     heapq.heappush(open_queue, elem)
01172                 del extra_elem_q, elem
01173                 seed_change_counter = 0
01174                 # greed_count = 102
01175 
01176             if seeds_next:
01177                 seed_list = seeds_next
01178                 rm_seed_dirs(seed_dirs)
01179                 seed_dirs = seed_dirs_next
01180                 res_arr = second_chance(open_queue[0:min(len(open_queue)-1, max(40, 4*counter_since_seed_changed))],
01181                                         visited_queue[min(-1, -counter_since_seed_changed):],
01182                                         best_so_far_name, cur_metric, main_dict, node_max_att,
01183                                         cur_metric_name, best_so_far, tol_error, greed_mult)
01184                 counter_since_seed_changed = 0
01185                 for elem in res_arr:
01186                     heapq.heappush(open_queue, elem)
01187                     # print(elem)
01188                     db_input_queue.put_nowait((insert_into_log,
01189                                                ('result', cur_hash_name, 'VIZ', 'WQ', best_so_far, greed_count, greed_mult,
01190                                                 main_dict[elem[2]]['{}_from_prev'.format(cur_metric_name)],
01191                                                 main_dict[elem[2]]['{}_to_goal'.format(cur_metric_name)], cur_metric_name)))
01192             else:
01193                 print('\nOUT OF SEEDS\n')
01194                 greed_count = 102  # will be changed soon
01195             del seeds_next, seed_dirs_next
01196         del cur_hash_name, cur_name, new_nodes, node_info
01197         new_filtered.clear()
01198         iter_from_bak += 1
01199         if loop_start - bak_time_check > 60*60 and not time_for_backup:  # every hour
01200             if iter_from_bak < 1000:  # expected value 240 - means that we are computing (on 32 cores), but not reading from ./past, typical
       read speed 10 000 iterations/hour (for non SSD)
01201                 time_for_backup = True
01202             else:
01203                 iter_from_bak = 0
01204                 bak_time_check = loop_start
01205 
01206         if time_for_backup and (greed_count in range(104, 109) or greed_count in range(113, 117) or greed_count in range(93, 97)):
01207             main_state_backup((visited_queue, open_queue, main_dict))
01208             supp_state_backup((tol_error, seed_list, seed_dirs, seed_change_counter, skipped_counter, cur_metric_name,
01209                                cur_metric, counter_since_seed_changed, guiding_metric, greed_mult,
01210                                best_so_far_name, best_so_far, greed_count))
01211             time_for_backup = False
01212             bak_time_check = time.perf_counter()
01213             iter_from_bak = 0
01214 
01215 
01216     # except (KeyboardInterrupt, Exception) as e:
01217     #     print('Got exception: ', e)
01218     #     exc_type, exc_obj, exc_tb = sys.exc_info()
01219     #     fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
01220     #     print(exc_type, fname, exc_tb.tb_lineno)
01221     #     # print('Dumping work_queue')
01222     #     # dump_the_queue('work_queue.txt', open_queue, visited_queue, init_rmsd, tol_error, skipped_counter)
01223     #     # print('Dumping visited_queue')
01224     #     # dump_the_queue('visited_queue.txt', visited_queue, visited_queue, init_rmsd, tol_error, skipped_counter)
01225     #     # print('Done dumping ')
01226     #     # exit(-1)
01227     #
01228     #     # if keyboard.is_pressed('md_process'):
01229     #     #     print('Dumping ')
01230     #     #     dump_the_queue('work_queue.txt', open_queue, visited_queue, init_rmsd, tol_error, skipped_counter)
01231     #     #     print('Dumping ')
01232     #     #     dump_the_queue('visited_queue.txt', visited_queue, visited_queue, init_rmsd, tol_error, skipped_counter)
01233     #     #     print('Done dumping ')
01234     #
01235     #     # ne = open_queue[0]
01236     #     # trav = ne[1]
01237     #     # to_goal = ne[2]
01238     #     # sds = ne[3]
01239     #     # tot_points = len(sds.split("_")) - 1
01240     #     # from_prev_dist, prev_goal_dist = current_job[1], current_job[2]
```

```
01241     #     # trav_from_prev = trav - from_prev_dist
01242     #     # coef_1 = 1 - to_goal / init_rmsd
01243     #     # coef_1_a = coef_1 / tot_points if tot_points != 0 else 9999
01244     #     # deriv = (prev_goal_dist - to_goal) / trav_from_prev  # this cannot be zero
01245     #     # full_line = '{:.5f} {:.5f} {:.5f} {:.5f} {:.5f} {:.5f} {}\n'.format(trav,
01246     #     #                                                                     to_goal,
01247     #     #                                                                     trav_from_prev,
01248     #     #                                                                     coef_1,
01249     #     #                                                                     coef_1_a,
01250     #     #                                                                     deriv,
01251     #     #                                                                     sds)
01252     #     # file.write(full_line)
01253     #
01254     #     # check_end = time.perf_counter()
01255     #
01256     # print('We are finally done with search.')
01257     # print('Current queue size: ', len(open_queue))
01258     # print('Current visited_queue queue: ', len(visited_queue))
01259     # # dump_the_queue('work_queue.txt', open_queue, visited_queue, init_rmsd, tol_error, skipped_counter)
01260     # # dump_the_queue('visited_queue.txt', visited_queue, visited_queue, init_rmsd, tol_error, skipped_counter)
```

References check_dupl(), helper_funcs.check_precomputed_noize(), metric_funcs.compute_init_metric(), metric_funcs.compute_metric(), compute_on_local_machine(), compute_with_mpi(), db_proc.copy_old_db(), helper_funcs.create_core_mapping(), metric_funcs.gen_file_for_amb_noize(), metric_funcs.get_angle_to_sincos_mdsctk(), get_atom_num(), metric_funcs.get_bb_to_angle_mdsctk(), metric_funcs.get_contat_profile_mdsctk(), helper_funcs.get_digest(), metric_funcs.get_native_contacts(), helper_funcs.get_new_seeds(), helper_funcs.get_seed_dirs(), gmx_wrappers.gmx_trjcat(), gmx_wrappers.gmx_trjconv(), helper_funcs.main_state_backup(), helper_funcs.main_state_recover(), parse_hostnames(), parse_topology_for_hydrogens.parse_top_for_h(), queue_rebuild(), helper_funcs.rm_seed_dirs(), metric_funcs.save_an_file(), second_chance(), metric_funcs.select_metrics_by_snr(), helper_funcs.supp_state_backup(), and helper_funcs.supp_state_recover().

Here is the call graph for this function:



3.11.1.7 parse_hostnames()

```
tuple GMDA_main.parse_hostnames (
        int  seednum,
        str  hostfile =  'hostfile' )
```

Spreads the load among the hosts found in the hostfile.
Needed for MPI

seednum: total number of seeds used in the current run
hostfile: filename of the hostfile

Returns

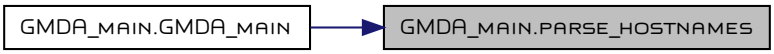
:return: hosts split partitioned according to the number of seeds and total number of cores for each job

Definition at line 228 of file GMDA_main.py.

```
00228     """
00229     with open(hostfile, 'r') as f:
00230         hosts = f.readlines()
00231     del hostfile
00232     hostnames = [elem.strip().split(' ')[0] for elem in hosts]
00233     ncores = [int(elem.strip().split(' ')[1].split('=')[1]) for elem in hosts]
00234     ev_num = len(hosts) // seednum
00235     if ev_num == 0:
00236         raise Exception('Special case is not implemented')
00237     else:
00238         chopped = [tuple (hostnames[i:i+ev_num]) for i in range(0, len(hostnames), ev_num)]
00239         ncores_sum = [sum(ncores[i:i+ev_num]) for i in range(0, len(ncores), ev_num)]
00240     return chopped, ncores_sum
00241
00242
```

Referenced by GMDA_main().

Here is the caller graph for this function:



3.11.1.8 queue_rebuild()

```
list GMDA_main.queue_rebuild (
                list  process_queue,
                list  open_queue_to_rebuild,
                dict  node_info,
                float  cur_mult,
                str  new_metr_name,
                bool  sep_proc =  True )
```

Resorts the queue according to the new metric.

list process_queue: queue to use if function is executed in a separate process
list open_queue_to_rebuild: sorted queue that contains nodes about to be processed. This is actually only a partial queue (only top elements)
dict node_info:
float cur_mult: current greedy factor
str new_metr_name: defines how to sort the new queue
bool sep_proc: whether the function runs in a separate process

Returns

:return: if separate process - then new queue and metric name are pushed into the queue, otherwise returned :rtype: list

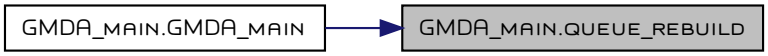
Definition at line 180 of file GMDA_main.py.

```
00180     """
00181     gc.collect()
00182     new_queue = list()
00183     to_goal, total = '{}_to_goal'.format(new_metr_name), '{}_dist_total'.format(new_metr_name)
00184     try:
00185         for elem in open_queue_to_rebuild[1:]:
00186             heapq.heappush(new_queue, (cur_mult*node_info[elem[2]][total] + node_info[elem[2]][to_goal], 0, elem[2]))
00187     except Exception:
00188         print(len(node_info))
00189         print(len(open_queue_to_rebuild))
00190         print(new_metr_name)
00191         print(cur_mult)
00192         print(sep_proc)
00193     del open_queue_to_rebuild
00194     gc.collect()
00195     if sep_proc:
```

```
00196         process_queue.put((new_queue, new_metr_name))
00197     else:
00198         return new_queue
00199 
00200 
```

Referenced by GMDA_main().

Here is the caller graph for this function:



3.11.1.9 second_chance()

```
list GMDA_main.second_chance (
                list  open_queue,
                list  visited_queue,
                str  best_so_far_name,
                str  cur_metric,
                dict  main_dict,
                int  node_max_att,
                str  cur_metric_name,
                str  best_so_far,
                float  tol_error,
                float  greed_mult )
```

Typically executed during the seed change.

We want to give the second chance to a promising trajectories with different seeds. Typically, we allow up to 4 attempts. However, the best trajectories are always readded to the queue.

list open_queue: sorted queue that contains nodes about to be processed. This is actually only a partial queue (only top elements)
list visited_queue: sorted queue that contains nodes processed prior. This is actually only a partial queue (only top elements)
str best_so_far_name: node with the closest distance to the goal according to
the guiding metric - we want to keep it for a long time, with hope that it will jump over the energy barrier
str cur_metric: index of the current metric
dict main_dict: map with all the information (prior and goal distances for all metrics, names, hashnames, attempts, etc)
int node_max_att: defines how many attempts each node can have
str cur_metric_name: name of the current metric
str best_so_far: name of the node with the closest metric distance to the goal
float tol_error: minimal metric vibration of the NMR structure
float greed_mult: greedy multiplier, used to assign correct metric value (ballance between optimality and greedyness)

Returns

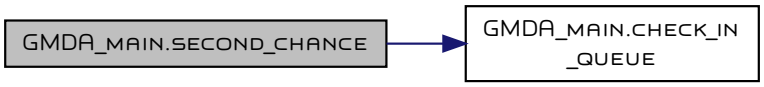
:return: short list of promising nodes, they will be merged with the open queue later :rtype: list

Definition at line 462 of file GMDA_main.py.

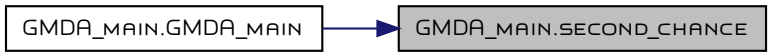
```
00462         return type: list
00463     """
00464 
00465     res_arr = list()
00466     recover_best = True
00467     for elem in open_queue:
00468         if elem[2] == best_so_far_name[cur_metric]:
00469             recover_best = False
00470             break
00471 
00472     for elem in visited_queue:  # elem structure: tot_dist, att, cur_name
00473         # we give node_max_att attempts for a node to make progress with different seed
00474         if (elem[1] < node_max_att and main_dict[elem[2]]['{}_to_goal'.format(cur_metric_name)] - best_so_far[cur_metric] <
       tol_error[cur_metric_name]):  # \
00475             # and elem[2] != best_so_far_name[cur_metric]:
00476             # or main_dict[elem[2]]['{}_to_goal'.format(cur_metric_name)] != best_so_far[cur_metric]:
00477             if elem[2] == best_so_far_name[cur_metric]:
00478                 if recover_best:
00479                     res_arr.append(elem)
00480                     recover_best = False
00481                     break
00482             else:
```

```
00483                 if elem[1] > 1 and check_in_queue(open_queue, elem[2]):
00484                     print('Not adding regular node (already in the queue)')
00485                 else:
00486                     res_arr.append(elem)
00487                     print('Readding "{}" with attempt counter: {} and dist: {}'.format(elem[2], elem[1], elem[0]))
00488 
00489     elem = main_dict[best_so_far_name[cur_metric]]
00490     if recover_best:
00491         res_arr.append((elem['{}_dist_total'.format(cur_metric_name)] * greed_mult + elem['{}_to_goal'.format(cur_metric_name)],
00492                         0, best_so_far_name[cur_metric]))
00493         print('Recovering best')
00494     else:
00495         print('Not recovering best (already in the open queue)')
00496     del elem
00497 
00498     return res_arr
00499 
00500 
```

References check_in_queue().
Referenced by GMDA_main().
Here is the call graph for this function:



Here is the caller graph for this function:



3.11.2 Variable Documentation

3.11.2.1 MAX_ITEMS_TO_HANDLE

int GMDA_main.MAX_ITEMS_TO_HANDLE = 50000

Definition at line 37 of file GMDA_main.py.

3.12 gmx_wrappers Namespace Reference

Functions

- str convert_gro_to_xtc (str gro_file, str ndx_file)

  Converts .gro into .xtc format.
- NoReturn gmx_trjconv (str f, str o, str n=None, str s=None, int b=None, int e=None, int dump=None, str fit=None, str vel=None, str pbc=None)
- NoReturn gmx_trjcat (str f, str o, str n, bool cat=True, bool vel=False, bool sort=False, bool overwrite=True)

  'gmx trjcat' - GROMACS tool - concatenates several input trajectory files in sorted order
- NoReturn gmx_eneconv (str f, str o)

  'gmx eneconv' - GROMACS tool - Concatenates several energy files in sorted order
- NoReturn gmx_energy (str f, str o, bool w=None, str w_prog=None, bool fee=True, float fetemp=300)

  'gmx trjconv' - GROMACS tool - extracts energy components from an energy file
- NoReturn gmx_mdrun (str work_dir, int seed, str new_name, int ncores=multiprocessing.cpu_count(), str thread_type='nt')

  gmx localhost version.

- NoReturn gmx_mdrun_mpi (str work_dir, int seed, str new_name, list hostnames, list ncores=None, str thread_type='ntomp')

  gmx MPI version

- NoReturn gmx_mdrun_mpi_with_sched (str work_dir, int seed, str new_name, list ncores=None, int ntomp=1)

  gmx MPI version with scheduler

- NoReturn gmx_grompp (str work_dir, int seed, str top_file, str prev_name)

  gmx grompp (the gromacs preprocessor) reads a molecular topology file, checks the validity of the file, expands the topology from a molecular description to an atomic description.

Variables

- my_env = os.environ.copy()

3.12.1 Function Documentation

3.12.1.1 convert_gro_to_xtc()

```
str gmx_wrappers.convert_gro_to_xtc (
        str gro_file,
        str ndx_file )
```

Converts .gro into .xtc format.
Just a wrapper around trjconv.

str gro_file: input filename
str ndx_file: index file, shows which atoms to store in .xtc

Returns

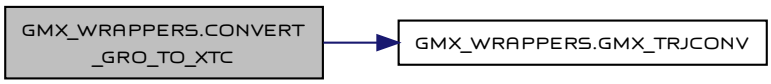
:return: .xtc filename

Definition at line 30 of file gmx_wrappers.py.

```
00030     """
00031     out_filename = gro_file[0:-3] + 'xtc'
00032     gmx_trjconv(f=gro_file, o=out_filename, n=ndx_file)
00033     return out_filename
00034
00035
```

References gmx_trjconv().

Here is the call graph for this function:



3.12.1.2 gmx_eneconv()

```
NoReturn gmx_wrappers.gmx_eneconv (
        str f,
        str o )
```

'gmx eneconv' - GROMACS tool - Concatenates several energy files in sorted order
Stores converted energy files. Not used by main algorithm, but during the postprocessing.

str f: Input trajectory: xtc trr cpt gro g96 pdb tng
str o: Output trajectory: xtc trr gro g96 pdb tng

Returns

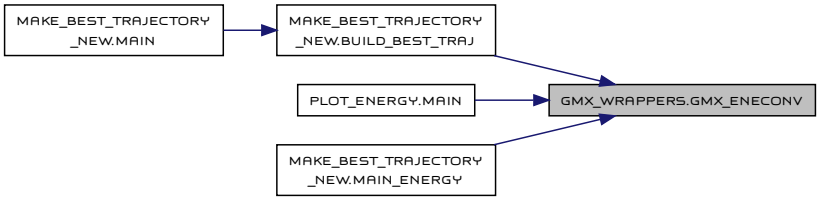
Generates one output energy file passed with -o parameter.

Definition at line 164 of file gmx_wrappers.py.

```
00164         command_eneconv += '-f ' + ' '.join(f) + ' -nosort -settime '
00165         #command_eneconv += '-f ' + ' '.join(f) + ' -settime '
00166         # command_eneconv = 'echo -e "{}" | '.format('\n'.join([str(i) for i in range(0, len(f) * 20, 20)])) + command_eneconv
00167         command_eneconv = 'echo -e "{}" | '.format('\n'.join(['c']*(len(f)+1))) + command_eneconv
00168     else:
00169         command_eneconv += '-f {:s} '.format(f)
00170 
00171     command_eneconv = os.path.expandvars(command_eneconv)
00172     proc_obj = subprocess.Popen(command_eneconv, stdout=-1, shell=True, cwd='.', stderr=-1, env=my_env)
00173     output, error = proc_obj.communicate()
00174     error = error.decode("utf-8")
00175     if 'error' in error.lower():
00176         print(error)
00177 
00178 
00179 def gmx_energy(f: str, o: str, w: bool = None, w_prog: str  = None, fee: bool = True, fetemp: float = 300) -> NoReturn:
00180     """'gmx trjconv' - GROMACS tool - extracts energy components from an energy file
00181 
00182     Args:
00183          str  f: .edr Energy file
00184          str  o: energy.xvg - xvgr/xmgr file
```

Referenced by make_best_trajectory_new.build_best_traj(), plot_energy.main(), and make_best_trajectory_new.main_energy().

Here is the caller graph for this function:



3.12.1.3 gmx_energy()

```
NoReturn  gmx_wrappers.gmx_energy (
                str  f,
                str  o,
                bool  w =  None,
                str   w_prog =  None,
                bool  fee =  True,
                float  fetemp =  300 )
```

'gmx trjconv' - GROMACS tool - extracts energy components from an energy file

```
    str  f: .edr Energy file
    str  o: energy.xvg - xvgr/xmgr file
    str  w: View output .xvg, .xpm, .eps and .pdb files
    str  w_prog: viewing programm
    bool   fee: Do a free energy estimate
    float   fetemp: Reference temperature for free energy calculation
```

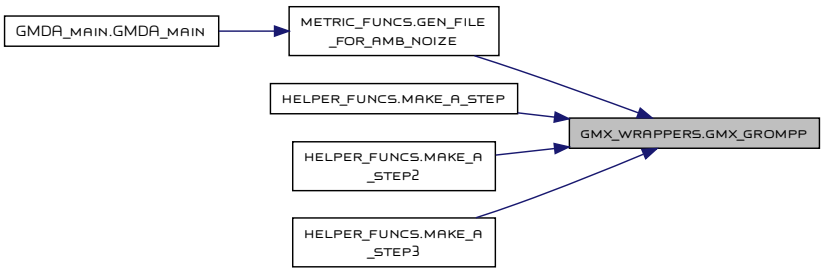
Returns

Generates one output .xvg file passed with -o parameter.

Definition at line 198 of file gmx_wrappers.py.

```
00198     if fee:
00199         command_energy += ' -fee '
00200     if fetemp:
00201         command_energy += ' -fetemp {}'.format(fetemp)
00202     command_energy = 'echo -e "10" | ' + command_energy
00203     command_energy = os.path.expandvars(command_energy)
00204     proc_obj = subprocess.Popen(command_energy, stdout=-1, shell=True, cwd='.', stderr=-1, env=my_env)
00205     output, error = proc_obj.communicate()
00206     error = error.decode("utf-8")
00207     if 'error' in error.lower():
```

```
00208         print(error)
00209
00210
00211 def gmx_mdrun(work_dir: str, seed: int, new_name: str, ncores: int = multiprocessing.cpu_count(), thread_type: str  = 'nt') -> NoReturn:
00212     """gmx mdrun - localhost version.
00213
00214     Args:
00215         str  work_dir: path to work directory, where all seed directories reside
00216         int seed: seed value used in the MD simulation
gmx grompp (the gromacs preprocessor) reads a molecular topology file, checks the validity of the file, expands the topology from a molecular
 description to an atomic description.

    Args\+:\+:    str  work_dir: path to work directory, where all seed directories reside
    int  seed: seed value used in the MD simulation
    str  top_file: .top - topology of the conformation
    str  prev_name: previous simulation digest. Used as starting point.

Returns
Creates binary config file.
Definition at line 341 of file gmx_wrappers.py.
00341     #     log_out.write(error)
00342     # with open(str(os.getpid())+'_out.log', 'a') as log_out:
00343     #     log_out.write(output.decode("utf-8"))
00344
00345     if 'error' in error.lower():
00346         print(error)
Referenced by metric_funcs.gen_file_for_amb_noize(), helper_funcs.make_a_step(), helper_funcs.make_a_step2(), and helper_funcs.make_a_step3().
Here is the caller graph for this function:
```



3.12.1.4 gmx_mdrun()

```
NoReturn  gmx_wrappers.gmx_mdrun (
                str  work_dir,
                int  seed,
                str  new_name,
                int  ncores =  multiprocessing.cpu_count(),
                str   thread_type =  'nt' )
```

gmx localhost version.

    str work_dir: path to work directory, where all seed directories reside
    int seed: seed value used in the MD simulation
    str new_name: output name for a final state
    int ncores: number of cores to use in the current simulation
    str thread_type: thread type: MPI ? OMP ? TMPI ?

Returns

Starts a shell in a separate process and runs mdrun there.

Definition at line 229 of file gmx_wrappers.py.

```
00229     # command_run_md = "gmx mdrun -deffnm md -{} {} -c {} -pin on -reprod".format(thread_type, ncores, new_name)
00230     proc_obj = subprocess.Popen(command_run_md, stdout=-1, shell=True, cwd='{}/{}/'.format(work_dir, seed), stderr=-1, env=my_env)
00231     output, error = proc_obj.communicate()
00232     error = error.decode("utf-8")
00233     output = output.decode("utf-8")
00234     # with open(str(os.getpid())+'_err.log', 'a') as log_out:
00235     #     log_out.write(error)
```

```
00236     # with open(str(os.getpid())+'_out.log', 'a') as log_out:
00237     #     log_out.write(output.decode("utf-8"))
00238
00239     if 'error' in error.lower():
00240         print(error)
00241
00242
00243 def gmx_mdrun_mpi(work_dir: str, seed: int, new_name: str, hostnames: list, ncores: list = None, thread_type: str  = 'ntomp') -> NoReturn:
00244     """gmx mdrun - MPI version
00245
00246     Args:
00247          str  work_dir: path to work directory, where all seed directories reside
00248          int seed: seed value used in the MD simulation
Referenced by helper_funcs.make_a_step().
Here is the caller graph for this function:
```

3.12.1.5 gmx_mdrun_mpi()

```
NoReturn  gmx_wrappers.gmx_mdrun_mpi (
                str  work_dir,
                int  seed,
                str  new_name,
                list  hostnames,
                list   ncores =  None,
                str   thread_type =  'ntomp' )
```

gmx MPI version

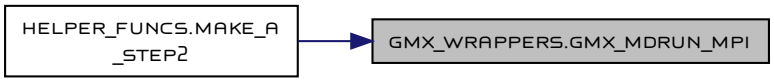
```
    str  work_dir: path to work directory, where all seed directories reside
    int  seed: seed value used in the MD simulation
    str  new_name: output name for a final state
    list  hostnames: must be a list
    list  ncores: number of cores to use in the current simulation
    str  thread_type: type of the thread, OMP ? MPI ?
```

Returns

Starts a shell in a separate process and runs mdrun there. This version uses MPI to run on a separate host

```
Definition at line 263 of file gmx_wrappers.py.
00263                             ".format(','.join(hostnames), new_name, int(ncores))
00264     else:
00265         if ncores:
00266             command_run_md = "mpirun -host {0} -np {1} mdrun -deffnm md -c {2} -ntomp 2 -nt {1} -pin on -reprod \
00267                             ".format(','.join(hostnames), min(1, int(ncores)), new_name)
00268             # command_run_md = "mpirun -host {} -np {} mdrun_mpi -deffnm md -c {} -ntomp 2 -pin on -reprod \
00269             #                  ".format(','.join(hostnames), min(1, int(ncores)//2), new_name)
00270         else:
00271             command_run_md = "mpirun -hosts {} gmx mdrun -deffnm md -c {} -ntomp 2 -pin on -reprod".format(','.join(hostnames), new_name)
00272     proc_obj = subprocess.Popen(command_run_md, stdout=-1, shell=True, cwd='{}/{}/'.format(work_dir, seed), stderr=-1, env=my_env)
00273     output, error = proc_obj.communicate()
00274     error = error.decode("utf-8")
00275     output = output.decode("utf-8")
00276     # with open(str(os.getpid())+'_err.log', 'a') as log_out:
00277     #     log_out.write(error)
00278     # with open(str(os.getpid())+'_out.log', 'a') as log_out:
00279     #     log_out.write(output.decode("utf-8"))
00280
00281     if 'error' in error.lower():
00282         print(error)
00283
00284
00285 def gmx_mdrun_mpi_with_sched(work_dir: str, seed: int, new_name: str, ncores: list = None, ntomp: int = 1) -> NoReturn:
00286     """gmx mdrun - MPI version with scheduler
00287
00288     Args:
00289          str  work_dir: path to work directory, where all seed directories reside
```

```
00290         int seed: seed value used in the MD simulation
```
Referenced by helper_funcs.make_a_step2().
Here is the caller graph for this function:



3.12.1.6 gmx_mdrun_mpi_with_sched()

```
NoReturn  gmx_wrappers.gmx_mdrun_mpi_with_sched (
                str  work_dir,
                int  seed,
                str  new_name,
                list   ncores =  None,
                int  ntomp =  1 )
```

gmx MPI version with scheduler

str work_dir: path to work directory, where all seed directories reside
int seed: seed value used in the MD simulation
str new_name: output name for a final state
list ncores: number of cores to use in the current simulation
int ntomp: number of OMP threads

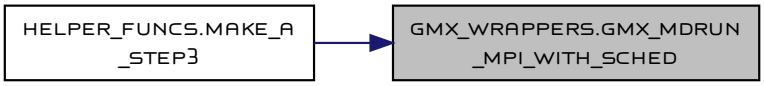
Returns

Starts a shell in a separate process and runs mdrun there. This version uses MPI but does not specify the host, it should be done through the scheduler. Do not use this version if you know the exact host names - then you have more control and potentially less overhead.

Definition at line 305 of file gmx_wrappers.py.

```
00305     else:
00306         command_run_md = "mpirun -np {0} mdrun -deffnm md -c {1} -ntomp {2} -pin on -reprod".format(ncores, new_name, ntomp)
00307 
00308     proc_obj = subprocess.Popen(command_run_md, stdout=-1, shell=True, cwd='{}/{}/'.format(work_dir, seed), stderr=-1, env=my_env)
00309     output, error = proc_obj.communicate()
00310     error = error.decode("utf-8")
00311     output = output.decode("utf-8")
00312     # with open(str(os.getpid())+'_err.log', 'a') as log_out:
00313     #     log_out.write(error)
00314     # with open(str(os.getpid())+'_out.log', 'a') as log_out:
00315     #     log_out.write(output.decode("utf-8"))
00316 
00317     if 'error' in error.lower():
00318         print(error)
00319 
00320 
00321 def gmx_grompp(work_dir: str, seed: int, top_file: str, prev_name: str) -> NoReturn:
00322     """gmx grompp (the gromacs preprocessor) reads a molecular topology file, checks the validity of the file,
00323      expands the topology from a molecular description to an atomic description.
00324 
00325     Args::
00326 
```

Referenced by helper_funcs.make_a_step3().
Here is the caller graph for this function:



3.12.1.7 gmx_trjcat()

```
NoReturn gmx_wrappers.gmx_trjcat (
                str  f,
                str  o,
                str  n,
                bool  cat =  True,
                bool  vel =  False,
                bool  sort =  False,
                bool  overwrite =  True )
```

'gmx trjcat' - GROMACS tool - concatenates several input trajectory files in sorted order
Outputs one .xtc file that contains all frames (99% frames are NOT sorted, since trajectories have the same time)

```
    str  f: Input trajectory: xtc trr cpt gro g96 pdb tng
    str  o: Output trajectory: xtc trr gro g96 pdb tng
    str  n: Index file
    bool   cat: Do not discard double time frames
    bool   vel: Read and write velocities if possible
    bool   sort: Sort trajectory files (not frames)
    bool   overwrite: Overwrite overlapping frames during appending
```

Returns

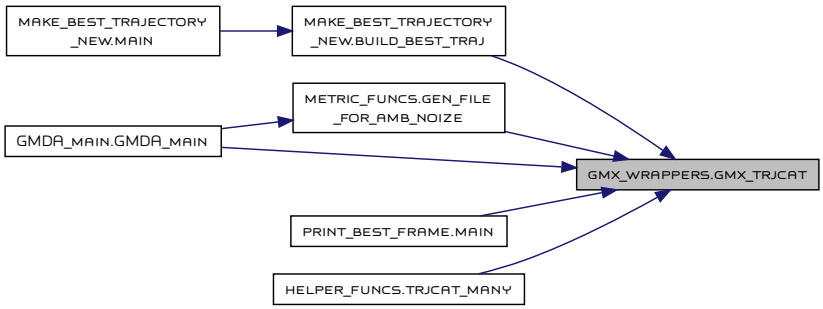
Generates one output file passed with -o parameter.

Definition at line 119 of file gmx_wrappers.py.

```
00119         command_trjcat += '-f ' + ' '.join(f) + ' '
00120     else:
00121         command_trjcat += '-f {:s} '.format(f)
00122     if n:
00123         command_trjcat += '-n {} '.format(n)
00124     if cat:
00125         command_trjcat += '-cat '
00126     else:
00127         command_trjcat += '-nocat '
00128     # if vel:
00129     #     command_trjcat += '-vel '
00130     # else:
00131     #     command_trjcat += '-novel '
00132     if sort:
00133         command_trjcat += '-sort '
00134     else:
00135         command_trjcat += '-nosort '
00136     if overwrite:
00137         command_trjcat += '-overwrite '
00138 
00139     command_trjcat = os.path.expandvars(command_trjcat)
00140     proc_obj = subprocess.Popen(command_trjcat, stdout=-1, shell=True, cwd='.', stderr=-1, env=my_env)
00141     output, error = proc_obj.communicate()
00142     error = error.decode("utf-8")
00143     if 'error' in error.lower():
00144         print(error)
00145 
00146 
00147 def gmx_eneconv(f: str, o: str) -> NoReturn:
00148     """'gmx eneconv' - GROMACS tool - Concatenates several energy files in sorted order
00149 
00150     Stores converted energy files. Not used by main algorithm, but during the postprocessing.
00151 
00152     Args:
```

Referenced by make_best_trajectory_new.build_best_traj(), metric_funcs.gen_file_for_amb_noize(), GMDA_main.GMDA_main(), print_best_frame.main(), and helper_funcs.trjcat_many().

Here is the caller graph for this function:



3.12.1.8 gmx_trjconv()

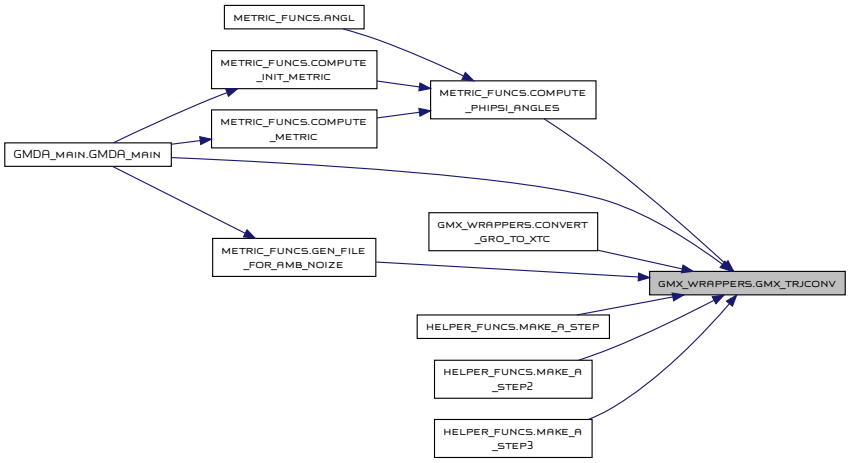
```
NoReturn gmx_wrappers.gmx_trjconv (
                str  f,
                str  o,
                str   n =  None,
                str   s =  None,
                int  b =  None,
                int  e =  None,
                int  dump =  None,
                str   fit =  None,
                str   vel =  None,
                str   pbc =  None )
```

Definition at line 64 of file gmx_wrappers.py.

```
00064         command_trjconv += '-n {} '.format(n)
00065     if s:
00066         command_trjconv += '-s {} '.format(s)
00067     if b:
00068         command_trjconv += '-b {} '.format(b)
00069     if e:
00070         command_trjconv += '-e {} '.format(e)
00071     if dump:
00072         command_trjconv += '-dump {} '.format(dump)
00073     # if vel:
00074     #     command_trjconv += '-vel '
00075     # else:
00076     #     command_trjconv += '-novel '
00077     if fit:
00078         if fit not in ['none', 'rot+trans', 'rotxy+transxy', 'translation', 'transxy', 'progressive']:
00079             raise Exception('Wrong fit parameter in gmx_trjconv.')
00080         command_trjconv += '-fit {} '.format(fit)
00081     if pbc:
00082         if pbc not in ['none', 'mol', 'res', 'atom', 'nojump', 'cluster', 'whole']:
00083             raise Exception('Wrong pbc parameter in gmx_trjconv.')
00084         command_trjconv += '-pbc {} '.format(pbc)
00085
00086     # command_trjconv = os.path.expandvars(command_trjconv)
00087     # print(command_trjconv)
00088     proc_obj = subprocess.Popen(command_trjconv, stdout=-1, shell=True, cwd='.', stderr=-1, env=my_env)
00089     output, error = proc_obj.communicate()
00090     error = error.decode("utf-8")
00091     if 'error' in error.lower():
00092         print(error)
00093     # print(output.decode("utf-8"))
00094     # print(error)
00095
00096
00097 def gmx_trjcat(f: str, o: str, n: str, cat: bool = True, vel: bool = False, sort: bool = False, overwrite: bool = True) -> NoReturn:
00098     """'gmx trjcat' - GROMACS tool - concatenates several input trajectory files in sorted order
00099
00100     Outputs one .xtc file that contains all frames (99% frames are NOT sorted, since trajectories have the same time)
00101
00102     Args:
```

Referenced by metric_funcs.compute_phipsi_angles(), convert_gro_to_xtc(), metric_funcs.gen_file_for_amb_noize(), GMDA_main.GMDA_main(), helper_funcs.make_a_step(), helper_funcs.make_a_step2(), and helper_funcs.make_a_step3().

Here is the caller graph for this function:



3.12.2 Variable Documentation

3.12.2.1 my_env `gmx_wrappers.my_env = os.environ.copy()`

Definition at line 15 of file gmx_wrappers.py.

3.13 helper_funcs Namespace Reference

Functions

- str get_digest (str in_str)

  Computes digest of the input string.
- list create_core_mapping (int ncores=mp.cpu_count(), int nseeds=1)

  Tries to map cores evenly among tasks.
- list get_previous_runs_info (str check_dir)

  Scans direcotory for prior results and outputs the list of filenames.
- def check_precomputed_noize (str an_file, list metr_order)

  Checks whether file with precomputed ambient noise exists.
- NoReturn make_a_step (str work_dir, int cur_seed, dict seed_dirs, str top_file, str ndx_file, str seed_digest_filename, str old_name↩_digest, str past_dir, int ncores=1)

  Version for the case when you use one machine, for example, local computer or one remote server.
- NoReturn make_a_step2 (str work_dir, int cur_seed, dict seed_dirs, str top_file, str ndx_file, str seed_digest_filename, str old_↩name_digest, str past_dir, str hostname, int ncores)

  Version for the case when you use cluster and have hostnames.
- NoReturn make_a_step3 (str work_dir, int cur_seed, dict seed_dirs, str top_file, str ndx_file, str seed_digest_filename, str old_↩name_digest, str past_dir, int ncores, int ntomp=1)

  Version for the case when you use scheduler and have many cores, but no hostnames.
- dict get_seed_dirs (str work_dir, list list_with_cur_seeds, int simulation_temp, dict sd=None)

  Create directories with unique names for simulation with specified seeds and puts .mdp, config files for the MD simulation.
- NoReturn rm_seed_dirs (dict seed_dirs)

  Removes seed directory and all it's content.
- list get_new_seeds (list old_seeds, int seed_num=4)

  Returns next seed sequence.
- NoReturn trjcat_many (list hashed_names, str past_dir, str out_name)

  Concatenates many trajectories into one file.
- NoReturn general_bak (str fname, tuple state)

Stores variables in the picke with the specific name.

- tuple general_rec (str fname)

  Reads pickle content from the file.

- NoReturn main_state_backup (tuple state)

  Just a wrapper around the general_bak.

- NoReturn supp_state_backup (tuple state)

  Just a wrapper around the general_bak.

- tuple main_state_recover ()

  Just a wrapper around the general_rec.

- tuple supp_state_recover ()

  Just a wrapper around the general_rec.

3.13.1 Function Documentation

3.13.1.1 check_precomputed_noize()

```
def helper_funcs.check_precomputed_noize (
        str  an_file,
        list  metr_order )
```

Checks whether file with precomputed ambient noise exists.
Tries to read correct number of metrics, in case of error throws and exception Otherwise returns dict {metric_name: noise_value}

str an_file: ambient noise filename to check
list metr_order: order of metric names (should be correct sequence)

Returns

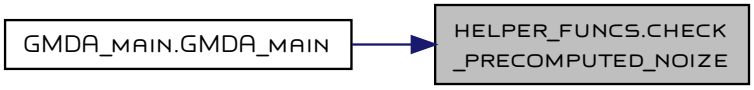
:return: dict {metric_name: noise_value} :rtype: dict or None

Definition at line 118 of file helper_funcs.py.

```
00118     """
00119     # TODO: rewrite function to save noise and metric name, so you do not read the wrong sequence (add a check)
00120     if an_file in os.walk(".").__next__()[2]:
00121         print(an_file, ' was found. Reading... ')
00122         with open(an_file, 'r') as f:
00123             noize_arr = f.readlines()
00124         try:
00125             res_arr = [float(res.strip()) for res in noize_arr]
00126             err_node = dict ()
00127             for i in range(len(res_arr)):
00128                 err_node[metr_order[i]] = res_arr[i]
00129         except Exception as e:
00130             print(e)
00131             return None
00132         return err_node
00133     return None
00134
00135
```

Referenced by GMDA_main.GMDA_main().

Here is the caller graph for this function:



3.13.1.2 create_core_mapping()

```
list helper_funcs.create_core_mapping (
                int  ncores =  mp.cpu_count(),
                int  nseeds =  1 )
```

Tries to map cores evenly among tasks.

int ncores: number of cores available
int nseeds: number of seeds used in current run

Returns

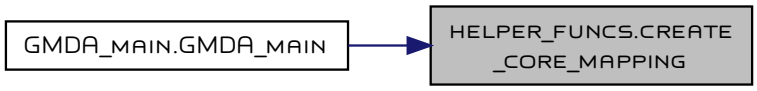
:return: list of tuple s, each tuple consist of (cores number, task identifier) :rtype: list

Definition at line 50 of file helper_funcs.py.

```
00050     """
00051     ncores = ncores if ncores > 0 else 1
00052     nseeds = nseeds if nseeds > 0 else 1
00053     print('I will use {} cores for {} seeds'.format(ncores, nseeds))
00054 
00055     even = ncores // nseeds
00056     remainder = ncores % nseeds
00057 
00058     sched_arr = list()
00059     if even:
00060         cur_sched = [(even+1, i) if i < remainder else (even, i) for i in range(nseeds)]
00061         sched_arr.append(cur_sched)
00062     else:
00063         seeds_range_iter = iter(range(nseeds))
00064         tot_batches = nseeds//ncores
00065         remainder = nseeds-tot_batches*ncores
00066         tot_batches = tot_batches if not remainder else tot_batches+1  # if we can't divide tasks evenly, we need one more batch
00067         for i in range(tot_batches):
00068             if i < tot_batches-1:
00069                 cur_sched = [(1, 0)]*ncores
00070             else:
00071                 cur_sched = [(1, 0) if i < remainder else (0, 0) for i in range(ncores)]
00072                 free_cores = ncores - sum(i for i, j in cur_sched)
00073                 if free_cores:
00074                     cur_sched = [(j[0]+1, 0) if i < free_cores else (j[0], 0) for i, j in enumerate(cur_sched)]
00075             sched_arr.append(cur_sched)
00076         for i, cur_sched in enumerate(sched_arr):
00077             for j, cornum_seed in enumerate(cur_sched):
00078                 if cornum_seed[0]:
00079                     cur_seed = next(seeds_range_iter)
00080                     sched_arr[i][j] = (cornum_seed[0], cur_seed)
00081                     print('Seed {} will be run on {} cores.'.format(cur_seed, cornum_seed[0]))
00082 
00083     return sched_arr
00084 
00085 
```

Referenced by GMDA_main.GMDA_main().

Here is the caller graph for this function:



3.13.1.3 general_bak()

```
NoReturn  helper_funcs.general_bak (
                str  fname,
                tuple  state )
```

Stores variables in the picke with the specific name.

str fname: filename for the pickle
tuple state: variables to store

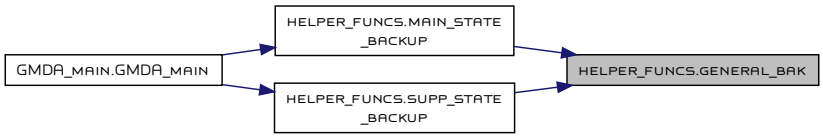
Returns

Generates a file with pickled data.

Definition at line 341 of file helper_funcs.py.

```
00341     """
00342     if os.path.exists(os.path.join(os.getcwd(), fname)):
00343         try:
00344             os.rename(os.path.join(os.getcwd(), fname), os.path.join(os.getcwd(), fname + '_prev'))
00345         except Exception as e:
00346             # print(e)
00347             os.remove(os.path.join(os.getcwd(), fname))
00348             os.rename(os.path.join(os.getcwd(), fname), os.path.join(os.getcwd(), fname + '_prev'))
00349
00350     with open(fname, 'wb') as f:
00351         pickle.dump(state, f)
00352
00353
```

Referenced by main_state_backup(), and supp_state_backup().
Here is the caller graph for this function:



3.13.1.4 general_rec()

tuple helper_funcs.general_rec (str fname)

Reads pickle content from the file.

str fname: pickle filename

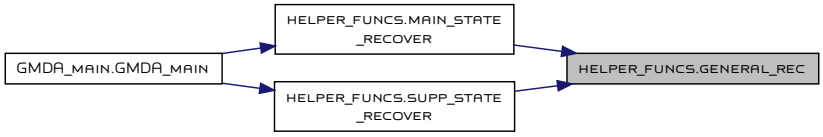
Returns

:return: state from the pickle :rtype: tuple

Definition at line 363 of file helper_funcs.py.

```
00363     """
00364     with open(fname, 'rb') as f:
00365         state = pickle.load(f)
00366     return state
00367
00368
```

Referenced by main_state_recover(), and supp_state_recover().
Here is the caller graph for this function:



3.13.1.5 get_digest()

str helper_funcs.get_digest (str in_str)

Computes digest of the input string.

str in_str: typically list of seeds concatenated with _. like s_0_1_5

Returns

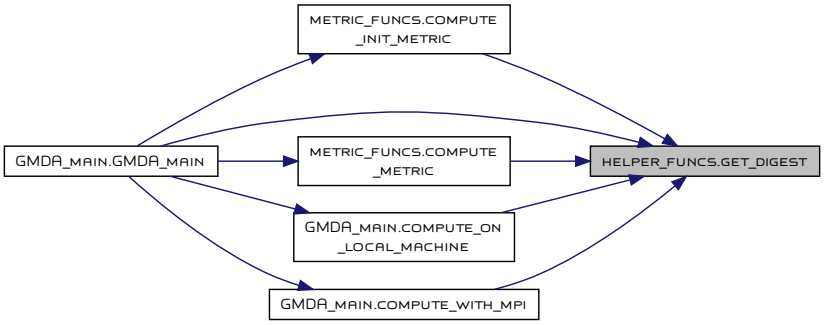
:return: blake2 hash of the in_str. We use short version, but you can use full version - slightly slower, but less chances of name collision.
:rtype: str

Definition at line 34 of file helper_funcs.py.

```
00034     """
00035     # return hashlib.md5(in_str.encode()).hexdigest()
00036     # if you have python older than 3.6 - use md5 or update python
00037     return hashlib.blake2s(in_str.encode()).hexdigest()
00038
00039
```

Referenced by metric_funcs.compute_init_metric(), metric_funcs.compute_metric(), GMDA_main.compute_on_local_machine(), GMDA_main.compute_with_mpi(), and GMDA_main.GMDA_main().

Here is the caller graph for this function:



3.13.1.6 get_new_seeds()

```
list helper_funcs.get_new_seeds (
                list  old_seeds,
                int  seed_num =  4 )
```

Returns next seed sequence.

list old_seeds: list of previous seeds
int seed_num: number of unique seeds in the current run

Returns

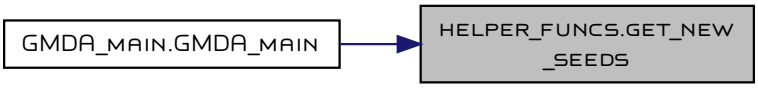
:return: list of new seeds :rtype list

Definition at line 297 of file helper_funcs.py.

```
00297     """
00298     max_seeds = 64000  # change this if you want more exploration
00299     if min(old_seeds) + seed_num > max_seeds:
00300         return None
00301     return [seed + seed_num for seed in old_seeds]
00302
00303
```

Referenced by GMDA_main.GMDA_main().

Here is the caller graph for this function:



3.13.1.7 get_previous_runs_info()

```
list helper_funcs.get_previous_runs_info (
        str  check_dir )
```

Scans direcotory for prior results and outputs the list of filenames.

```
str  check_dir:  directory to scan for prior trajectories
```

Returns

```
:return: list of filenames .xtc or .gro :rtype: list
```

Definition at line 95 of file helper_funcs.py.

```
00095     """
00096     # filenames_found = os.walk(check_dir).__next__()[2]
00097     filenames_found = [f.split("/")[-1] for f in os.listdir(check_dir)]
00098     # filenames_found = [f.path.split("/")[-1] for f in os.scandir(check_dir)]
00099     filenames_found_important = [f for f in filenames_found if f.split('.')[1] in ['xtc', 'gro']]
00100     del filenames_found
00101     print('Found files: {} with .gro and .xtc'.format(len(filenames_found_important)))
00102     return filenames_found_important
00103
00104
```

Create directories with unique names for simulation with specified seeds and puts .mdp, config files for the MD simulation.

```
str  work_dir: path to work directory, where all seed directories reside
list  list_with_cur_seeds:   list  of seed currently used
int  simulation_temp: simulation temperature used to generate proper .mdp file
dict   sd: Not used anymore, but left for sime time as deprecated. sd - previous seed deers
```

Returns

```
:return: dict ionary with seed dir paths :rtype dict
```

Definition at line 261 of file helper_funcs.py.

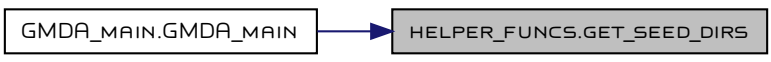
```
00261     """
00262     if not sd:
00263         sd = dict ()
00264     for seed in list_with_cur_seeds:
00265         seed_dir = os.path.join(work_dir, str(seed))
00266         sd[seed] = seed_dir
00267         if not os.path.exists(seed_dir):
00268             os.makedirs(seed_dir)
00269         with open(os.path.join(sd[seed], 'md.mdp'), 'w') as f:
00270             f.write(get_mdp(seed, simulation_temp))
00271     return sd
00272
00273
```

References gen_mdp.get_mdp().

Referenced by GMDA_main.GMDA_main().

Here is the call graph for this function:

Here is the caller graph for this function:



3.13.1.8 main_state_backup()

NoReturn helper_funcs.main_state_backup (tuple state)

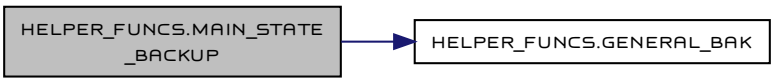
Just a wrapper around the general_bak.

tuple state: (visited_queue, open_queue, main_dict)

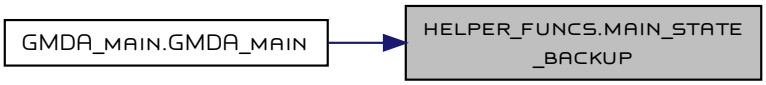
Definition at line 374 of file helper_funcs.py.

```
00374     """
00375     general_bak('small.pickle', state)
00376
00377
```

References general_bak().
Referenced by GMDA_main.GMDA_main().
Here is the call graph for this function:



Here is the caller graph for this function:



3.13.1.9 main_state_recover()

tuple helper_funcs.main_state_recover ( )

Just a wrapper around the general_rec.

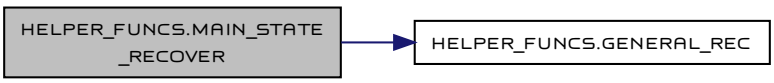
Returns

:return: state from the pickle

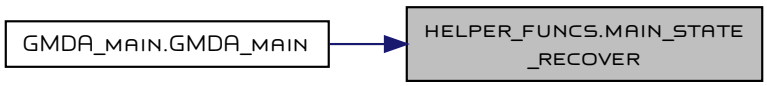
Definition at line 396 of file helper_funcs.py.

```
00396
00397
00398 def supp_state_recover() -> tuple :
00399     """Just a wrapper around the general_rec
```

References general_rec().
Referenced by GMDA_main.GMDA_main().
Here is the call graph for this function:



Here is the caller graph for this function:



3.13.1.10 make_a_step()

```
NoReturn helper_funcs.make_a_step (
                str  work_dir,
                int  cur_seed,
                dict  seed_dirs,
                str  top_file,
                str  ndx_file,
                str  seed_digest_filename,
                str  old_name_digest,
                str  past_dir,
                int  ncores =  1 )
```

Version for the case when you use one machine, for example, local computer or one remote server.
Generates the actual MD simulation by first - setting the simulation with grommp, then using several mdruns, and finally conctatenating the result into the one file.

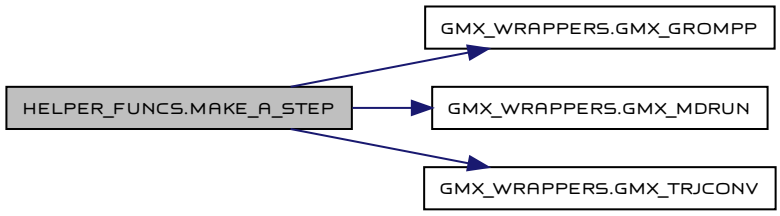
```
    str  work_dir: path to the directory where seed dirs reside
    int  cur_seed: current seed value used for MD production
    dict   seed_dirs:  dict   which contains physical path to
    the directory where simulation with particular seed is performed
    str  top_file: .top - topology of the current conformation
    str  ndx_file: .ndx - index of the protein atoms of the current conformation
    str  seed_digest_filename: digest for a current MD simulation, used to store files in the past
    str  old_name_digest: digest for a prior MD simulation
    str  past_dir: path to the directory with prior computations
    int  ncores: number of cores to use for this task
```

Definition at line 153 of file helper_funcs.py.

```
00153          int ncores: number of cores to use for this task
00154     """
00155     # global extra_past
00156     old_name = os.path.join(past_dir, old_name_digest)
00157     if not os.path.exists(old_name+'.gro'):
00158         # old_name = os.path.join(extra_past, old_name_digest)
00159         # if not os.path.exists(old_name + '.gro'):
00160         raise Exception("make_a_step: did not find {} in {} ".format(old_name_digest, past_dir))
00161     gmx_grompp(work_dir, cur_seed, top_file, old_name)
00162     new_name = os.path.join(past_dir, seed_digest_filename)
00163     gmx_mdrun(work_dir, cur_seed, new_name + '.gro', ncores)
00164     gmx_trjconv(f=os.path.join(seed_dirs[cur_seed], 'md.xtc'), o='{}.xtc'.format(new_name),
00165                 n=ndx_file, s=os.path.join(seed_dirs[cur_seed], 'md.tpr'), pbc='mol', b=1)
00166     try:
00167         cp2(os.path.join(seed_dirs[cur_seed], 'md.edr'), '{}.edr'.format(new_name))
00168     except:
00169         print('Error when tried to copy energy file. Maybe you do not produce them ? Then comment this line.')
00170     os.remove(os.path.join(seed_dirs[cur_seed], 'md.xtc'))
00171
00172
```

References gmx_wrappers.gmx_grompp(), gmx_wrappers.gmx_mdrun(), and gmx_wrappers.gmx_trjconv().

Here is the call graph for this function:



3.13.1.11 make_a_step2()

```
NoReturn helper_funcs.make_a_step2 (
                str  work_dir,
                int  cur_seed,
                dict  seed_dirs,
                str  top_file,
                str  ndx_file,
                str  seed_digest_filename,
                str  old_name_digest,
                str  past_dir,
                str  hostname,
                int  ncores )
```

Version for the case when you use cluster and have hostnames.
Generates the actual MD simulation by first - setting the simulation with grommp, then using several mdruns, and finally conctatenating the result into the one file.

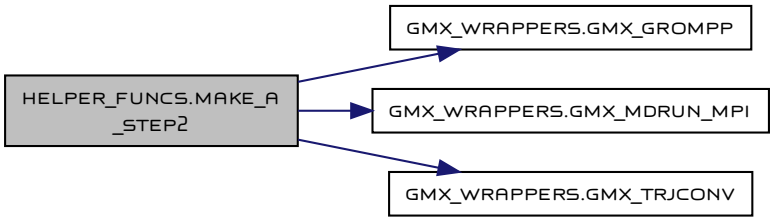
str work_dir: path to the directory where seed dirs reside
int cur_seed: current seed value used for MD production
dict seed_dirs: dict which contains physical path to the directory
where simulation with particular seed is performed
str top_file: .top - topology of the current conformation
str ndx_file: .ndx - index of the protein atoms of the current conformation
str seed_digest_filename: digest for a current MD simulation, used to store files in the past
str old_name_digest: digest for a prior MD simulation
str past_dir: path to the directory with prior computations
str hostname: hostname to use for MD simulation
intncores: number of cores to use for this task

Definition at line 191 of file helper_funcs.py.

```
00191          intncores: number of cores to use for this task
00192     """
00193     # global extra_past
00194     old_name = os.path.join(past_dir, old_name_digest)
00195     if not os.path.exists(old_name + '.gro'):
00196         # old_name = os.path.join(extra_past, old_name_digest)
00197         # if not os.path.exists(old_name + '.gro'):
00198         raise Exception("make_a_step2: did not find {} in {}".format(old_name_digest, past_dir))
00199     gmx_grompp(work_dir, cur_seed, top_file, old_name)
00200     new_name = os.path.join(past_dir, seed_digest_filename)
00201     gmx_mdrun_mpi(work_dir, cur_seed, new_name + '.gro', hostname, ncores)
00202     gmx_trjconv(f=os.path.join(seed_dirs[cur_seed], 'md.xtc'), o='{}.xtc'.format(new_name),
00203                 n=ndx_file, s=os.path.join(seed_dirs[cur_seed], 'md.tpr'), pbc='mol', b=1)
00204     try:
00205         cp2(os.path.join(seed_dirs[cur_seed], 'md.edr'), '{}.edr'.format(new_name))
00206     except:
00207         print('Error when tried to copy energy file. Maybe you do not produce them ? Then comment this line.')
00208     os.remove(os.path.join(seed_dirs[cur_seed], 'md.xtc'))
00209
00210
```

References gmx_wrappers.gmx_grompp(), gmx_wrappers.gmx_mdrun_mpi(), and gmx_wrappers.gmx_trjconv().

Here is the call graph for this function:



3.13.1.12 make_a_step3()

```
NoReturn helper_funcs.make_a_step3 (
                str  work_dir,
                int  cur_seed,
                dict  seed_dirs,
                str  top_file,
                str  ndx_file,
                str  seed_digest_filename,
                str  old_name_digest,
                str  past_dir,
                int  ncores,
                int  ntomp =  1 )
```

Version for the case when you use scheduler and have many cores, but no hostnames.
Generates the actual MD simulation by first - setting the simulation with grommp, then using several mdruns, and finally conctatenating the result into the one file.

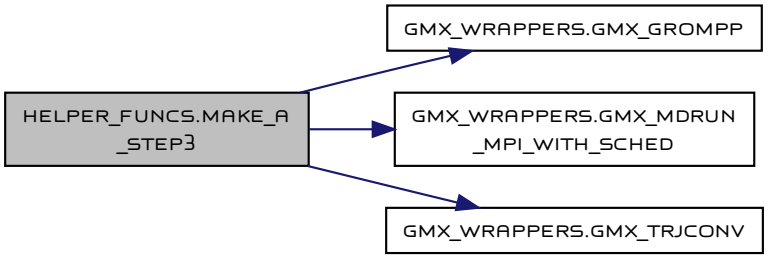
str work_dir: path to the directory where seed dirs reside
int cur_seed: current seed value used for MD production
dict seed_dirs: dict which contains physical path to the directory where simulation with particular seed is performed
str top_file: .top - topology of the current conformation
str ndx_file: .ndx - index of the protein atoms of the current conformation
str seed_digest_filename: digest for a current MD simulation, used to store files in the past
str old_name_digest: digest for a prior MD simulation
str past_dir: path to the directory with prior computations
int ncores: number of cores to use for this task
int ntomp: number of OMP threads to use during the simulation

Definition at line 228 of file helper_funcs.py.

```
00228          int ntomp: number of OMP threads to use during the simulation
00229     """
00230     # global extra_past
00231     old_name = os.path.join(past_dir, old_name_digest)
00232     if not os.path.exists(old_name +'.gro'):
00233         # old_name = os.path.join(extra_past, old_name_digest)
00234         # if not os.path.exists(old_name + '.gro'):
00235         raise Exception("make_a_step3: did not find {} in {}".format(old_name_digest, past_dir))
00236     gmx_grompp(work_dir, cur_seed, top_file, old_name)
00237     new_name = os.path.join(past_dir, seed_digest_filename)
00238     # gmx_mdrun_mpi(work_dir, cur_seed, new_name + '.gro', hostname, ncores)
00239     gmx_mdrun_mpi_with_sched(work_dir, cur_seed, new_name + '.gro', ncores, ntomp)
00240     gmx_trjconv(f=os.path.join(seed_dirs[cur_seed], 'md.xtc'), o='{}.xtc'.format(new_name),
00241                 n=ndx_file, s=os.path.join(seed_dirs[cur_seed], 'md.tpr'), pbc='mol', b=1)
00242     try:
00243         cp2(os.path.join(seed_dirs[cur_seed], 'md.edr'), '{}.edr'.format(new_name))
00244     except:
00245         print('Error when tried to copy energy file. Maybe you do not produce them ? Then comment this line.')
00246     os.remove(os.path.join(seed_dirs[cur_seed], 'md.xtc'))
00247
00248
```

References gmx_wrappers.gmx_grompp(), gmx_wrappers.gmx_mdrun_mpi_with_sched(), and gmx_wrappers.gmx_trjconv().

Here is the call graph for this function:



3.13.1.13 rm_seed_dirs()

NoReturn helper_funcs.rm_seed_dirs (dict seed_dirs)

Removes seed directory and all it's content.

dict seed_dirs: dict which contains physical path to the directory where simulation with particular seed is performed

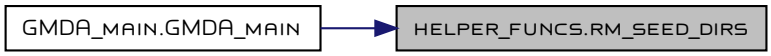
Removes old working directories to save disc space.

Definition at line 281 of file helper_funcs.py.

```
00281     """
00282     for seed_dir in seed_dirs.values():
00283         if os.path.exists(seed_dir):
00284             shutil.rmtree(seed_dir, ignore_errors=True)
00285 
00286 
```

Referenced by GMDA_main.GMDA_main().

Here is the caller graph for this function:



3.13.1.14 supp_state_backup()

NoReturn helper_funcs.supp_state_backup (tuple state)

Just a wrapper around the general_bak.

tuple state: (tol_error, seed_list, seed_dirs, seed_change_counter, skipped_counter, cur_metric_name,

cur_metric, counter_since_seed_changed, guiding_metric, greed_mult,
best_so_far_name, best_so_far, greed_count)

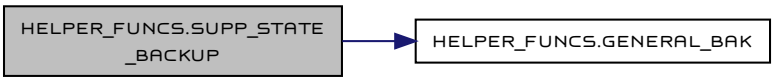
Definition at line 387 of file helper_funcs.py.

```
00387 
00388 
00389 def main_state_recover() -> tuple :
00390     """Just a wrapper around the general_rec
```

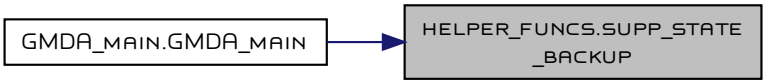
References general_bak().

Referenced by GMDA_main.GMDA_main().

Here is the call graph for this function:



Here is the caller graph for this function:



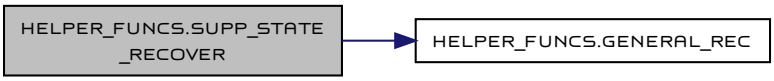
3.13.1.15 supp_state_recover() tuple helper_funcs.supp_state_recover ( )

Just a wrapper around the general_rec.

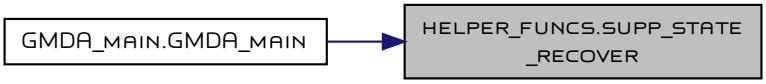
Returns

:return: state from the pickle

Definition at line 405 of file helper_funcs.py.
References general_rec().
Referenced by GMDA_main.GMDA_main().
Here is the call graph for this function:



Here is the caller graph for this function:



3.13.1.16 trjcat_many()

```
NoReturn helper_funcs.trjcat_many (
                list  hashed_names,
                str  past_dir,
                str  out_name )
```

Concatenates many trajectories into one file.

```
    list  hashed_names: .xtc filenames to concatenate
    str  past_dir: path to the directory with prior computations
    str  out_name: single output filename
```

Returns

Generates one file with many frames.

Definition at line 314 of file helper_funcs.py.

```
00314     """
00315     wave = 100
00316     tot_chunks = int((len(hashed_names) + 1) / wave)
00317     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00318     gmx_trjcat(f=[os.path.join(past_dir, hashed_name) + '.xtc' for hashed_name in hashed_names[:wave]],
00319                o='./combinded_traj.xtc', n='./prot_dir/prot.ndx', cat=True, vel=False, sort=False, overwrite=True)
00320     for i in range(wave, len(hashed_names), wave):
00321         os.rename('./combinded_traj.xtc', './combinded_traj_prev.xtc')
00322         gmx_trjcat(f=[" ./combinded_traj_prev.xtc "] + [os.path.join(past_dir, hashed_name) + '.xtc' for hashed_name in
       hashed_names[i:i+wave]],
00323                    o='./combinded_traj.xtc',
00324                    n='./prot_dir/prot.ndx', cat=True, vel=False, sort=False, overwrite=True)
00325         if int(i / wave) % 10 == 0:
00326             print('{}/{} ({:.1f}%)'.format(int(i / wave), tot_chunks, 100 * int(i / wave) / tot_chunks))
00327     if os.path.exists('./combinded_traj_prev.xtc'):
00328         os.remove('./combinded_traj_prev.xtc')
00329     os.rename('./combinded_traj.xtc', out_name)
00330
00331
```

References gmx_wrappers.gmx_trjcat().

Here is the call graph for this function:

3.14 main Namespace Reference

Functions

- def main ()

  This function is basically a launcher.

3.14.1 Function Documentation

3.14.1.1 main()

```
def main.main ( )
```

This function is basically a launcher.

Parallel threads did not result in a much better performance and was masked for better times. However, if you decide to implement C++ parallel I/O - it should help.

Definition at line 25 of file main.py.

```
00025     """
00026     # Compilation steps:
00027     # compile latest gcc
00028     # compile gromacs with shared libs and static libs, without mpi; install
00029     # compile mdsctk
00030     # OPTIONAL: compile gromacs with mpi/openmp if needed.
00031     tot_seeds = 4
00032     # get_db_con(tot_seeds=4)
00033
00034     past_dir = os.path.join(os.getcwd(), 'past/')
00035     #
00036     # PRINT_LOCK = Lock()
```

```
00037     # COPY_LOCK = Lock()
00038     # RM_LOCK = Lock()
00039
00040     # print_queue = queue.Queue()
00041     # printing_thread = Thread(target=threaded_print, args=(print_queue,))
00042     # printing_thread.start()
00043
00044     # db_input_queue = queue.Queue()
00045     # db_input_thread = Thread(target=threaded_db_input, args=(db_input_queue, tot_seeds,))
00046     # db_input_thread.start()
00047     # # db_input_queue.put(None)
00048     #
00049     # copy_queue = queue.Queue()
00050     # copy_thread = Thread(target=threaded_copy, args=(copy_queue,))
00051     # copy_thread.start()
00052     #
00053     # rm_queue = queue.Queue()
00054     # rm_thread = Thread(target=threaded_rm, args=(rm_queue, RM_LOCK,))
00055     # rm_thread.start()
00056
00057     # prev_runs_files = get_previous_runs_info(past_dir)
00058     prev_runs_files = None
00059
00060     print_queue = multiprocessing.JoinableQueue(102400)
00061     printing_thread = multiprocessing.Process(target=threaded_print, args=(print_queue,))
00062     printing_thread.start()
00063
00064     db_input_queue = multiprocessing.JoinableQueue(102400)
00065     db_input_thread = multiprocessing.Process(target=threaded_db_input, args=(db_input_queue, tot_seeds,))
00066     db_input_thread.start()
00067
00068     # no need in the next queues. Maybe helpful if working with /dev/shm
00069     copy_queue = None
00070     # copy_queue = multiprocessing.Queue()
00071     # copy_thread = multiprocessing.Process(target=threaded_copy, args=(copy_queue,))
00072     # copy_thread.start()
00073
00074     rm_queue = None
00075     # rm_queue = multiprocessing.JoinableQueue(3)
00076     # rm_thread = multiprocessing.Process(target=threaded_rm, args=(rm_queue,))
00077     # rm_thread.start()
00078
00079     GMDA_main(prev_runs_files, past_dir, print_queue, db_input_queue, copy_queue, rm_queue, tot_seeds)
00080
00081     printing_thread.join()
00082     db_input_thread.join()
00083     print_queue.put_nowait(None)
00084     db_input_queue.put_nowait(None)
00085     rm_queue.put_nowait(None)
00086     # print_queue.join()
00087     # db_input_queue.join()
00088     # rm_queue.join()
00089
00090
```

3.15 make_best_trajectory_new Namespace Reference

Functions

- def main ()
- def build_best_traj (str metr_name, str db_to_connect)

  Finds the lowest value of the metric and builds the trajectory that leads to this point.
- def main_energy ()

3.15.1 Function Documentation

3.15.1.1 build_best_traj() def make_best_trajectory_new.build_best_traj (str metr_name, str db_to_connect)

Finds the lowest value of the metric and builds the trajectory that leads to this point.
Once best value is found, we search for a name, parse it (name consist of prev seeds separated by _). Once we have all the preceeding seeds, we can extract their frames and join them.

Parameters str metr_name: str db_to_connect:

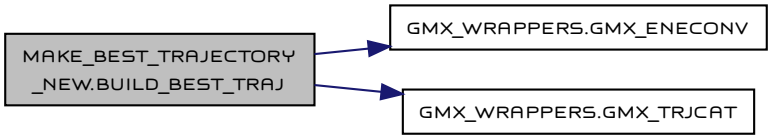
Returns Generates one .xtc trajectory with frames that result in the best conformation according to the specific metric.

Definition at line 55 of file make_best_trajectory_new.py.

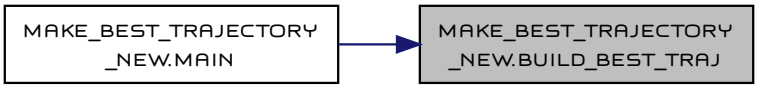
```
00055     """
00056
00057     # db_to_connect = 'results_opls_trp_300_2_fixed'
00058
00059     past_dir = './past'
00060     if not os.path.exists(db_to_connect + '.sqlite3'):
00061         raise Exception('DB not found')
00062
00063     con = lite.connect(db_to_connect + '.sqlite3', check_same_thread=False, isolation_level=None)
00064     cur = con.cursor()
00065
00066     qry = "select a.name, a.hashed_name, a.{0}_goal_dist from main_storage a \
00067            where a.{0}_goal_dist= ( select min(b.{0}_goal_dist) from main_storage b)".format(metr_name)
00068     result = cur.execute(qry)
00069     all_res = result.fetchone()
00070     print('The closest frame to goal has {} {} and name:\n{}'.format(metr_name, all_res[2], all_res[1]))
00071     name = all_res[0]
00072     spname = name.split('_')
00073     all_prev_names = ['\'{}\"'.format('_'.join(spname[:i])) for i in range(1, len(spname)+1)]
00074     long_line = ", ".join(all_prev_names)
00075
00076     qry = "select name, hashed_name from main_storage where name in ({})".format(long_line)
00077     result = cur.execute(qry)
00078     all_res = result.fetchall()
00079     con.close()
00080
00081     names, hashed_names = zip(*all_res)
00082
00083     # for file in [os.path.join(past_dir, hashed_name) for hashed_name in hashed_names]:
00084     #     copy2('{}.xtc'.format(file), './best_past/')
00085     #     try:
00086     #         copy2('{}.edr'.format(file), './best_past/')
00087     #     except:
00088     #         print('Failed to copy {}; Normal for the first frame.'.format(file))
00089
00090     wave = 100
00091     tot_chunks = int((len(hashed_names) + 1) / wave)
00092     print('Computing best trajectory for {}'.format(metr_name))
00093     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00094     if os.path.exists('./{}_combined_traj.xtc'.format(metr_name)):
00095         os.remove('./{}_combined_traj.xtc'.format(metr_name))
00096     if os.path.exists('./{}_combined_traj_prev.xtc'.format(metr_name)):
00097         os.remove('./{}_combined_traj_prev.xtc'.format(metr_name))
00098
00099     gmx_trjcat(f=[os.path.join(past_dir, hashed_name) + '.xtc' for hashed_name in hashed_names[:wave]],
00100                o='./{}_combined_traj.xtc'.format(metr_name), n='./prot_dir/prot_unfolded.ndx', cat=True, vel=False, sort=False, overwrite=True)
00101     for i in range(wave, len(hashed_names), wave):
00102         os.rename('./{}_combined_traj.xtc'.format(metr_name), './{}_combined_traj_prev.xtc'.format(metr_name))
00103         gmx_trjcat(f=[" ./{}_combined_traj_prev.xtc ".format(metr_name)] + [os.path.join(past_dir, hashed_name) + '.xtc' for hashed_name in
      hashed_names[i:i+wave]],
00104                    o='./{}_combined_traj.xtc'.format(metr_name), n='./prot_dir/prot_unfolded.ndx', cat=True, vel=False, sort=False,
      overwrite=True)
00105         if int(i / wave) % 10 == 0:
00106             print('{}/{} ({:.1f}%)'.format(int(i / wave), tot_chunks, 100 * int(i / wave) / tot_chunks))
00107
00108     if os.path.exists('./{}_combined_traj.xtc'.format(metr_name)):
00109         os.rename('./{}_combined_traj.xtc'.format(metr_name), './{}_{}_traj_best.xtc'.format(metr_name, db_to_connect))
00110     if os.path.exists('./{}_combined_traj_prev.xtc'.format(metr_name)):
00111         os.remove('./{}_combined_traj_prev.xtc'.format(metr_name))
00112     print('Done with best for {}: {}'.format(metr_name, db_to_connect))
00113
00114
00115     # ###### ENERGIES
00116     if os.path.exists('./{}_combined_energy.edr'.format(metr_name)):
00117         os.remove('./{}_combined_energy.edr'.format(metr_name))
00118     if os.path.exists('./{}_combined_energy_prev.edr'.format(metr_name)):
00119         os.remove('./{}_combined_energy_prev.edr'.format(metr_name))
00120     hashed_names = hashed_names[1:]
00121     tot_chunks = int((len(hashed_names) + 1) / wave)
00122     print('Computing energy for best trajectory for {}'.format(metr_name))
00123     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00124     gmx_eneconv(f=[os.path.join("./past", hashed_name) + '.edr' for hashed_name in hashed_names[:wave]],
      o='./{}_combined_energy.edr'.format(metr_name))
00125     for i in range(wave, len(hashed_names), wave):
00126         os.rename('./{}_combined_energy.edr'.format(metr_name), './{}_combined_energy_prev.edr'.format(metr_name))
00127         gmx_eneconv(f=["./{}_combined_energy_prev.edr".format(metr_name)] + [os.path.join("./past", hashed_name + '.edr') for hashed_name in
      hashed_names[i:i + wave if i + wave < len(hashed_names) else -1]],
00128                     o='./{}_combined_energy.edr'.format(metr_name))
00129         if int(i / wave) % 10 == 0:
```

```
00130             print('{}/{} ({:.1f}%)'.format(int(i / wave), tot_chunks, 100 * int(i / wave) / tot_chunks))
00131
00132     os.rename('./{}_combined_energy.edr'.format(metr_name), './{}_combined_energy_best.edr'.format(metr_name))
00133
00134
```

References gmx_wrappers.gmx_eneconv(), and gmx_wrappers.gmx_trjcat().
Referenced by main().
Here is the call graph for this function:



Here is the caller graph for this function:



3.15.1.2 main() def make_best_trajectory_new.main ( )

Definition at line 23 of file make_best_trajectory_new.py.

```
00023 def main():
00024     db_to_connect = 'results_opls_trp_300_fixed'
00025     # if len(sys.argv) < 2:
00026     #     raise Exception('Not enough arguments')
00027     # db_to_connect = sys.argv[1]
00028     # try:
00029     #     os.mkdir('best_past')
00030     # except:
00031     #     pass
00032     for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00033         build_best_traj(metr, db_to_connect)
00034     # pool = mp.Pool(len(['rmsd', 'angl', 'andh', 'and', 'xor'])) # we are IO bound in graphs, no need to use exact number of CPUs
       mp.cpu_count()
00035     # results1 = pool.starmap_async(build_best_traj, [(metr, db_to_connect) for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']])
00036     # results1.get()
00037     # pool.close()
00038
00039
00040
```

References build_best_traj().
Here is the call graph for this function:

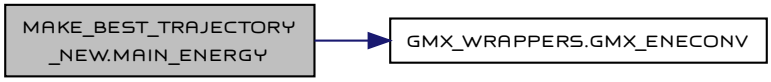
3.15.1.3 main_energy() def make_best_trajectory_new.main_energy ( )

Definition at line 145 of file make_best_trajectory_new.py.

```
00145     """
00146     past_dir = './past'
00147     db_to_connect = 'results_12'
00148     polynomial = False
00149     font = {'family': 'serif',
00150             'color': 'darkred',
00151             'weight': 'normal',
00152             'size': 16,
00153             }
00154     if not os.path.exists(db_to_connect + '.sqlite3'):
00155         raise Exception('DB not found')
00156 
00157     con = lite.connect(db_to_connect + '.sqlite3', check_same_thread=False, isolation_level=None)
00158     cur = con.cursor()
00159 
00160     qry = "select a.name, a.hashed_name from main_storage a  where a.goal_dist= ( select min(b.goal_dist) from main_storage b)"
00161     result = cur.execute(qry)
00162     all_res = result.fetchone()
00163     name = all_res[0]
00164     spname = name.split('_')
00165     all_prev_names = ['\'{}\"".format('_'.join(spname[:i])) for i in range(1, len(spname))]
00166     long_line = ", ".join(all_prev_names)
00167 
00168     qry = "select name, hashed_name from main_storage where name in ({})".format(long_line)
00169     result = cur.execute(qry)
00170     _ = result.fetchone()
00171     all_res = result.fetchall()
00172     names, hashed_names = zip(*all_res)
00173     wave = 100
00174     tot_chunks = int((len(hashed_names) + 1) / wave)
00175     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00176     gmx_eneconv(f=[os.path.join("./past", hashed_name) + '.edr' for hashed_name in hashed_names[:wave]], o='./combined_energy.edr')
00177     for i in range(wave, len(hashed_names) + 1 - wave, wave):
00178         os.rename('./combined_energy.edr', './combined_energy_prev.edr')
00179         gmx_eneconv(f=["./combined_energy_prev.edr"] + [os.path.join("./past", hashed_name + '.edr') for hashed_name in hashed_names[i:i + wave
      if i + wave < len(hashed_names) else -1]],
00180                     o='./combined_energy.edr')
00181         if int(i / wave) % 10 == 0:
00182             print('{}/{} ({:.1f}%)'.format(int(i / wave), tot_chunks, 100 * int(i / wave) / tot_chunks))
00183 
00184     os.rename('./combined_energy.edr', './combined_energy_best.edr')
00185     print('Done with best')
00186 
00187 
00188 
00189     qry = "select a.name, a.hashed_name from main_storage a "
00190     result = cur.execute(qry)
00191     _ = result.fetchone()
00192     all_res = result.fetchall()
00193     names, hashed_names = zip(*all_res)
00194 
00195     # gmx_eneconv(f=[os.path.join(past_dir, hash_name+'.edr') for hash_name in hashed_names], o='./combined_energy.edr')
00196 
00197     wave = 100
00198     tot_chunks = int((len(hashed_names)+1)/wave)
00199     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00200     gmx_eneconv(f=[os.path.join("./past", hashed_name)+'.edr' for hashed_name in hashed_names[:wave]], o='./combined_energy.edr')
00201     for i in range(wave, len(hashed_names)+1-wave, wave):
00202         os.rename('./combined_energy.edr', './combined_energy_prev.edr')
00203         gmx_eneconv(f=["./combined_energy_prev.edr"] +[os.path.join("./past", hashed_name + '.edr') for hashed_name in hashed_names[i:i+wave if
      i+wave < len(hashed_names) else -1]], o='./combined_energy.edr')
00204         if int(i/wave) % 10 == 0:
00205             print('{}/{} ({:.1f}%)'.format(int(i/wave), tot_chunks, 100*int(i/wave)/tot_chunks))
00206 
00207     os.rename('./combined_energy.edr', './combined_energy_all_main.edr')
00208     print('Done with all main')
00209 
00210 
00211     qry = "select a.name, a.hashed_name from main_storage a join log b on a.id=b.id where b.dst='VIZ' order by b.timestamp"
00212     result = cur.execute(qry)
00213     _ = result.fetchone()
00214     all_res = result.fetchall()
00215     names, hashed_names = zip(*all_res)
00216 
00217     wave = 100
00218     tot_chunks = int((len(hashed_names)+1)/wave)
00219     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
```

```
00220     gmx_eneconv(f=[os.path.join("./past", hashed_name)+'.edr' for hashed_name in hashed_names[:wave]], o='./combined_energy.edr')
00221     for i in range(wave, len(hashed_names)+1-wave, wave):
00222         os.rename('./combined_energy.edr', './combined_energy_prev.edr')
00223         gmx_eneconv(f=["./combined_energy_prev.edr"] +[os.path.join("./past", hashed_name + '.edr') for hashed_name in hashed_names[i:i+wave if
      i+wave < len(hashed_names) else -1]], o='./combined_energy.edr')
00224         if int(i/wave) % 10 == 0:
00225             print('{}/{} ({:.1f}%)'.format(int(i/wave), tot_chunks, 100*int(i/wave)/tot_chunks))
00226 
00227     os.rename('./combined_energy.edr', './combined_energy_all_viz.edr')
00228     print('Done with viz')
00229 
00230 
00231     # gmx_energy('./combined_energy.edr', './combined_energy.xvg', fee=True, fetemp=300)
00232 
00233 
```

References gmx_wrappers.gmx_eneconv().
Here is the call graph for this function:



3.16 metric_funcs Namespace Reference

Functions

- list get_knn_dist_mdsctk (str ref_file, str fitfile, str topology)

  'knn_rms' - MDSCTK tool - computes RMSD between two (or more) structures
- np.ndarray get_contat_profile_mdsctk (str ref_file, str fitfile, str index, float dist=2.7)

  'contact_profile' - MDSCTK tool - computes number of contacts between two (or more) structures
- NoReturn get_bb_to_angle_mdsctk (str x='noise_bb.xtc', str o='noise_angle.dat')

  'bb_xtc_to_phipsi' - MDSCTK tool - takes backbone structure and computes dihedral angles between atoms
- NoReturn get_angle_to_sincos_mdsctk (str i='noise_angle.dat', str o='noise_sincos.dat')

  'angles_to_sincos' - MDSCTK tool - converts dihedrals into sin/cos values
- str gen_file_for_amb_noize (str work_dir, int seeds, dict seed_dirs, str ndx_file, str top_file, str goal_file='folded_for_noise.gro', list hostnames=None, list cpu_map=None)

  Performs simulation of the NMR (not unfolded) conformation to measure ambient vibrations.
- np.ndarray compute_phipsi_angles (int angl_num, str target_filename, str ndx, str stor_name=None)

  Top level function that outputs sin/cos of the dihedral angles of the provided conformation.
- np.ndarray ang_dist (list target_ang, list goal_ang)

  Computes difference between two angle lists.
- NoReturn save_an_file (str an_file_name, dict tol_error, list metr_order)

  Writes noise values into the specified file for future use during the restarts.
- tuple get_native_contacts (str goal_prot_only, list files_to_check, str ndx_file, np.ndarray cont_corr, int atom_num, float dist=2.↩
  7, np.ufunc logic_fun=np.logical_xor, list h_filter=None, mp.Pool pool=None, bool just_contacts=False)

  Computes number of contacts between the goal_prot_only and files_to_check.
- NoReturn and_h (mp.Queue q, np.int goal_contacts_and_h_sum, list goal_cont_h, list contacts_h, list prev_contacts_h, np.int and_h_dist_↩
  tot)

  Separate AND_H computation, used to be executed in parallel,.
- NoReturn and_p (mp.Queue q, np.int goal_contacts_and_sum, list goal_contacts, list contacts, list prev_contacts, np.int prev_tot_dist)

  Separate AND computation, used to be executed in parallel,.
- NoReturn rmsd (mp.Queue q, str combined_pg, str temp_xtc_file, str goal_prot_only, np.float64 prev_tot_dist)

  Separate RMSD computation, used to be executed in parallel,.
- NoReturn angl (mp.Queue q, int angl_num, str temp_xtc_file, str init_bb_ndx, list pangl, list goal_angles, np.float64 prev_tot_dist)

  Separate ANGL computation, used to be executed in parallel,.

- list compute_metric (str past_dir, list new_nodes_names, int tot_seeds, str combined_pg, str temp_xtc_file, str goal_prot_only, dict node_info, int angl_num, str init_bb_ndx, list goal_angles, str init_prot_only, list files_for_trjcat, str ndx_file_init, list goal↩ _cont_h, int atom_num, float cont_dist, list h_filter_init, list goal_contacts, int cur_metric, np.int goal_contacts_and_h_sum, np.int goal_contacts_and_sum, bool chance_to_reuse=False, mp.Pool cpu_pool=None, bool compute_all_at_once=True)

  Computes metric distances from the previous node and to the goal (NMR) conformation.
- list compute_init_metric (str past_dir, int tot_seeds, str init_xtc, str goal_xtc, str goal_prot_only, int angl_num, str init_bb_ndx, np.ndarray goal_angles, str init_prot_only, str ndx_file_init, np.ndarray goal_cont_h, int atom_num, float cont_dist, np.ndarray h_↩ filter_init, np.ndarray goal_contacts, np.int 64 goal_contacts_and_h_sum, np.int 64 goal_contacts_and_sum)

  Special case of the "compute_metric".
- str select_metrics_by_snr (list cur_nodes, dict prev_node, list metric_names, dict tol_error, bool compute_all_at_once, list alowed_↩ metrics, str cur_metr)

  SNR approach to a metric selection.

3.16.1 Function Documentation

3.16.1.1 and_h()

```
NoReturn metric_funcs.and_h (
        mp.Queue q,
        np.int goal_contacts_and_h_sum,
        list goal_cont_h,
        list contacts_h,
        list prev_contacts_h,
        np.int and_h_dist_tot )
```

Separate AND_H computation, used to be executed in parallel,.
NOT used anymore since does not result in any significant speed up, but left here "just in case".

mp.Queue q: queue used to communicate with the parent process
np.int goal_contacts_and_h_sum: exact number of NMR contacts
list goal_cont_h: correct (NMR) contacts
list contacts_h: current nodes' contacts
list prev_contacts_h: previous node contacts
np.int and_h_dist_tot: distance accumulated from the origin

Returns

:return: Returns by putting into the queue (metric to goal, metric from previous, total traveled in metric units).

Definition at line 452 of file metric_funcs.py.

```
00452     """
00453     goal_cont_dist_and_h = goal_contacts_and_h_sum - [np.logical_and(arr_elem, goal_cont_h).sum() for arr_elem in contacts_h]
00454     prev_cont_dist_and_h_1 = [np.logical_xor(arr_elem, prev_contacts_h).sum() for arr_elem in contacts_h]
00455     prev_cont_dist_and_h_2 = [arr_elem.sum() for arr_elem in contacts_h] + prev_contacts_h.sum()
00456     prev_cont_dist_and_h_2 = prev_cont_dist_and_h_2 / 2 - \
00457         [elem.sum() for elem in [np.logical_and(arr_elem, prev_contacts_h) for arr_elem in contacts_h]]
00458     total_cont_dist_and_h = and_h_dist_tot + prev_cont_dist_and_h_1
00459     q.put((goal_cont_dist_and_h, prev_cont_dist_and_h_2, total_cont_dist_and_h))
00460
00461
```

Separate AND computation, used to be executed in parallel,.
NOT used anymore since does not result in any significant speed up, but left here "just in case".

mp.Queue q: queue used to communicate with the parent process
np.int goal_contacts_and_sum: exact number of NMR contacts
list goal_contacts: correct (NMR) contacts
list contacts: current nodes' contacts
list prev_contacts: previous node contacts
np.int prev_tot_dist: distance accumulated from the origin

Returns

:return: Returns by putting into the queue (metric to goal, metric from previous, total traveled in metric units).

Definition at line 477 of file metric_funcs.py.

```
00477     """
00478     goal_cont_dist_and = goal_contacts_and_sum - [np.logical_and(arr_elem, goal_contacts).sum() for arr_elem in contacts]
00479     prev_cont_dist_and_1 = [np.logical_xor(arr_elem, prev_contacts).sum() for arr_elem in contacts]
00480     prev_cont_dist_and_2 = [arr_elem.sum() for arr_elem in contacts] + prev_contacts.sum()
00481     prev_cont_dist_and_2 = prev_cont_dist_and_2 / 2 - \
00482         [elem.sum() for elem in [np.logical_and(arr_elem, prev_contacts) for arr_elem in contacts]]
00483     total_cont_dist_and = prev_tot_dist + prev_cont_dist_and_1
00484     q.put((goal_cont_dist_and, prev_cont_dist_and_2, total_cont_dist_and))
00485
00486
```

Computes difference between two angle lists.

list target_ang: angles to test
list goal_ang: goal angles

Returns

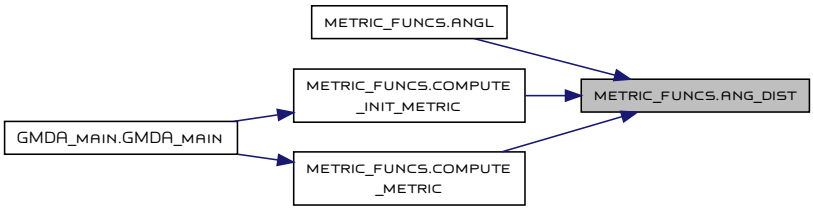
:return: one number when input is a list or list of sums in case intput is list of lists :rtype: np.ndarray

Definition at line 326 of file metric_funcs.py.

```
00326     """
00327     if target_ang.shape[0] == 1 or target_ang.ndim == 1:
00328         return np.abs(target_ang - goal_ang).sum()
00329     else:
00330         return [np.abs(target_ang[i] - goal_ang).sum() for i in range(target_ang.shape[0])]
00331
00332
00333 # def get_ambient_noise_contacts_xor(goal_prot_only, noize_xtc, ndx_file_cont, atom_num, logic_fun,
00334 # corr_contacts, cont_dist, prev_cont, mult=0.8):
00335 #     cont_sum, nat_contacts = get_native_contacts(goal_prot_only, [noize_xtc], ndx_file_cont,
00336 # corr_contacts, atom_num, dist=cont_dist, logic_fun=logic_fun)
00337 #     return max(1,int(min(abs(prev_cont - cont_sum))*mult))
00338
00339 # def get_ambient_noise_contacts(goal_prot_only, noize_xtc, ndx_file_cont, atom_num, logic_fun,
00340 # corr_contacts, cont_dist, prev_cont, mult=0.8):
00341 #     cont_sum, nat_contacts = get_native_contacts(goal_prot_only, [noize_xtc], ndx_file_cont,
00342 # corr_contacts, atom_num, dist=cont_dist, logic_fun=logic_fun)
00343 #     return max(1, int(min(abs(prev_cont - cont_sum)) * mult))
00344
00345
```

Referenced by angl(), compute_init_metric(), and compute_metric().

Here is the caller graph for this function:



3.16.1.2 angl()

```
NoReturn metric_funcs.angl (
                mp.Queue q,
                int angl_num,
                str temp_xtc_file,
                str init_bb_ndx,
                list pangl,
                list goal_angles,
                np.float64 prev_tot_dist )
```

Separate ANGL computation, used to be executed in parallel,.
NOT used anymore since does not result in any significant speed up, but left here "just in case".

mp.Queue q: queue used to communicate with the parent process
int angl_num: total number of angles in the protein
str temp_xtc_file: new frames (same as number of seeds) you want to measure distance from previous and to the goal
str init_bb_ndx: .ndx to extract the backbone atoms
list pangl: previous node angles
list goal_angles: correct angles (NMR angles)
np.float64 prev_tot_dist: distance accumulated from the origin

Returns

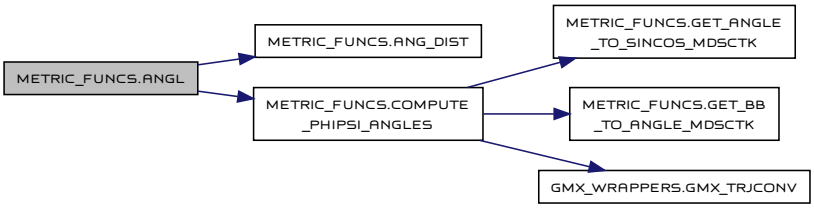
:return: Returns by putting into the queue (metric to goal, metric from previous, total traveled in metric units).

Definition at line 525 of file metric_funcs.py.

```
00525     """
00526     cur_angles = compute_phipsi_angles(angl_num, temp_xtc_file.split('.')[0], init_bb_ndx)
```

```
00527     angl_sum_from_prev = ang_dist(cur_angles, pangl)
00528     angl_sum_to_goal = ang_dist(cur_angles, goal_angles)
00529     angl_sum_tot = prev_tot_dist + angl_sum_from_prev
00530     q.put((angl_sum_to_goal, angl_sum_from_prev, angl_sum_tot, cur_angles))
00531
00532
```

References ang_dist(), and compute_phipsi_angles().
Here is the call graph for this function:



3.16.1.3 compute_init_metric()

```
list metric_funcs.compute_init_metric (
                str  past_dir,
                int  tot_seeds,
                str  init_xtc,
                str  goal_xtc,
                str  goal_prot_only,
                int  angl_num,
                str  init_bb_ndx,
                np.ndarray  goal_angles,
                str  init_prot_only,
                str  ndx_file_init,
                np.ndarray  goal_cont_h,
                int  atom_num,
                float  cont_dist,
                np.ndarray  h_filter_init,
                np.ndarray  goal_contacts,
                np.int 64  goal_contacts_and_h_sum,
                np.int 64  goal_contacts_and_sum )
```

Special case of the "compute_metric".
Computes metric distances to the goal (NMR) conformation and sets previous distances to 0

```
    str  past_dir: path to the directory with prior computation results
    int  tot_seeds: total number of seed in the current run
    str  init_xtc: initial (unfolded) conformation with water and salt
    str  goal_xtc: NMR (folded) conformation with water and salt
    str  goal_prot_only: NMR (folded) conformation without water and salt (protein only)
    int  angl_num: number of dihedral angles in the protein
    str  init_bb_ndx: index file with backbone atom positions for the initial conformation
    np.ndarray    goal_angles: angle values of the NMR structure
    str  init_prot_only: initial (unfolded) conformation without water and salt (protein only)
    str  ndx_file_init: index file with backbone atom positions for the NMR conformation
    np.ndarray    goal_cont_h: contact values of the NMR structure (hydrogens only)
    int  atom_num: total number of atoms in the protein (same for folded and unfolded)
    float   cont_dist: distance between atoms treated as 'contact'
    np.ndarray    h_filter_init: positions of the hydrogen atoms in the initial (unfolded) conformation
    np.ndarray    goal_contacts:   list  of correct contacts in the NMR (folded) conformation
    np.int   64    goal_contacts_and_h_sum: total sum of the contacts between hydrogents in the NMR (folded) conformation
    np.int   64    goal_contacts_and_sum: total sum of the contacts in the NMR (folded) conformation
```

Returns

```
     :return: node structure with the initial metrics :rtype: list
```

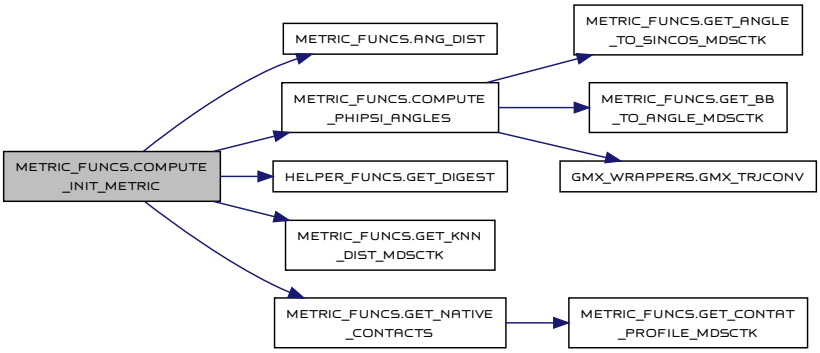
Definition at line 857 of file metric_funcs.py.

```
00857     Returns:
00858         :return: node structure with the initial metrics
00859         return type: list
```

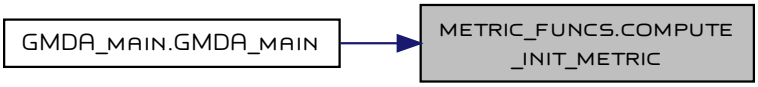
```
00860     """
00861     init_node = [None] * tot_seeds
00862     dim = 1 if tot_seeds > 1 else 0
00863     # *********  RMSD ************
00864     rmsd_to_goal = get_knn_dist_mdsctk(init_xtc, goal_xtc, goal_prot_only)
00865     # *********  ANG ************
00866     cur_angles = compute_phipsi_angles(angl_num, init_xtc.split('.')[0], init_bb_ndx)
00867 
00868     angl_sum_to_goal = ang_dist(cur_angles, goal_angles)
00869 
00870     contacts = get_native_contacts(init_prot_only, [init_xtc], ndx_file_init, None, atom_num, cont_dist, None, just_contacts=True)[1]
00871     # print(init_prot_only, init_xtc, ndx_file_init, atom_num, cont_dist)
00872     #  Cont prep
00873     contacts_h = np.logical_and(contacts, h_filter_init)
00874     # *********  AND_H ************
00875     goal_cont_dist_and_h = goal_contacts_and_h_sum - np.logical_and(contacts_h, goal_cont_h).sum(axis=dim)
00876     # *********  AND ************
00877     goal_cont_dist_and = goal_contacts_and_sum - np.logical_and(contacts, goal_contacts).sum(axis=dim)
00878     # *********  XOR ************
00879     goal_cont_dist_sum_xor = np.logical_xor(contacts, goal_contacts).sum(axis=dim)
00880 
00881     if dim == 0:
00882         contacts = [contacts]
00883         # contacts_h = [contacts_h]
00884         angl_sum_to_goal = [angl_sum_to_goal]
00885         goal_cont_dist_and_h = [goal_cont_dist_and_h]
00886         goal_cont_dist_and = [goal_cont_dist_and]
00887         goal_cont_dist_sum_xor = [goal_cont_dist_sum_xor]
00888 
00889     # store all metrics
00890     for i in range(tot_seeds):
00891         init_node[i] = dict ()
00892         init_node[i]['digest_name'] = get_digest('s')
00893 
00894         init_node[i]['RMSD_to_goal'] = np.float32(rmsd_to_goal[i])
00895         init_node[i]['RMSD_from_prev'] = np.uint32(0)
00896         init_node[i]['RMSD_dist_total'] = np.uint32(0)
00897 
00898         init_node[i]['ANGL_to_goal'] = np.float32(angl_sum_to_goal[i])
00899         init_node[i]['ANGL_from_prev'] = np.uint32(0)
00900         init_node[i]['ANGL_dist_total'] = np.uint32(0)
00901 
00902         init_node[i]['AND_H_to_goal'] = np.uint32(goal_cont_dist_and_h[i])
00903         init_node[i]['AND_H_from_prev'] = np.uint32(0)
00904         init_node[i]['AND_H_dist_total'] = np.uint32(0)
00905 
00906         init_node[i]['AND_to_goal'] = np.uint32(goal_cont_dist_and[i])
00907         init_node[i]['AND_from_prev'] = np.uint32(0)
00908         init_node[i]['AND_dist_total'] = np.uint32(0)
00909 
00910         init_node[i]['XOR_to_goal'] = np.uint32(goal_cont_dist_sum_xor[i])
00911         init_node[i]['XOR_from_prev'] = np.uint32(0)
00912         init_node[i]['XOR_dist_total'] = np.uint32(0)
00913         # init_node[i]['contacts'] = csc_matrix(contacts[i])
00914         save_npz(os.path.join(past_dir, '{}.cont'.format(init_node[i]['digest_name'])),
00915                  csc_matrix(contacts[i]), compressed=True)
00916 
00917         init_node[i]['native_name'] = zlib.compress('s'.encode(), 9)
00918 
00919         # init_node[i]['angles'] = cur_angles[i]
00920         cur_angles.astype('float32').tofile(os.path.join(past_dir, '{}.angl'.format(init_node[i]['digest_name'])))
00921 
00922     if len(init_node) == 1:
00923         return init_node[0]
00924     return init_node
00925 
00926 
```

References ang_dist(), compute_phipsi_angles(), helper_funcs.get_digest(), get_knn_dist_mdsctk(), and get_native_contacts().
Referenced by GMDA_main.GMDA_main().

Here is the call graph for this function:



Here is the caller graph for this function:



3.16.1.4 compute_metric()

```
list metric_funcs.compute_metric (
                str  past_dir,
                list  new_nodes_names,
                int  tot_seeds,
                str  combined_pg,
                str  temp_xtc_file,
                str  goal_prot_only,
                dict  node_info,
                int  angl_num,
                str  init_bb_ndx,
                list  goal_angles,
                str  init_prot_only,
                list  files_for_trjcat,
                str  ndx_file_init,
                list  goal_cont_h,
                int  atom_num,
                float  cont_dist,
                list  h_filter_init,
                list  goal_contacts,
                int  cur_metric,
                np.int  goal_contacts_and_h_sum,
                np.int  goal_contacts_and_sum,
                bool  chance_to_reuse =  False,
                mp.Pool   cpu_pool =  None,
                bool  compute_all_at_once =  True )
```

Computes metric distances from the previous node and to the goal (NMR) conformation.
Before I was computing metrics separately, but computing them all at once add very little overhead and allows to track trajectory behavior, so later I fixed only the code with all at once option.

```
    str  past_dir: path to the directory with prior computation results
    list  new_nodes_names: full names of newly computed nodes (not current)
    int  tot_seeds: total number of seed in the current run
    str  combined_pg: previous and goal frames combined into one trajectory
```

```
    str  temp_xtc_file: new nodes' final frames
    str  goal_prot_only: NMR (folded) conformation without water and salt (protein only)
    dict   node_info: info about the current node (not just computed, but rather previous)
    int  angl_num: number of dihedral angles in the protein
    str  init_bb_ndx: index file with backbone atom positions for the initial conformation
    list  goal_angles: angle values of the NMR structure
    str  init_prot_only: initial (unfolded) conformation without water and salt (protein only)
    list  files_for_trjcat:   list  of newly computed nodes (files, with hash as a name)
    str  ndx_file_init: index file with backbone atom positions for the NMR conformation
    list  goal_cont_h: contact values of the NMR structure (hydrogens only)
    int  atom_num: total number of atoms in the protein (same for folded and unfolded)
    float   cont_dist: distance between atoms treated as 'contact'
    list  h_filter_init: positions of the hydrogen atoms in the initial (unfolded) conformation
    list  goal_contacts:   list  of correct contacts in the NMR (folded) conformation
    int  cur_metric: metric index
    np.int    goal_contacts_and_h_sum: total sum of the contacts between hydrogents in the NMR (folded) conformation
    np.int    goal_contacts_and_sum: total sum of the contacts in the NMR (folded) conformation
    bool   chance_to_reuse:
    mp.Pool cpu_pool: CPU pool for local parallel processing
    bool   compute_all_at_once: toggle whether to compute all metrics at the same time or not (yes, if no check the code)
```

Returns

:return: new nodes with all metrics (compute_all_at_once only) and current metric distances :rtype: list

Definition at line 568 of file metric_funcs.py.

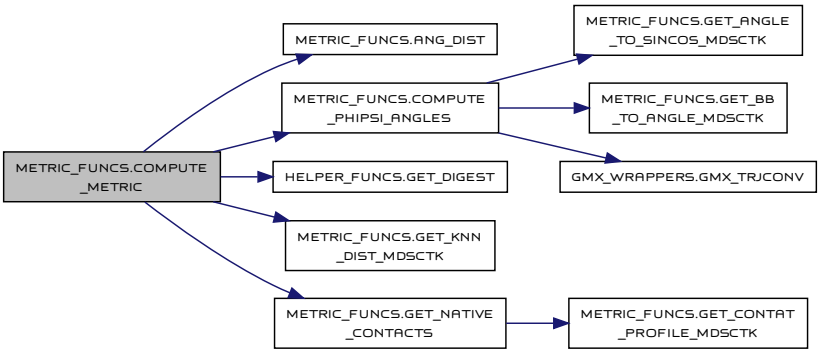
```
00568     Returns:
00569         :return: new nodes with all metrics (compute_all_at_once only) and current metric distances
00570         return type: list
00571     """
00572     # global extra_past
00573     new_nodes = [None] * tot_seeds
00574     # prev_contacts = node_info['contacts']
00575     try:
00576         prev_contacts = load_npz(os.path.join(past_dir, '{}.cont.npz'.format(node_info['digest_name']))).toarray()
00577     except:
00578         print('Previous contact do not exists. Probably error in the previous step.\nFile: ',
00579               os.path.join(past_dir, '{}.cont.npz'.format(node_info['digest_name'])),
00580               ' was not found')
00581         exit(-10)
00582         # prev_contacts = load_npz(os.path.join(extra_past, '{}.cont.npz'.format(node_info['digest_name']))).toarray()
00583     digests = [get_digest(new_nodes_names[i]) for i in range(tot_seeds)]
00584     if compute_all_at_once:
00585         # Parallel approach does not work on small/medium proteins. Overhead of proc creation is more than time to compute.
00586         # However, when you decide to speed up execution, make only angl dist to be computed in sep process.
00587         # q = mp.Queue()
00588         # pid = multiprocessing.Process(target=angl, args=(q, angl_num, temp_xtc_file, init_bb_ndx, node_info['angles'],
00589         # goal_angles, node_info['ANGL_dist_total']))
00590         # pid.start()
00591 
00592         # *********  PREP ************
00593         reusing_old_cont = False
00594         # if chance_to_reuse:
00595         try:  # lets always check for previous files and regenerate them in case of the error - incomplete or do not exist
00596             contacts = [load_npz(os.path.join(past_dir, '{}.cont.npz'.format(digests[i]))).toarray() for i in range(tot_seeds)]
00597             reusing_old_cont = True
00598         except OSError:
00599             contacts = get_native_contacts(init_prot_only, files_for_trjcat, ndx_file_init, None,
00600                                            atom_num, cont_dist, None, pool=cpu_pool, just_contacts=True)[1]
00601         # else:
00602         #     contacts = get_native_contacts(init_prot_only, files_for_trjcat, ndx_file_init, None,
00603         #                                    atom_num, cont_dist, None, pool=cpu_pool, just_contacts=True)[1]
00604 
00605         # print(init_prot_only, files_for_trjcat, ndx_file_init, atom_num, cont_dist)
00606         #  Cont prep
00607         contacts_h = [np.logical_and(arr_elem, h_filter_init) for arr_elem in contacts]
00608         prev_contacts_h = np.logical_and(prev_contacts, h_filter_init)
00609 
00610         # ************** PAR ************
00611         # q = [mp.Queue() for i in range(4)]
00612         # bad approach
00613         # par_metr = [multiprocessing.Process(target=and_h, args=(q[0], goal_contacts_and_h_sum, goal_cont_h, contacts_h,
00614         # prev_contacts_h, node_info['AND_H_dist_total'])),
00615         #             multiprocessing.Process(target=and_p, args=(q[1], goal_contacts_and_sum, goal_contacts, contacts,
00616         # prev_contacts, node_info['AND_dist_total'])),
00617         #             multiprocessing.Process(target=rmsd, args=(q[2], combined_pg, temp_xtc_file,
00618         # goal_prot_only, node_info['RMSD_dist_total'])),
```

```
00619         #             multiprocessing.Process(target=angl, args=(q[3], angl_num, temp_xtc_file, init_bb_ndx,
00620         # node_info['angles'], goal_angles, node_info['ANGL_dist_total']))]
00621         # [pid.start() for pid in par_metr]
00622         # [pid.join() for pid in par_metr]
00623         # goal_cont_dist_and_h, prev_cont_dist_and_h_2, total_cont_dist_and_h = q[0].get()
00624         # goal_cont_dist_and, prev_cont_dist_and_2, total_cont_dist_and = q[1].get()
00625         # rmsd_to_goal, from_prev_dist, rmsd_total_trav = q[2].get()
00626         # angl_sum_to_goal, angl_sum_from_prev, angl_sum_tot, cur_angles = q[3].get()
00627         #
00628         # better approach
00629         # q = [mp.Queue() for i in range(4)]
00630         # pid = multiprocessing.Process(target=angl, args=(q[3], angl_num, temp_xtc_file, init_bb_ndx, node_info['angles'],
00631         # goal_angles, node_info['ANGL_dist_total']))
00632         # pid.start()
00633         # and_h(q[0], goal_contacts_and_h_sum, goal_cont_h, contacts_h, prev_contacts_h, node_info['AND_H_dist_total'])
00634         # and_p(q[1], goal_contacts_and_sum, goal_contacts, contacts, prev_contacts, node_info['AND_dist_total'])
00635         # rmsd(q[2], combined_pg, temp_xtc_file, goal_prot_only, node_info['RMSD_dist_total'])
00636         # pid.join()
00637         # angl_sum_to_goal, angl_sum_from_prev, angl_sum_tot, cur_angles = q[3].get()
00638
00639         # *********  RMSD ************
00640         dist_arr = get_knn_dist_mdsctk(combined_pg, temp_xtc_file, goal_prot_only)
00641         from_prev_dist = dist_arr[0::2]
00642         rmsd_to_goal = dist_arr[1::2]
00643         rmsd_total_trav = [node_info['RMSD_dist_total'] + elem for elem in from_prev_dist]
00644
00645         # *********  ANG ************
00646         reusing_old_angl = False
00647         # if chance_to_reuse:
00648         try:
00649             cur_angles = [np.fromfile(os.path.join(past_dir, '{}.angl'.format(digests[i])), dtype=np.float32) for i in range(tot_seeds)]
00650             cur_angles = np.asarray(cur_angles, dtype=np.float32)
00651             reusing_old_angl = True
00652         except OSError:
00653             cur_angles = compute_phipsi_angles(angl_num, temp_xtc_file.split('.')[0], init_bb_ndx)
00654         # else:
00655         #     cur_angles = compute_phipsi_angles(angl_num, temp_xtc_file.split('.')[0], init_bb_ndx)
00656
00657         # angl_sum_from_prev = ang_dist(cur_angles, node_info['angles'])
00658         # if os.path.exists(os.path.join(past_dir, '{}.angl'.format(node_info['digest_name']))):
00659         try:
00660             angl_sum_from_prev = ang_dist(cur_angles, np.fromfile(os.path.join(past_dir, '{}.angl'.format(node_info['digest_name'])),
      dtype=np.float32))
00661         except Exception as e:
00662             print('Error during previous angle read.\nCheck ', os.path.join(past_dir, '{}.angl'.format(node_info['digest_name'])), 'Error: ',
      e)
00663             exit(-10)
00664         # else:
00665             # angl_sum_from_prev = ang_dist(cur_angles, np.fromfile(os.path.join(extra_past, '{}.angl'.format(node_info['digest_name'])),
      dtype=np.float32))
00666         angl_sum_to_goal = ang_dist(cur_angles, goal_angles)
00667         angl_sum_tot = node_info['ANGL_dist_total'] + angl_sum_from_prev
00668
00669         # *********  AND_H ************
00670         goal_cont_dist_and_h = goal_contacts_and_h_sum - [np.logical_and(arr_elem, goal_cont_h).sum() for arr_elem in contacts_h]
00671         prev_cont_dist_and_h_1 = [np.logical_xor(arr_elem, prev_contacts_h).sum() for arr_elem in contacts_h]
00672         # prev_cont_dist_and_h_2 = [arr_elem.sum() for arr_elem in contacts_h] + prev_contacts_h.sum()
00673         # prev_cont_dist_and_h_2 = prev_cont_dist_and_h_2 / 2 - \
00674         #    [elem.sum() for elem in [np.logical_and(arr_elem, prev_contacts_h) for arr_elem in contacts_h]]
00675         total_cont_dist_and_h = node_info['AND_H_dist_total'] + prev_cont_dist_and_h_1
00676
00677         # *********  AND ************
00678         goal_cont_dist_and = goal_contacts_and_sum - [np.logical_and(arr_elem, goal_contacts).sum() for arr_elem in contacts]
00679         prev_cont_dist_and_1 = [np.logical_xor(arr_elem, prev_contacts).sum() for arr_elem in contacts]
00680         # prev_cont_dist_and_2 = [arr_elem.sum() for arr_elem in contacts] + prev_contacts.sum()
00681         # prev_cont_dist_and_2 = prev_cont_dist_and_2 / 2 - \
00682         #                     [elem.sum() for elem in [np.logical_and(arr_elem, prev_contacts) for arr_elem in contacts]]
00683         total_cont_dist_and = node_info['AND_dist_total'] + prev_cont_dist_and_1
00684
00685         # *********  XOR ************
00686         goal_cont_dist_sum_xor = [np.logical_xor(arr_elem, goal_contacts).sum() for arr_elem in contacts]
00687         # prev_cont_dist_sum_xor = [np.logical_xor(arr_elem, prev_contacts).sum() for arr_elem in contacts]
00688         prev_cont_dist_sum_xor = prev_cont_dist_and_1  # it is the same, no need to compute twice
00689         total_cont_dist_xor = node_info['XOR_dist_total'] + prev_cont_dist_sum_xor
00690
00691         # # END PAR
00692         # pid.join()
00693         # angl_sum_to_goal, angl_sum_from_prev, angl_sum_tot, cur_angles = q.get()
00694
00695         # store all metrics
00696         for i in range(tot_seeds):
```

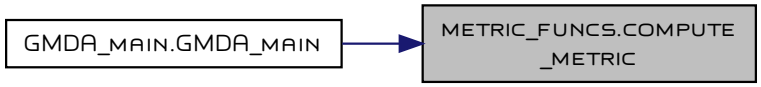
```
00697             new_nodes[i] = dict ()
00698             new_nodes[i]['digest_name'] = get_digest(new_nodes_names[i])
00699 
00700             new_nodes[i]['RMSD_to_goal'] = np.float32(rmsd_to_goal[i])
00701             new_nodes[i]['RMSD_from_prev'] = np.float32(from_prev_dist[i])
00702             new_nodes[i]['RMSD_dist_total'] = np.float32(rmsd_total_trav[i])
00703 
00704             new_nodes[i]['ANGL_to_goal'] = np.float32(angl_sum_to_goal[i])
00705             new_nodes[i]['ANGL_from_prev'] = np.float32(angl_sum_from_prev[i])
00706             new_nodes[i]['ANGL_dist_total'] = np.float32(angl_sum_tot[i])
00707 
00708             new_nodes[i]['AND_H_to_goal'] = np.int 32(goal_cont_dist_and_h[i])
00709             new_nodes[i]['AND_H_from_prev'] = np.int 32(prev_cont_dist_and_h_1[i])
00710             new_nodes[i]['AND_H_dist_total'] = np.int 32(total_cont_dist_and_h[i])
00711 
00712             new_nodes[i]['AND_to_goal'] = np.int 32(goal_cont_dist_and[i])
00713             new_nodes[i]['AND_from_prev'] = np.int 32(prev_cont_dist_and_1[i])
00714             new_nodes[i]['AND_dist_total'] = np.int 32(total_cont_dist_and[i])
00715 
00716             new_nodes[i]['XOR_to_goal'] = np.int 32(goal_cont_dist_sum_xor[i])
00717             new_nodes[i]['XOR_from_prev'] = np.int 32(prev_cont_dist_sum_xor[i])
00718             new_nodes[i]['XOR_dist_total'] = np.int 32(total_cont_dist_xor[i])
00719 
00720             new_nodes[i]['native_name'] = zlib.compress(new_nodes_names[i].encode(), 9)
00721             # new_nodes[i]['contacts'] = csc_matrix(contacts[i])  # csc is the most efficient for contacts data, I tested it.
00722             # new_nodes[i]['angles']   = cur_angles[i].astype('float32')
00723 
00724             if not reusing_old_cont:
00725                 save_npz((os.path.join(past_dir, '{}.cont'.format(new_nodes[i]['digest_name']))), csc_matrix(contacts[i]), compressed=True)
00726 
00727             if not reusing_old_angl:
00728                 cur_angles[i].astype('float32').tofile(os.path.join(past_dir, '{}.angl'.format(new_nodes[i]['digest_name'])))
00729 
00730         if cur_metric == 0:
00731             return new_nodes, rmsd_to_goal, from_prev_dist, rmsd_total_trav
00732         elif cur_metric == 1:
00733             return new_nodes, angl_sum_to_goal, angl_sum_from_prev, angl_sum_tot
00734         elif cur_metric == 2:
00735             # if not isinstance(goal_cont_dist_and_h, (list,)):
00736             #     raise Exception('AND_H_to_goal: ', goal_cont_dist_and_h)
00737             return new_nodes, list(goal_cont_dist_and_h), list(prev_cont_dist_and_h_1), list(total_cont_dist_and_h)
00738         elif cur_metric == 3:
00739             # if not isinstance(goal_cont_dist_and, (list,)):
00740             #     raise Exception('AND_to_goal: ', goal_cont_dist_and)
00741             return new_nodes, list(goal_cont_dist_and), list(prev_cont_dist_and_1), list(total_cont_dist_and)
00742         elif cur_metric == 4:
00743             # if not isinstance(goal_cont_dist_sum_xor, (list,)):
00744             #     raise Exception('XOR_to_goal: ', goal_cont_dist_sum_xor)
00745             return new_nodes, list(goal_cont_dist_sum_xor), list(prev_cont_dist_sum_xor), list(total_cont_dist_xor)
00746         else:
00747             raise Exception('Unknown metric')
00748     else:  # This version is outdated. Using one metric does not produce significant speedup
00749         if cur_metric == 0:  # RMSD
00750             dist_arr = get_knn_dist_mdsctk(combined_pg, temp_xtc_file, goal_prot_only)
00751             # TODO: fix rm files and check if other files has to be removed
00752             # rm_queue.put_nowait(combined_pg)
00753             # rm_queue.put_nowait(temp_xtc_file)
00754             # since combined_pg had two points we have to divide result into two arrays
00755             from_prev_dist = dist_arr[0::2]
00756             rmsd_to_goal = dist_arr[1::2]
00757             rmsd_total_trav = [node_info['RMSD_dist_total'] + elem for elem in from_prev_dist]
00758             for i in range(tot_seeds):
00759                 new_nodes[i]['RMSD_to_goal'] = rmsd_to_goal[i]
00760                 new_nodes[i]['RMSD_from_prev'] = from_prev_dist[i]
00761                 new_nodes[i]['RMSD_dist_total'] = rmsd_total_trav[i]
00762 
00763             return new_nodes, rmsd_to_goal, from_prev_dist, rmsd_total_trav
00764 
00765         elif cur_metric == 1:  # PhyPsi
00766             cur_angles = compute_phipsi_angles(angl_num, temp_xtc_file.split('.')[0], init_bb_ndx)
00767             angl_sum_from_prev = ang_dist(cur_angles, node_info['angles'])
00768             angl_sum_to_goal = ang_dist(cur_angles, goal_angles)
00769             angl_sum_tot = node_info['ANG_dist_total'] + angl_sum_from_prev
00770             for i in range(tot_seeds):
00771                 new_nodes[i]['ANGL_to_goal'] = angl_sum_to_goal[i]
00772                 new_nodes[i]['ANGL_from_prev'] = angl_sum_from_prev[i]
00773                 new_nodes[i]['ANGL_dist_total'] = angl_sum_tot[i]
00774                 new_nodes[i]['angles'] = cur_angles[i]
00775 
00776             return new_nodes, angl_sum_to_goal, angl_sum_from_prev, angl_sum_tot
00777 
```

```
00778         elif cur_metric == 2:  # AND_H
00779             contacts = get_native_contacts(init_prot_only, files_for_trjcat, ndx_file_init, goal_contacts,
00780                                            atom_num, cont_dist, np.logical_and, pool=cpu_pool)[1]
00781             # although it is possible to get h_contacts from the get_native_contacts, then I'll not be able to get pure contacts to store
00782             contacts_h = [np.logical_and(arr_elem, h_filter_init) for arr_elem in contacts]
00783             goal_cont_dist_and_h = [np.logical_and(arr_elem, goal_cont_h).sum() for arr_elem in contacts_h]
00784             prev_contacts_h = np.logical_and(prev_contacts.toarray(), h_filter_init)
00785             prev_cont_dist_and_h_1 = [np.logical_xor(arr_elem, prev_contacts_h).sum() for arr_elem in contacts_h]
00786             prev_cont_dist_and_h_2 = [arr_elem.sum() for arr_elem in contacts_h] + prev_contacts_h.sum()
00787             prev_cont_dist_and_h_2 = prev_cont_dist_and_h_2 / 2 - \
00788                 [elem.sum() for elem in [np.logical_and(arr_elem, prev_contacts_h) for arr_elem in contacts_h]]
00789             total_cont_dist_and_h = node_info['AND_H_dist_total'] + prev_cont_dist_and_h_1
00790             for i in range(tot_seeds):
00791                 new_nodes[i]['AND_H_to_goal'] = goal_cont_dist_and_h[i]
00792                 new_nodes[i]['AND_H_from_prev'] = prev_cont_dist_and_h_1[i]
00793                 new_nodes[i]['AND_H_dist_total'] = total_cont_dist_and_h[i]
00794                 new_nodes[i]['contacts'] = csc_matrix(contacts[i])
00795 
00796             return new_nodes, goal_cont_dist_and_h, prev_cont_dist_and_h_1, total_cont_dist_and_h
00797 
00798         elif cur_metric == 3:  # AND
00799             goal_cont_dist_and, contacts = get_native_contacts(init_prot_only, files_for_trjcat, ndx_file_init, goal_contacts,
00800                                                                atom_num, cont_dist, np.logical_and, pool=cpu_pool)
00801             prev_cont_dist_and_1 = [np.logical_xor(arr_elem, prev_contacts.toarray()).sum() for arr_elem in contacts]
00802             prev_cont_dist_and_2 = [arr_elem.sum() for arr_elem in contacts] + prev_contacts.sum()
00803             prev_cont_dist_and_2 = prev_cont_dist_and_2 / 2 - \
00804                 [elem.sum() for elem in [np.logical_and(arr_elem, prev_contacts.toarray()) for arr_elem in contacts]]
00805             total_cont_dist_and = node_info['AND_dist_total'] + prev_cont_dist_and_1
00806             for i in range(tot_seeds):
00807                 new_nodes[i]['AND_to_goal'] = goal_cont_dist_and[i]
00808                 new_nodes[i]['AND_from_prev'] = prev_cont_dist_and_1[i]
00809                 new_nodes[i]['AND_dist_total'] = total_cont_dist_and[i]
00810                 new_nodes[i]['contacts'] = csc_matrix(contacts[i])
00811 
00812             return new_nodes, goal_cont_dist_and, prev_cont_dist_and_1, total_cont_dist_and
00813 
00814         elif cur_metric == 4:  # XOR
00815             goal_cont_dist_xor, contacts = get_native_contacts(init_prot_only, files_for_trjcat, ndx_file_init, goal_contacts,
00816                                                                atom_num, cont_dist, np.logical_xor, pool=cpu_pool)
00817             prev_cont_dist_sum_xor = [np.logical_xor(arr_elem, prev_contacts.toarray()).sum() for arr_elem in contacts]
00818             total_cont_dist_xor = node_info['XOR_dist_total'] + prev_cont_dist_sum_xor
00819             for i in range(tot_seeds):
00820                 new_nodes[i]['XOR_to_goal'] = goal_cont_dist_xor[i]
00821                 new_nodes[i]['XOR_from_prev'] = prev_cont_dist_sum_xor[i]
00822                 new_nodes[i]['XOR_dist_total'] = total_cont_dist_xor[i]
00823                 new_nodes[i]['contacts'] = csc_matrix(contacts[i])
00824 
00825             return new_nodes, goal_cont_dist_xor, prev_cont_dist_sum_xor, total_cont_dist_xor
00826 
00827     raise Exception("You cant be here")
00828 
00829 
```

References ang_dist(), compute_phipsi_angles(), helper_funcs.get_digest(), get_knn_dist_mdsctk(), and get_native_contacts().
Referenced by GMDA_main.GMDA_main().
Here is the call graph for this function:



Here is the caller graph for this function:



3.16.1.5 compute_phipsi_angles()

```
np.ndarray  metric_funcs.compute_phipsi_angles (
                int  angl_num,
                str  target_filename,
                str  ndx,
                str   stor_name =  None )
```

Top level function that outputs sin/cos of the dihedral angles of the provided conformation.

int angl_num: total number of angles in the protein
str target_filename:
str ndx: index file to extract only specific atoms (extract the backbone)
str stor_name:

Returns

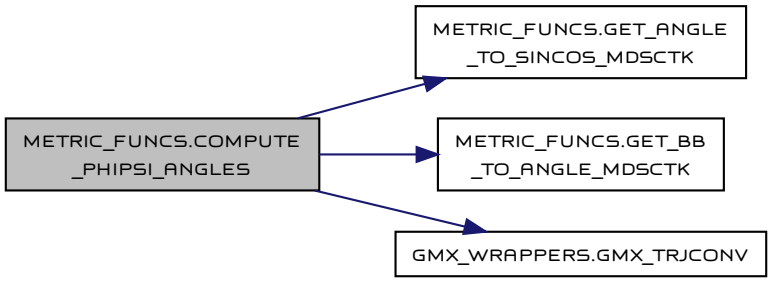
:return: array with sin/cos values of the backbone angles. :rtype: np.ndarray

Definition at line 288 of file metric_funcs.py.

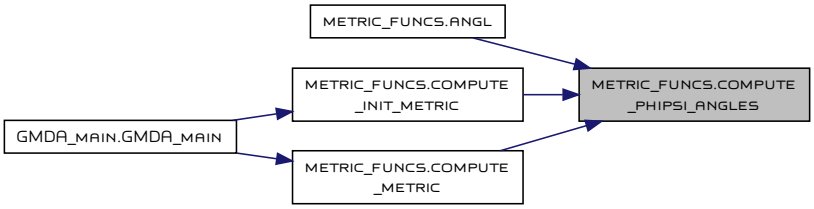
```
00288     """
00289     xtc_filename = "{}.xtc".format(target_filename)
00290     if stor_name is None:  # then create temp file in /dev/shm
00291         bb_filename = "{}_bb.xtc".format(target_filename)
00292         ang_filename = "{}_bb.ang".format(target_filename)
00293         sin_cos_filename = "{}_bb.sc".format(target_filename)
00294         # making sure that we do not reuse old files
00295         if os.path.exists(bb_filename):
00296             os.remove(bb_filename)
00297         if os.path.exists(bb_filename):
00298             os.remove(bb_filename)
00299     else:  # then store in ./past/
00300         bb_filename = "{}_bb.xtc".format(stor_name)
00301         ang_filename = "{}_bb.ang".format(stor_name)
00302         sin_cos_filename = "{}_bb.sc".format(stor_name)
00303
00304     gmx_trjconv(f=xtc_filename, o=bb_filename, n=ndx)
00305     get_bb_to_angle_mdsctk(x=bb_filename, o=ang_filename)
00306     get_angle_to_sincos_mdsctk(i=ang_filename, o=sin_cos_filename)
00307
00308     with open(sin_cos_filename, 'rb') as file:
00309         initial_1d_array = np.frombuffer(file.read(), dtype=np.float64 , count=-1)
00310     check_arr = np.reshape(initial_1d_array, (-1, angl_num * 2))
00311     if len(check_arr) == 1:
00312         return check_arr[0]
00313     return check_arr
00314
00315
```

References get_angle_to_sincos_mdsctk(), get_bb_to_angle_mdsctk(), and gmx_wrappers.gmx_trjconv().
Referenced by angl(), compute_init_metric(), and compute_metric().

Here is the call graph for this function:



Here is the caller graph for this function:



3.16.1.6 gen_file_for_amb_noize()

```
str  metric_funcs.gen_file_for_amb_noize (
                str  work_dir,
                int  seeds,
                dict  seed_dirs,
                str  ndx_file,
                str  top_file,
                str   goal_file =  'folded_for_noise.gro',
                list   hostnames =  None,
                list   cpu_map =  None )
```

Performs simulation of the NMR (not unfolded) conformation to measure ambient vibrations.

```
    str  work_dir: path to the working directory
    int  seeds: number of seed in the current run
    dict   seed_dirs: paths to directories where emulation is performed with particular seed
    str  ndx_file: index file to extract only specific atoms (strip water)
    str  top_file: .top topology file of the simulation box
    str  goal_file: goal (typically NMR) conformation
    list  hostnames: for MPI, to perform parallel computation
    listcpu_map: number of cores for particular task (seed)
```

Returns

```
     :return: filename which contains all seed simulations concatenated :rtype: str
```

Generates a file with trajectories from the goal.

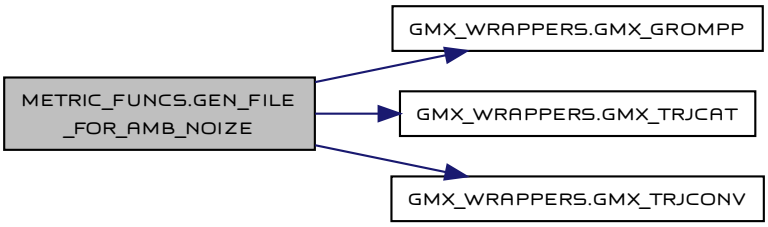
Definition at line 209 of file metric_funcs.py.

```
00209     Generates a file with trajectories from the goal.
00210     """
00211     # if file ambient.rmsd found, read it
```

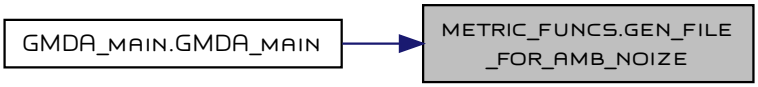
```
00212
00213     temp_xtc_file = 'noise.xtc'
00214     #  generate and save if not found
00215     if temp_xtc_file not in os.walk(".").__next__()[2]:
00216         pid_arr = list()
00217         for i, seed in enumerate(seeds):
00218
00219             gmx_grompp(work_dir, seed, top_file,
00220                        goal_file[:-4])  # TODO: update filenames
00221
00222             if hostnames:
00223                 md_process = mp.Process(target=gmx_mdrun_mpi,
00224                                         args=(work_dir, seed, os.path.join(seed_dirs[seed], 'md.gro'), hostnames[i], cpu_map[i]))
00225                 # gmx_mdrun_mpi(work_dir, seed, seed_dirs[seed] + '/md.gro', hostnames[i], cpu_map[i])
00226             else:
00227                 md_process = mp.Process(target=gmx_mdrun, args=(work_dir, seed, os.path.join(seed_dirs[seed], 'md.gro')))
00228                 # gmx_mdrun(work_dir, seed, seed_dirs[seed] + '/md.gro')
00229             md_process.start()
00230             pid_arr.append(md_process)
00231         [proc.join() for proc in pid_arr]
00232         for i, seed in enumerate(seeds):
00233             gmx_trjconv(
00234                 f=os.path.join(seed_dirs[seed], 'md.xtc'),
00235                 o=os.path.join(seed_dirs[seed], 'md_prot.xtc'),
00236                 n=ndx_file,
00237                 b=1)  # , dump=20
00238
00239         results_arr = list(os.path.join(os.path.join(work_dir, str(seed)), 'md_prot.xtc') for seed in seeds)
00240         gmx_trjcat(f=results_arr, o=temp_xtc_file, n=ndx_file, cat=True, vel=False, sort=False, overwrite=True)
00241
00242     return temp_xtc_file
00243
00244
00245 # def get_ambient_noise_rmsd(goal_xtc, noize_file, goal_prot_only, mul=0.8):
00246 #     dist_arr = get_knn_dist_mdsctk(goal_xtc, noize_file, goal_prot_only)
00247 #     min_rmsd = min(dist_arr)*mul  # I expect that current min does not represent real min.
00248 #     print('Min rmsd for simulation is going to be : ', min_rmsd)
00249 #     return min_rmsd
00250 #
00251 #
00252 # def get_ambient_noise_angles(num_el, gro_file, noize_file, goal_bb_ndx, goal_angles, mul=0.8):
00253 #     # generate filename
00254 #     # convert_gro_to_xtc(gro_file, goal_bb_ndx)
00255 #     sincos_file = 'noise_sincos.dat'
00256 #     noize_file_bb = 'noize_bb.xtc'
00257 #     angle_file = 'noise_angle.dat'
00258 #
00259 #     gmx_trjconv(f=noize_file, o=noize_file_bb, n=goal_bb_ndx, s=gro_file)
00260 #     get_bb_to_angle_mdsctk(x=noize_file_bb, o=angle_file)
00261 #     get_angle_to_sincos_mdsctk(i=angle_file, o=sincos_file)
00262 #
00263 #     os.remove(angle_file)
00264 #
00265 #     with open(sincos_file, 'rb') as file:
00266 #         initial_1d_array = np.frombuffer(file.read(), dtype=np.float64 , count=-1)
00267 #     check_arr = np.reshape(initial_1d_array, (-1, num_el*2))
00268 #     del initial_1d_array
00269 #
00270 #     res_arr = [None]*check_arr.shape[0]
00271 #     for i in range(check_arr.shape[0]):
00272 #         res_arr[i] = np.sum(abs(check_arr[i] - goal_angles))
00273 #     return float(np.min(res_arr)*mul)
00274
00275
```

References gmx_wrappers.gmx_grompp(), gmx_wrappers.gmx_trjcat(), and gmx_wrappers.gmx_trjconv().
Referenced by GMDA_main.GMDA_main().

Here is the call graph for this function:



Here is the caller graph for this function:



3.16.1.7 get_angle_to_sincos_mdsctk()

```
NoReturn metric_funcs.get_angle_to_sincos_mdsctk (
                str  i =  'noise_angle.dat',
                str  o =  'noise_sincos.dat' )
```

'angles_to_sincos' - MDSCTK tool - converts dihedrals into sin/cos values

str i: filename that contains angle values in the binary form
str o: filename that contains sin/cos values in the binary form

Returns

Generates file with sin/cos values.

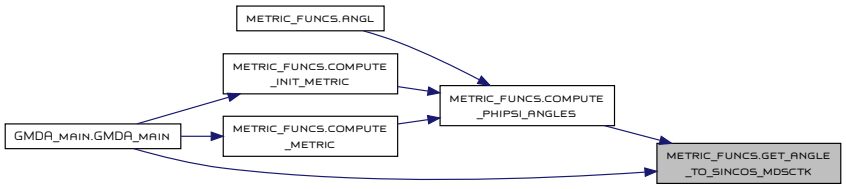
Definition at line 162 of file metric_funcs.py.

```
00162     """
00163     if os.path.exists(os.path.join(os.getcwd(), 'local.comp')):
00164         mdsctk_bash = 'source /opt/mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00165     else:
00166         mdsctk_bash = 'source ./mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00167     # angles_to_sincos -i angles_bb_315.dat -o sincos_bb_315.dat
00168     command = '{} angles_to_sincos -i {} -o {} 2>/dev/null 1>/dev/null'.format(
00169         mdsctk_bash, i, o)
00170     proc_obj = subprocess.Popen(
00171         os.path.expandvars(command), stdout=None, shell=True, stderr=None)
00172     # proc_obj = subprocess.Popen(os.path.expandvars(command), stdout=subprocess.PIPE, shell=True, stderr=None)
00173     try:
00174         output, error = proc_obj.communicate()
00175     except Exception as e:
00176         print(command)
00177         # print(e)
00178         raise Exception(e)
00179     if error:
00180         error = error.decode("utf-8")
00181         if 'error' in error.lower():
00182             print(command)
00183             print(error)
00184     if output:
```

```
00185         output = output.decode("utf-8")
00186         if 'error' in output.lower():
00187             print(command)
00188             print(output)
00189 
00190 
```

Referenced by compute_phipsi_angles(), and GMDA_main.GMDA_main().

Here is the caller graph for this function:



3.16.1.8 get_bb_to_angle_mdsctk()

```
NoReturn metric_funcs.get_bb_to_angle_mdsctk (
                str   x =  'noise_bb.xtc',
                str   o =  'noise_angle.dat' )
```

'bb_xtc_to_phipsi' - MDSCTK tool - takes backbone structure and computes dihedral angles between atoms

str x: backbone input trajectory
str o: filename of the binary C array

Returns

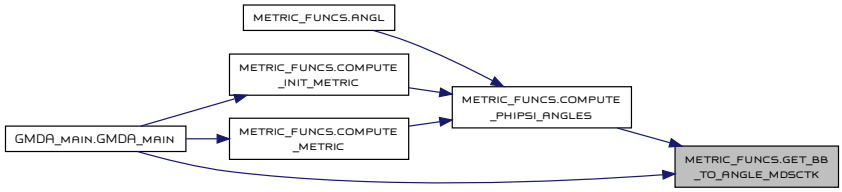
Generates a file with dihedral angles.

Definition at line 124 of file metric_funcs.py.

```
00124     """
00125     if os.path.exists(os.path.join(os.getcwd(), 'local.comp')):
00126         mdsctk_bash = 'source /opt/mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00127     else:
00128         mdsctk_bash = 'source ./mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00129     #  bb_xtc_to_phipsi -x traj_bb_315.xtc -o angles_bb_315.dat
00130     command = '{} bb_xtc_to_phipsi -x {} -o {} 2>/dev/null 1>/dev/null'.format(
00131         mdsctk_bash, x, o)
00132     proc_obj = subprocess.Popen(
00133         os.path.expandvars(command), stdout=None, shell=True, stderr=None)
00134     # proc_obj = subprocess.Popen(os.path.expandvars(command), stdout=subprocess.PIPE, shell=True, stderr=None)
00135     try:
00136         output, error = proc_obj.communicate()
00137     except Exception as e:
00138         print(command)
00139         # print(e)
00140         raise Exception(e)
00141     if error:
00142         error = error.decode("utf-8")
00143         if 'error' in error.lower():
00144             print(command)
00145             print(error)
00146     if output:
00147         output = output.decode("utf-8")
00148         if 'error' in output.lower():
00149             print(command)
00150             print(output)
00151 
00152 
```

Referenced by compute_phipsi_angles(), and GMDA_main.GMDA_main().

Here is the caller graph for this function:



3.16.1.9 get_contat_profile_mdsctk()

```
np.ndarray metric_funcs.get_contat_profile_mdsctk (
                str  ref_file,
                str  fitfile,
                str  index,
                float  dist =  2.7 )
```

'contact_profile' - MDSCTK tool - computes number of contacts between two (or more) structures

str ref_file: reference file - .xtc or .gro filename
str fitfile: .xtc or .gro filename - structure will be centered according
to the fitfile and used in distance computation
str index: .ndx file to compute distance among particular atoms
floatdist: in Angstroms - how close should two atoms be, so treat them as a contact

Returns

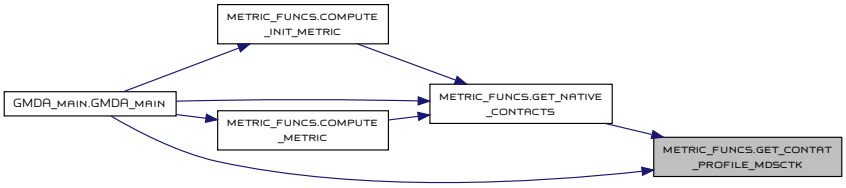
:return: ndarray, first value - number of indices with contacts, next N indices are atoms with contact :rtype np.ndarray

Definition at line 78 of file metric_funcs.py.

```
00078     """
00079     if os.path.exists(os.path.join(os.getcwd(), 'local.comp')):
00080         mdsctk_bash = 'source /opt/mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00081     else:
00082         mdsctk_bash = 'source ./mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00083 
00084     slash_pos = fitfile.rfind('/')
00085     if slash_pos >= 0:
00086         unique_name = '{}/{}.svi'.format(fitfile[:slash_pos], fitfile.split('/')[-1].split('.')[0])
00087     else:
00088         unique_name = '{}.svi'.format(fitfile.split('/')[-1].split('.')[0])
00089     command = '{} contact_profile -p {} -x {} -n {} -e {} -i {} -d /dev/null 2>/dev/null 1>/dev/null'.format(
00090         mdsctk_bash, ref_file, fitfile, index, dist, unique_name)
00091     proc_obj = subprocess.Popen(os.path.expandvars(command), stdout=None, shell=True, stderr=None)
00092     try:
00093         output, error = proc_obj.communicate()
00094     except Exception as e:
00095         print(command)
00096         print(e)
00097         return None
00098     if error:
00099         error = error.decode("utf-8")
00100         if 'error' in error.lower():
00101             print(command)
00102             print(error)
00103     if output:
00104         output = output.decode("utf-8")
00105         if 'error' in output.lower():
00106             print(command)
00107             print(output)
00108     cont_arr = np.fromfile(unique_name, dtype=np.uint32)
00109 
00110     os.remove(unique_name)
00111 
00112     return cont_arr
00113 
00114 
```

Referenced by get_native_contacts(), and GMDA_main.GMDA_main().

Here is the caller graph for this function:



3.16.1.10 get_knn_dist_mdsctk()

```
list metric_funcs.get_knn_dist_mdsctk (
        str  ref_file,
        str  fitfile,
        str  topology )
```

'knn_rms' - MDSCTK tool - computes RMSD between two (or more) structures

str ref_file: reference file - .xtc or .gro filename
str fitfile: .xtc or .gro filename - structure will be centered according to the fitfile and used in distance computation
str topology: .top topology file of the simulation box

Returns

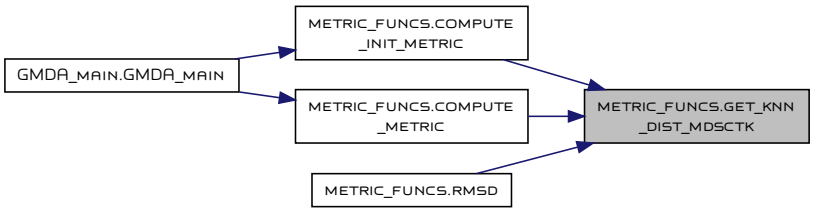
:return: list of RMSD distances from all frames to the goal :rtype: list

Definition at line 37 of file metric_funcs.py.

```
00037     """
00038     if os.path.exists(os.path.join(os.getcwd(), 'local.comp')):
00039         mdsctk_bash = 'source /opt/mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00040     else:
00041         mdsctk_bash = 'source ./mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00042 
00043     command = '{} knn_rms -s {} -p {} -r {} -f {}'.format(mdsctk_bash, 0, topology, ref_file, fitfile)
00044     proc_obj = subprocess.Popen(os.path.expandvars(command), stdout=subprocess.PIPE, shell=True, stderr=None)
00045     try:
00046         output, error = proc_obj.communicate()
00047     except Exception as e:
00048         print(e)
00049         return None
00050     if error:
00051         error = error.decode("utf-8")
00052         if 'error' in error.lower():
00053             print(error)
00054     if output:
00055         output = output.decode("utf-8")
00056         if 'error' in output.lower():
00057             print(output)
00058     dist_arr = np.fromfile('distances.dat', dtype=np.double)
00059     os.remove('distances.dat')
00060     os.remove('indices.dat')
00061 
00062     return dist_arr.tolist()
00063 
00064 
```

Referenced by compute_init_metric(), compute_metric(), and rmsd().

Here is the caller graph for this function:



3.16.1.11 get_native_contacts()

```
tuple metric_funcs.get_native_contacts (
        str goal_prot_only,
        list files_to_check,
        str ndx_file,
        np.ndarray cont_corr,
        int atom_num,
        float dist = 2.7,
        np.ufunc logic_fun = np.logical_xor,
        list h_filter = None,
        mp.Pool pool = None,
        bool just_contacts = False )
```

Computes number of contacts between the goal_prot_only and files_to_check.
If files to check is a single list of contacts, then function returns int and list Otherwise it returns list of ints and list of lists

str goal_prot_only: .gro filename with stripped waters and salt
list files_to_check: .xtc filename with frames we want to measure number of contacts with the goal
str ndx_file: .ndx - index filename to select protein only in .xtc
np.ndarray cont_corr: correct contacts between goal and goal (no mistakes) to compare with the files_to_check
int atom_num: number of atoms used for memory (structure) allocation
dist: distance that defines a contact
np.ufunc logic_fun: defines what relation between the goal and the files_to_check we want to measure - AND, XOR
:type logic_fun: Numpy logic function, typically logical_xor or logical_and
list h_filter: bool ean array with 1s in positions of H atoms, used to filter the final contacts
mp.Pool pool: CPU pool - passed, since each instance does not deallocate the RAM
bool just_contacts: flags to skip computation of the sum of correct contacts

Returns

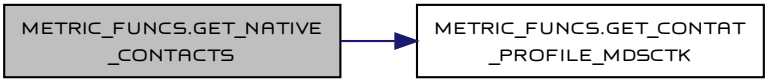
:return: sum of the correct contacts and contacts. :rtype: tuple

Definition at line 384 of file metric_funcs.py.

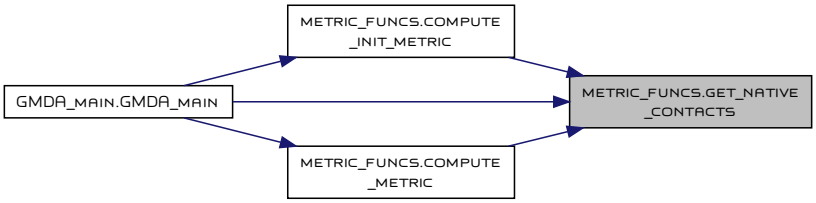
```
00384         :return: sum of the correct contacts and contacts.
00385         return type: tuple
00386     """
00387     # nat_cont_arr = list()
00388     # contacts = list()
00389     if len(files_to_check) == 0:
00390         return None
00391     elif len(files_to_check) > 1:  # case for many files with one frame
00392         if pool is None:
00393             # pool = mp.Pool(mp.cpu_count())  # creation pool every time creates memory leak on python3.6.6 compiled with gcc 8.2.0
00394             raise Exception('Please pass pool variable')
00395         # ind = [get_contat_profile_mdsctk(goal_prot_only, file, ndx_file, dist)[1:] for file in files_to_check]
00396         ind = [elem[1:] for elem in pool.starmap(get_contat_profile_mdsctk,
00397                                                  ((goal_prot_only, file, ndx_file, dist) for file in files_to_check))]
00398         # corr_len = [elem[:1] for elem in ind if len(elem) > 0]
00399         contacts = [None] * len(ind)
00400         for i in range(len(ind)):
00401             elem = np.zeros(atom_num * atom_num, dtype=np.bool)
00402             elem[ind[i]] = True
00403             contacts[i] = elem
00404         del ind, elem, i
00405     else:  # case for one file with any number of frames
00406         cont_arr = get_contat_profile_mdsctk(goal_prot_only, files_to_check[0], ndx_file, dist)
00407         # print('Done with cont prof')
```

```
00408         if cont_arr[0] + 1 == len(cont_arr):  # we have only one frame
00409             full_arr = np.zeros(atom_num * atom_num, dtype=np.bool)
00410             full_arr[cont_arr[1:]] = True
00411             contacts = [full_arr]
00412             del full_arr
00413         else:  # we have many frames
00414             tot_ind = 0
00415             contacts = list()
00416             while tot_ind < len(cont_arr):
00417                 tot_ind += 1
00418                 next_ind = tot_ind + cont_arr[tot_ind - 1]
00419                 full_arr = np.zeros(atom_num * atom_num, dtype=np.bool)
00420                 full_arr[cont_arr[tot_ind:next_ind]] = True
00421                 contacts.append(full_arr)
00422                 tot_ind += cont_arr[tot_ind - 1]
00423             del cont_arr, tot_ind, next_ind, full_arr
00424     if not just_contacts:
00425         if h_filter is not None:
00426             contacts = [np.logical_and(arr_elem, h_filter) for arr_elem in contacts]  # while here we can just use logic_fun,
00427             # since we use filter only with AND to compute AND_H, I took a safe path
00428         nat_cont_sum_arr = [logic_fun(arr_elem, cont_corr).sum() for arr_elem in contacts]
00429     else:
00430         nat_cont_sum_arr = [None] * len(contacts)
00431
00432     if len(nat_cont_sum_arr) == 1:
00433         return nat_cont_sum_arr[0], contacts[0]
00434     return nat_cont_sum_arr, contacts
00435
00436
```

References get_contat_profile_mdsctk().
Referenced by compute_init_metric(), compute_metric(), and GMDA_main.GMDA_main().
Here is the call graph for this function:



Here is the caller graph for this function:



3.16.1.12 rmsd()

```
NoReturn  metric_funcs.rmsd (
                mp.Queue q,
                str  combined_pg,
                str  temp_xtc_file,
                str  goal_prot_only,
                np.float64  prev_tot_dist )
```

Separate RMSD computation, used to be executed in parallel,.
NOT used anymore since does not result in any significant speed up, but left here "just in case".

mp.Queue q: queue used to communicate with the parent process

str combined_pg: two frames previous and goal
str temp_xtc_file: new frames (same as number of seeds) you want to measure distance from previous and to the goal
str goal_prot_only: goal protein only conformation
np.float64 rev_tot_dist: distance accumulated from the origin

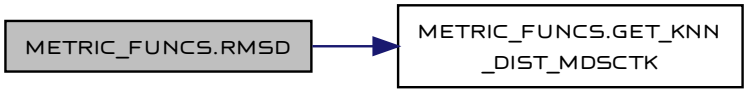
Returns

:return: Returns by putting into the queue (metric to goal, metric from previous, total traveled in metric units).

Definition at line 501 of file metric_funcs.py.

```
00501     """
00502     dist_arr = get_knn_dist_mdsctk(combined_pg, temp_xtc_file, goal_prot_only)
00503     from_prev_dist = dist_arr[0::2]
00504     rmsd_to_goal = dist_arr[1::2]
00505     rmsd_total_trav = [prev_tot_dist + elem for elem in from_prev_dist]
00506     q.put((rmsd_to_goal, from_prev_dist, rmsd_total_trav))
00507
00508
```

References get_knn_dist_mdsctk().
Here is the call graph for this function:



3.16.1.13 save_an_file()

NoReturn metric_funcs.save_an_file (
str an_file_name,
dict tol_error,
list metr_order)

Writes noise values into the specified file for future use during the restarts.

str an_file_name: ambient noise filename
dict tol_error: dict with ambient noise values for each metric
list metr_order: list of metrics used in the current run

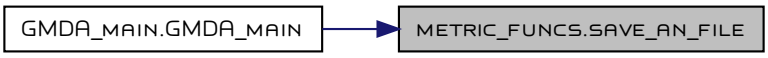
Returns

Generates a file with noise values.

Definition at line 356 of file metric_funcs.py.

```
00356     """
00357     with open(an_file_name, 'w') as f:
00358         for metr_name in metr_order:
00359             f.write('{}\n'.format(tol_error[metr_name]))
00360
00361
```

Referenced by GMDA_main.GMDA_main().
Here is the caller graph for this function:



3.16.1.14 select_metrics_by_snr()

```
str  metric_funcs.select_metrics_by_snr (
                list  cur_nodes,
                dict  prev_node,
                list  metric_names,
                dict  tol_error,
                bool compute_all_at_once,
                list  alowed_metrics,
                str  cur_metr )
```

SNR approach to a metric selection.
Metrics that had the highest SNR ratio (metric distance from the prev point)/(ambient noise) is selected next However, this approach does not always work and while you may a high SNR with contacts, there may be no real decrease in the rmsd. It is affected by the previous point performance.

```
list  cur_nodes: recent nodes
dict   prev_node: previous node
list  metric_names:   list  of metrics implemented (I want to know whole statistics, not only allowed metrics)
dict   tol_error:  dict   with noise data
bool   compute_all_at_once: toggle left as a reminder to not implement all at once
list  alowed_metrics:   list  of metrics that we allow to be used during the current run
str  cur_metr: name of the current metric
```

Returns

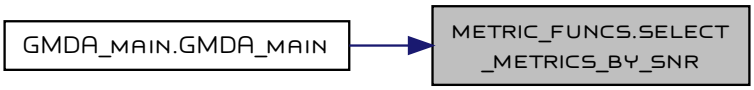
```
:return: metric name with the highest SNR
```

Definition at line 945 of file metric_funcs.py.

```
00945         :return: metric name with the highest SNR
00946     """
00947     if not compute_all_at_once:
00948         # easy to implement, but I do not have plans to use it since 'all at once' is very fast
00949         # just take last node and compute all metrics
00950         raise Exception('Not implemented')
00951 
00952     snr = False
00953     if snr:  # SNR approach may be biased. Additionally, prev_node should be computed here as prev point in name: s_1 is prev to s_1_3
00954         signal = dict ()
00955         best_metr = metric_names[0]
00956         best_val = -1
00957         for metr in metric_names:
00958             cur_name = '{}_to_goal'.format(metr)
00959             signal[metr] = 0
00960             for i in range(len(cur_nodes)):
00961                 signal[metr] += (cur_nodes[i][cur_name] - prev_node[cur_name]) / tol_error[metr]
00962             if metric_names != metric_names[0] and signal[metr] > best_val and metr in alowed_metrics:
00963                 best_val = signal[metr]
00964                 best_metr = metr
00965 
00966         if best_metr == cur_metr:
00967             print('New metric is the same as previous. Switching to next metric')
00968             while len(metric_names) > 1 and (best_metr == cur_metr or best_metr not in alowed_metrics):
00969                 best_metr = metric_names[(metric_names.index(best_metr) + 1) % len(metric_names)]
00970 
00971         print('SNR for metrics:')
00972         for metr in metric_names:
00973             if metr == best_metr:
00974                 print(' >*{}: {}'.format(metr, signal[metr]))
00975             elif best_val == signal[metr]:
00976                 print('  +{}: {}'.format(metr, signal[metr]))
00977             elif metr not in alowed_metrics:
00978                 print('   {}: {} # ignored'.format(metr, signal[metr]))
00979             else:
00980                 print('   {}: {}'.format(metr, signal[metr]))
00981     else:  # use round-robin
00982         best_metr = metric_names[(metric_names.index(cur_metr) + 1) % len(metric_names)]
00983         while best_metr not in alowed_metrics:
00984             print('Skipping {} since it is not in allowed list'.format(best_metr))
00985             best_metr = metric_names[(metric_names.index(cur_metr) + 1) % len(metric_names)]
00986         print('Switching to {}'.format(best_metr))
00987 
00988     return best_metr
```

Referenced by GMDA_main.GMDA_main().

Here is the caller graph for this function:



3.17 parse_topology_for_hydrogens Namespace Reference

Functions

- list parse_top_for_h (str top_filename)

  Reads the topology file and finds positions of the hydrogen atoms.

3.17.1 Function Documentation

3.17.1.1 parse_top_for_h() list parse_topology_for_hydrogens.parse_top_for_h (str top_filename)

Reads the topology file and finds positions of the hydrogen atoms.

top_filename: topology file .top

Returns

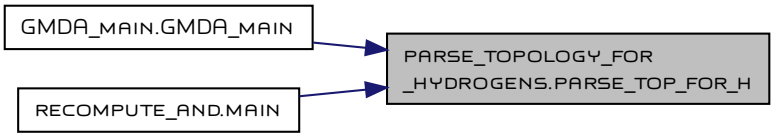
:return: list of hydrogen atoms position :rtype: list

Definition at line 10 of file parse_topology_for_hydrogens.py.

```
00010     """
00011     good_ind = list()
00012     with open(top_filename, 'r') as f:
00013         line = f.readline()
00014         while '[ atoms ]' not in line:
00015             line = f.readline()
00016         line = f.readline()
00017         atom_ind = line[1:].strip().split().index('atom')
00018         while ';' == line[0]:
00019             line = f.readline()
00020         line = line.strip()
00021         while len(line):
00022             if line[0] != ';':
00023                 parsed_line = line.split(';')[0].split()
00024                 if parsed_line[atom_ind][0] == 'H':
00025                     good_ind.append(int(parsed_line[0]))
00026                     # good_ind.append(int(parsed_line[0]) - 1)  # -1 for corr indexing
00027             line = f.readline().strip()
00028     return good_ind
00029
00030
00031 # parse_top_for_h('./prot_dir/topol.top')
```

Referenced by GMDA_main.GMDA_main(), and recompute_and.main().

Here is the caller graph for this function:



3.18 plot_energy Namespace Reference

Functions

- def main ()

3.18.1 Function Documentation

3.18.1.1 main() def plot_energy.main ( )

Definition at line 16 of file plot_energy.py.

```
00016 def main():
00017     past_dir = './past'
00018     db_to_connect = 'results_12'
00019     polynomial = False
00020     font = {'family': 'serif',
00021             'color': 'darkred',
00022             'weight': 'normal',
00023             'size': 16,
00024             }
00025     if not os.path.exists(db_to_connect + '.sqlite3'):
00026         raise Exception('DB not found')
00027 
00028     con = lite.connect(db_to_connect + '.sqlite3', check_same_thread=False, isolation_level=None)
00029     cur = con.cursor()
00030 
00031     qry = "select a.name, a.hashed_name from main_storage a  where a.goal_dist= ( select min(b.goal_dist) from main_storage b)"
00032     result = cur.execute(qry)
00033     all_res = result.fetchone()
00034     name = all_res[0]
00035     spname = name.split('_')
00036     all_prev_names = ['\'{}\"".format('_'.join(spname[:i])) for i in range(1, len(spname))]
00037     long_line = ", ".join(all_prev_names)
00038 
00039     qry = "select name, hashed_name from main_storage where name in ({})".format(long_line)
00040     result = cur.execute(qry)
00041     _ = result.fetchone()
00042     all_res = result.fetchall()
00043     names, hashed_names = zip(*all_res)
00044     wave = 100
00045     tot_chunks = int((len(hashed_names) + 1) / wave)
00046     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00047     gmx_eneconv(f=[os.path.join("./past", hashed_name) + '.edr' for hashed_name in hashed_names[:wave]], o='./combinded_energy.edr')
00048     for i in range(wave, len(hashed_names) + 1 - wave, wave):
00049         os.rename('./combinded_energy.edr', './combinded_energy_prev.edr')
00050         gmx_eneconv(f=["./combinded_energy_prev.edr"] + [os.path.join("./past", hashed_name + '.edr') for hashed_name in hashed_names[i:i +
    wave if i + wave < len(hashed_names) else -1]],
00051                     o='./combinded_energy.edr')
00052         if int(i / wave) % 10 == 0:
00053             print('{}/{} ({:.1f}%)'.format(int(i / wave), tot_chunks, 100 * int(i / wave) / tot_chunks))
00054 
00055     os.rename('./combinded_energy.edr', './combinded_energy_best.edr')
00056     print('Done with best')
00057 
00058 
00059 
00060     qry = "select a.name, a.hashed_name from main_storage a "
00061     result = cur.execute(qry)
00062     _ = result.fetchone()
00063     all_res = result.fetchall()
00064     names, hashed_names = zip(*all_res)
00065 
00066     # gmx_eneconv(f=[os.path.join(past_dir, hash_name+'.edr') for hash_name in hashed_names], o='./combinded_energy.edr')
00067 
00068     wave = 100
00069     tot_chunks = int((len(hashed_names)+1)/wave)
00070     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00071     gmx_eneconv(f=[os.path.join("./past", hashed_name)+'.edr' for hashed_name in hashed_names[:wave]], o='./combinded_energy.edr')
00072     for i in range(wave, len(hashed_names)+1-wave, wave):
00073         os.rename('./combinded_energy.edr', './combinded_energy_prev.edr')
00074         gmx_eneconv(f=["./combinded_energy_prev.edr"] +[os.path.join("./past", hashed_name + '.edr') for hashed_name in hashed_names[i:i+wave
    if i+wave < len(hashed_names) else -1]], o='./combinded_energy.edr')
00075         if int(i/wave) % 10 == 0:
00076             print('{}/{} ({:.1f}%)'.format(int(i/wave), tot_chunks, 100*int(i/wave)/tot_chunks))
00077 
00078     os.rename('./combinded_energy.edr', './combinded_energy_all_main.edr')
00079     print('Done with all main')
00080 
00081 
00082     qry = "select a.name, a.hashed_name from main_storage a join log b on a.id=b.id where b.dst='VIZ' order by b.timestamp"
```

```
00083     result = cur.execute(qry)
00084     _ = result.fetchone()
00085     all_res = result.fetchall()
00086     names, hashed_names = zip(*all_res)
00087 
00088     wave = 100
00089     tot_chunks = int((len(hashed_names)+1)/wave)
00090     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00091     gmx_eneconv(f=[os.path.join("./past", hashed_name)+'.edr' for hashed_name in hashed_names[:wave]], o='./combinded_energy.edr')
00092     for i in range(wave, len(hashed_names)+1-wave, wave):
00093         os.rename('./combinded_energy.edr', './combinded_energy_prev.edr')
00094         gmx_eneconv(f=["./combinded_energy_prev.edr"] +[os.path.join("./past", hashed_name + '.edr') for hashed_name in hashed_names[i:i+wave
      if i+wave < len(hashed_names) else -1]], o='./combinded_energy.edr')
00095         if int(i/wave) % 10 == 0:
00096             print('{}/{} ({:.1f}%)'.format(int(i/wave), tot_chunks, 100*int(i/wave)/tot_chunks))
00097 
00098     os.rename('./combinded_energy.edr', './combinded_energy_all_viz.edr')
00099     print('Done with viz')
00100 
00101 
00102     # gmx_energy('./combinded_energy.edr', './combinded_energy.xvg', fee=True, fetemp=300)
00103 
00104 
00105 
00106 if __name__ == '__main__':
00107     main()
```

References gmx_wrappers.gmx_eneconv().

Here is the call graph for this function:

3.19 plot_matplot_energy Namespace Reference

Functions

- def main ()
- int single_plot (int fig_num, dict ax_prop, list arr_A, list arr_B, list filenames_db, str marker, float mark_size, bool bsf, bool rev, bool shrink, str xlab, str ylab, str title, str filename, list extra_line=None, int mdpi=400)

3.19.1 Function Documentation

3.19.1.1 main() def plot_matplot_energy.main ( )

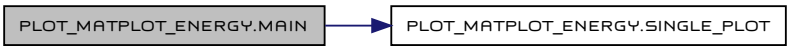
Definition at line 9 of file plot_matplot_energy.py.

```
00009 def main():
00010     filenames_found = [f.split("/")[-1] for f in os.listdir('./') if '.npy' in f]
00011     fig_num = 0
00012     for file in filenames_found:
00013         cur_arr = np.load(file)
00014         cur_arr = cur_arr.swapaxes(0, 1)
00015         new_name = file.split('.')[0]
00016         ax_prop = {"min_lim_x": min(cur_arr[0]), "max_lim_x": max(cur_arr[0]) + max(cur_arr[0]) / 80, "min_lim_y": min(cur_arr[1]),
      "max_lim_y": max(cur_arr[1]) - max(cur_arr[1]) / 80,
00017                    "min_ax_x": 0, "max_ax_x": max(cur_arr[0]) + max(cur_arr[0]) / 80, "min_ax_y": min(cur_arr[1]) + min(cur_arr[1]) / 80,
      "max_ax_y": max(cur_arr[1]) - max(cur_arr[1]) / 80,
00018                    "ax_step_x": (max(cur_arr[0]) - 0) / 16,
00019                    "ax_step_y": (max(cur_arr[1]) - min(cur_arr[1])) / 20}
00020         extra_line = [{"ax_type": 'ver', "val": 0, "name": "simulation origin", "col": "darkmagenta"}]
00021         fig_num = single_plot(fig_num, ax_prop, [cur_arr[0]], [cur_arr[1]], ['LJ interaction value'], '-', 1.0, True, True, False, 'Time, ps',
      'LJ-SR, kJ/mol', 'Lennard-Jones Short Range Protein-Protein Interaction', new_name, extra_line=extra_line)
00022         plt.close('all')
00023 
00024 
00025 def single_plot(fig_num: int, ax_prop: dict , arr_A: list, arr_B: list, filenames_db: list, marker: str, mark_size: float,
00026                 bsf: bool, rev: bool, shrink: bool, xlab: str, ylab: str,
00027                 title: str, filename: str, extra_line: list = None, mdpi: int = 400) -> int:
00028     """
```

```
00029
00030     Args:
00031         int fig_num:
00032         dict  ax_prop:
00033         list arr_A:
00034         list arr_B:
00035         list filenames_db:
00036         str  marker:
00037         float mark_size:
00038         bool bsf:
00039         bool rev:
00040         bool shrink:
00041         str  xlab:
00042         str  ylab:
00043         str  title:
00044         str  filename:
00045         list extra_line:
00046         int mdpi:
00047
00048     Returns:
```

References single_plot().

Here is the call graph for this function:



3.19.1.2 single_plot()

```
int plot_matplot_energy.single_plot (
                int  fig_num,
                dict  ax_prop,
                list  arr_A,
                list  arr_B,
                list  filenames_db,
                str  marker,
                float  mark_size,
                bool bsf,
                bool rev,
                bool shrink,
                str  xlab,
                str  ylab,
                str  title,
                str  filename,
                list   extra_line =  None,
                int  mdpi =  400 )
```

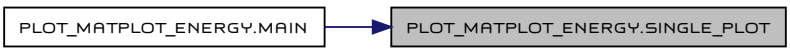
Definition at line 49 of file plot_matplot_energy.py.

```
00049         :return: last figure number.
00050         return type: int
00051     """
00052     fig_num += 1
00053
00054     w, h = figaspect(0.5)
00055     fig = plt.figure(fig_num, figsize=(w, h))
00056
00057     ax = fig.gca()
00058     plt.xlim(ax_prop["min_lim_x"], ax_prop["max_lim_x"])
00059     plt.ylim(ax_prop["min_lim_y"], ax_prop["max_lim_y"])
00060
00061     major_xticks = np.arange(ax_prop["min_ax_x"], ax_prop["max_ax_x"], ax_prop["ax_step_x"])
00062     major_yticks = np.arange(ax_prop["min_ax_y"], ax_prop["max_ax_y"], ax_prop["ax_step_y"])
00063
00064     if major_xticks is not None:
00065         ax.set_xticks(major_xticks)
00066     if major_yticks is not None:
00067         ax.set_yticks(major_yticks)
00068     # if minor_xticks is not None:
00069     #     ax.set_xticks(minor_xticks, minor=True)
00070     # if minor_yticks is not None:
00071     #     ax.set_yticks(minor_yticks, minor=True)
00072
00073     plt.grid(which='both')
00074     plt.xticks(rotation=30)
```

```
00075     plt.subplots_adjust(top=0.95, bottom=0.14, left=0.09, right=0.98)
00076
00077     lines_b = []
00078     for i, bsf_trav_to_goal in enumerate(arr_A):
00079         if not shrink:  # use provided array arr_B
00080             if rev:
00081                 line_b, = plt.plot(arr_A[i], arr_B[i], marker, markersize=mark_size)
00082             else:
00083                 line_b, = plt.plot(arr_B[i], arr_A[i], marker, markersize=mark_size)
00084         else:  # generate array from 0 to len(arr_A)
00085             if rev:
00086                 if bsf:
00087                     line_b, = plt.plot(arr_A[i], range(len(arr_A[i])), marker, markersize=mark_size)
00088                 else:
00089                     line_b, = plt.plot(arr_A[i], arr_B[i], marker, markersize=mark_size)
00090             else:
00091                 line_b, = plt.plot(range(len(arr_A[i])), arr_A[i], marker, markersize=mark_size)
00092         lines_b.append(line_b)
00093
00094     if extra_line is not None:
00095         for el in extra_line:
00096             if el["ax_type"] == 'ver':
00097                 straight_line = plt.axvline(x=el["val"], color=el["col"], linestyle='-')  #
00098             elif el["ax_type"] == 'hor':
00099                 straight_line = plt.axhline(y=el["val"], color=el["col"], linestyle='-')
00100             else:
00101                 raise Exception('Wrong ax type')
00102             lines_b.append(straight_line)
00103             filenames_db.append(el["name"])
00104         # if el["ax_type"] == 'ver':
00105         #     if not rev:
00106         #         ax.annotate('folding direction', xytext=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
       ax_prop["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 5 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 * ax_prop["ax_step_y"]),
       arrowprops={'arrowstyle': '->', 'lw': 1.5, 'color': 'mediumblue'}, va='center')  # -->
00107         #     else:
00108         #         ax.annotate('folding direction', xytext=(ax_prop["max_ax_x"] - 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
       ax_prop["ax_step_y"]), xy=(ax_prop["max_ax_x"] - 5 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 * ax_prop["ax_step_y"]),
       arrowprops={'arrowstyle': '->', 'lw': 1.5, 'color': 'mediumblue'}, va='center')  # -->
00109         # else:
00110         #     if not rev:
00111         #         ax.annotate('folding direction', xytext=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
       ax_prop["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 4 * ax_prop["ax_step_y"]),
       arrowprops={'arrowstyle': '->', 'lw': 1.5, 'color': 'mediumblue'}, ha='center')  # <--
00112         #     else:
00113         #         pass # does not exist
00114                 # ax.annotate('folding direction', xytext=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
       ax_prop["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 4 * ax_prop["ax_step_y"]),
       arrowprops={'arrowstyle': '->', 'lw': 1.5, 'color': 'mediumblue'}, ha='center')  # -->
00115
00116     ax.legend(lines_b, filenames_db)
00117     plt.xlabel(xlab)
00118     plt.ylabel(ylab)
00119     plt.title(title)
00120     try:
00121         plt.savefig(filename, dpi=mdpi)
00122     except:
00123         plt.show()
00124     plt.close('all')
00125     return fig_num
00126
00127
```

Referenced by main().

Here is the caller graph for this function:



3.20 print_best_frame Namespace Reference

Functions

- def main ()

3.20.1 Function Documentation

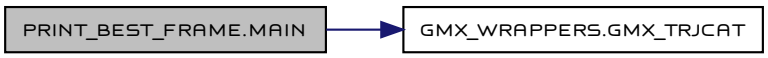
3.20.1.1 main() def print_best_frame.main ( )

Definition at line 9 of file print_best_frame.py.

```
00009 def main():
00010     if len(sys.argv) < 2:
00011         raise Exception('Not enough arguments')
00012     # db_to_connect = 'results_12'
00013     db_to_connect = sys.argv[1]
00014     past_dir = './past'
00015     if not os.path.exists(db_to_connect + '.sqlite3'):
00016         raise Exception('DB not found')
00017 
00018     con = lite.connect(db_to_connect + '.sqlite3', check_same_thread=False, isolation_level=None)
00019     cur = con.cursor()
00020 
00021     qry = "select a.name, a.hashed_name from main_storage a  where a.goal_dist= ( select min(b.goal_dist) from main_storage b)"
00022     result = cur.execute(qry)
00023     all_res = result.fetchone()
00024     name = all_res[0]
00025     spname = name.split('_')
00026     all_prev_names = ['\'{}\"'.format('_'.join(spname[:i])) for i in range(1, len(spname))]
00027     long_line = ", ".join(all_prev_names)
00028 
00029     qry = "select name, hashed_name from main_storage where name in ({})".format(long_line)
00030     result = cur.execute(qry)
00031     all_res = result.fetchall()
00032     names, hashed_names = zip(*all_res)
00033     wave = 100
00034     tot_chunks = int((len(hashed_names) + 1) / wave)
00035     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00036     if os.path.exists('./combinded_traj.xtc'):
00037         os.remove('./combinded_traj.xtc')
00038     if os.path.exists('./combinded_traj_prev.xtc'):
00039         os.remove('./combinded_traj_prev.xtc')
00040 
00041     gmx_trjcat(f=[os.path.join(past_dir, hashed_name) + '.xtc' for hashed_name in hashed_names[:wave]], o='./combinded_traj.xtc',
      n='./prot_dir/prot.ndx', cat=True, vel=False, sort=False, overwrite=True)
00042     for i in range(wave, len(hashed_names), wave):
00043         os.rename('./combinded_traj.xtc', './combinded_traj_prev.xtc')
00044         gmx_trjcat(f=[" ./combinded_traj_prev.xtc "] + [os.path.join(past_dir, hashed_name) + '.xtc' for hashed_name in
      hashed_names[i:i+wave]], o='./combinded_traj.xtc', n='./prot_dir/prot.ndx', cat=True, vel=False, sort=False, overwrite=True)
00045         if int(i / wave) % 10 == 0:
00046             print('{}/{} ({:.1f}%)'.format(int(i / wave), tot_chunks, 100 * int(i / wave) / tot_chunks))
00047 
00048     if os.path.exists('./combinded_traj.xtc'):
00049         os.rename('./combinded_traj.xtc', './{}_traj_best.xtc'.format(db_to_connect))
00050     if os.path.exists('./combinded_traj_prev.xtc'):
00051         os.remove('./combinded_traj_prev.xtc')
00052     print('Done with best for {}'.format(db_to_connect))
00053 
00054 
```

References gmx_wrappers.gmx_trjcat().

Here is the call graph for this function:



3.21 print_nat_cont Namespace Reference

Functions

- def main ()

3.21.1 Function Documentation

3.21.1.1 main() def print_nat_cont.main ( )

Definition at line 7 of file print_nat_cont.py.

```
00007 def main():
00008
00009     # with open('output.dat', 'r') as infile:
00010     #     arr = infile.readlines()
00011     #
00012     # arr = [int(val.strip()) for val in arr]
00013     arr = np.load('nat_cont_300_1_9_AND_H.npz')
00014     arr = arr[arr.files[0]]
00015     avg = reduce(lambda a, b: a + b, arr) / len(arr)
00016     # arr = [elem for elem in arr if elem < avg*5]
00017     max_val = max(arr)
00018     min_val = min(arr)
00019
00020
00021     fig_num = 0
00022     mdpi = 400
00023     major_xticks = None
00024     minor_xticks = None
00025     major_yticks = None
00026     minor_yticks = None
00027     w, h = figaspect(0.5)
00028     fig = plt.figure(fig_num, figsize=(w, h))
00029     plt.xlim(0, len(arr))
00030     ax = fig.gca()
00031     major_xticks = np.arange(0, len(arr) + len(arr) / 10, len(arr) / 10)
00032     if max_val - min_val > 0:
00033         major_yticks = np.arange(min_val, max_val + max_val / 16, (max_val - min_val) / 16)
00034     if major_xticks is not None:
00035         ax.set_xticks(major_xticks)
00036     if minor_xticks is not None:
00037         ax.set_xticks(minor_xticks, minor=True)
00038     if major_yticks is not None:
00039         ax.set_yticks(major_yticks)
00040     if minor_yticks is not None:
00041         ax.set_yticks(minor_yticks, minor=True)
00042     plt.grid(which='both')
00043     lines = []
00044
00045     line, = plt.plot(range(len(arr)), arr, '-', markersize=1)
00046     lines.append(line)
00047     ax.legend(lines, 'full cont')
00048     plt.xlabel("frame")
00049     plt.ylabel("contacts AND goal")
00050     plt.title('nat Hydrogen contacts (AND) for 20ns gb1 simulation for 300K d=1.9 (higher is better)')
00051     plt.savefig('nat_cont_300_1_9_AND_H.png', dpi=mdpi)
00052
00053 main()
```

3.22 rebuild Namespace Reference

Variables

- string filename = 's_5_6_5_2_5_2_7_6_7_4_6_3_4_4_7_4_3_7_5_6_5_1_2_7_1_1_5_1_5_6_1_1_4_6_3_4_2_3_0_1_0_4_4_7_5_5_1_0_3_2_2_2_5_2_7_4_0_0_7_↩
  1_0_6_6_7_3_6_7_3_4_3_2_4_1_1_3_4_6_4_4_1_6_3_4_7_0_2_6_3_0_2_1_0_0_4_7_1_3_6_0_5_5_5_0_4_5_3_7_5_7_4_3_0_6.xtc'
- string ext = filename.split('.')[1]
- string arr = filename.split('.')[0].split('_')
- list good_arr = [ ]
- string cummulative = ''
- f
- o
- n
- cat
- True
- vel
- False
- sort
- overwrite

3.22.1 Variable Documentation

3.22.1.1 arr string rebuild.arr = filename.split('.')[0].split('_')

Definition at line 5 of file rebuild.py.

3.22.1.2 cat rebuild.cat
Definition at line 13 of file rebuild.py.

3.22.1.3 cummulative string rebuild.cummulative = ''
Definition at line 7 of file rebuild.py.

3.22.1.4 ext string rebuild.ext = filename.split('.')[1]
Definition at line 4 of file rebuild.py.

3.22.1.5 f rebuild.f
Definition at line 13 of file rebuild.py.

3.22.1.6 False rebuild.False
Definition at line 13 of file rebuild.py.

3.22.1.7 filename string rebuild.filename = 's_5_6_5_2_5_2_7_6_7_4_6_3_4_4_7_4_3_7_5_6_5_1_2_7_1_1_5_1_5_6_1_1_4_6_3_4_2_↩
3_0_1_0_4_4_7_5_5_1_0_3_2_2_2_5_2_7_4_0_0_7_1_0_6_6_7_3_6_7_3_4_3_2_4_1_1_3_4_6_4_4_1_6_3_4_7_0_2_6_3_0_2_1_0_0_4_7_1_3_6_0_5_5_↩
5_0_4_5_3_7_5_7_4_3_0_6.xtc'
Definition at line 3 of file rebuild.py.

3.22.1.8 good_arr rebuild.good_arr = [ ]
Definition at line 6 of file rebuild.py.

3.22.1.9 n rebuild.n
Definition at line 13 of file rebuild.py.

3.22.1.10 o rebuild.o
Definition at line 13 of file rebuild.py.

3.22.1.11 overwrite rebuild.overwrite
Definition at line 13 of file rebuild.py.

3.22.1.12 sort rebuild.sort
Definition at line 13 of file rebuild.py.

3.22.1.13 True rebuild.True
Definition at line 13 of file rebuild.py.

3.22.1.14 vel rebuild.vel
Definition at line 13 of file rebuild.py.

3.23 recompute_and Namespace Reference

Functions

- def main ()

3.23.1 Function Documentation

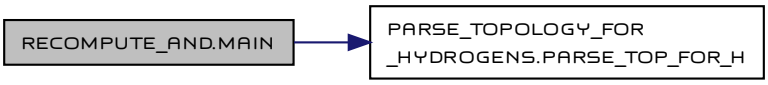
3.23.1.1 main() def recompute_and.main ( )
Definition at line 10 of file recompute_and.py.

```
00010 def main():
00011     cont_corr = np.load('cor_cont_300_1_9.npz')
00012     cont_corr = cont_corr[cont_corr.files[0]]
00013 
00014     contacts = np.load('full_cont_300_1_9.npz')
00015     contacts = contacts[contacts.files[0]]
00016     print('Corr contacts count: {}'.format(np.sum(cont_corr)))
00017     compute_h_only = False
00018     if compute_h_only:
00019         h_pos = parse_top_for_h('./prot_dir/topol.top')
00020         num_atoms = int(math.sqrt(len(contacts[0])))
00021         h_filter = np.zeros(num_atoms * num_atoms, dtype=np.uint8)
00022         for pos in h_pos:
```

```
00023             h_filter[(pos-1)*num_atoms:pos*num_atoms] = 1
00024         cont_corr_h = np.logical_and(cont_corr, h_filter)
00025         cont_corr = cont_corr_h
00026     pool = mp.Pool(mp.cpu_count())
00027     nat_cont_arr = [pool.apply(np.logical_xor, args=(cont_arr, cont_corr)) for cont_arr in contacts]
00028     print('Done with and')
00029     nat_cont_arr = [pool.apply(np.sum, args=(elem,)) for elem in nat_cont_arr]
00030     np.savez('nat_cont_300_1_9_XOR.npz', nat_cont_arr)
00031
00032
00033 main()
```

References parse_topology_for_hydrogens.parse_top_for_h().

Here is the call graph for this function:



3.24 test Namespace Reference

Functions

- def add_task (task, priority=0)
- def pop_task ()

Variables

- list pq = [ ]
- dict ionary entry_finder = {}
- string REMOVED = '<removed-task>'
- counter = itertools.count()

3.24.1 Function Documentation

3.24.1.1 add_task() def test.add_task (task, priority = 0)

```
Definition at line 9 of file test.py.
00009 def add_task(task, priority=0):
00010     'Add a new task or update the priority of an existing task'
00011     count = next(counter)
00012     entry = [priority, count, task]
00013     entry_finder[task] = entry
00014     heapq.heappush(pq, entry)
00015
00016
00017
```

Referenced by pop_task().

Here is the caller graph for this function:

3.24.1.2 pop_task() def test.pop_task ( )

Definition at line 18 of file test.py.

```
00018 def pop_task():
00019     'Remove and return the lowest priority task. Raise KeyError if empty.'
00020     while pq:
00021         priority, count, task = heapq.heappop(pq)
00022         if task is not REMOVED:
00023             del entry_finder[task]
00024             return task
00025     raise KeyError('pop from an empty priority queue')
00026
00027 add_task('kva10', 10)
00028 add_task('kva12', 12)
00029 add_task('kva7', 7)
00030 add_task('kva10', 10)
00031 add_task('kva10', 10)
00032 add_task('kva10', 10)
00033 add_task('kva10', 10)
00034 add_task('kva10', 10)
00035 add_task('kva10', 10)
00036 add_task('kva10', 10)
00037 add_task('kva10', 10)
```

References add_task().

Here is the call graph for this function:

3.24.2 Variable Documentation

3.24.2.1 counter test.counter = itertools.count()

Definition at line 7 of file test.py.

3.24.2.2 entry_finder dict ionary test.entry_finder = {}

Definition at line 5 of file test.py.

3.24.2.3 pq list test.pq = []

Definition at line 4 of file test.py.

3.24.2.4 REMOVED string test.REMOVED = '<removed-task>'

Definition at line 6 of file test.py.

3.25 testll Namespace Reference

Functions

- def permute (word)
- def permute_driver (word)
- def main ()

3.25.1 Function Documentation

3.25.1.1 main() def testll.main ( )

Definition at line 21 of file testll.py.

```
00021 def main():
00022     permute_driver('abcdefr')
00023
00024
00025
```

References permute_driver().
Here is the call graph for this function:

3.25.1.2 permute() def testll.permute (word)

Definition at line 1 of file testll.py.

```
00001 def permute(word):
00002     if len(word) == 1: return [word]
00003     a = list()
00004     for i in range(len(word)):
00005         res = permute(word[0:i]+word[i+1:])
00006         for j in range(len(res)):
00007             res[j] = word[i] + res[j]
00008         a.extend(res)
00009     return a
00010
00011
```

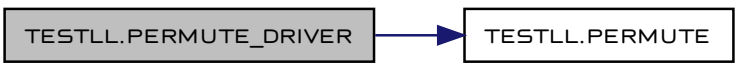
Referenced by permute_driver().
Here is the caller graph for this function:

3.25.1.3 permute_driver() def testll.permute_driver (word)

Definition at line 12 of file testll.py.

```
00012 def permute_driver(word):
00013     a = list()
00014     for i in range(len(word)):
00015         res = permute(word[0:i]+word[i+1:])
00016         for j in range(len(res)):
00017             res[j] = word[i] + res[j]
00018         a.extend(res)
00019     print(len(a))
00020
```

References permute().
Referenced by main().
Here is the call graph for this function:



Here is the caller graph for this function:

3.26 threaded_funcs Namespace Reference

Functions

- NoReturn print_async (str info_form_str, tuple tup)

  Test function used for async printing.
- NoReturn threaded_print (mp.JoinableQueue pipe)

  Prints statement provided from the pipe.
- NoReturn threaded_db_input (mp.JoinableQueue pipe, int len_seeds)

  Runs DB operation in a separate process.
- NoReturn threaded_copy (mp.JoinableQueue pipe)

  Recieves filenames (A, B) from the pipe and tries to copy A into B.
- NoReturn threaded_rm (mp.JoinableQueue pipe)

  Recieves filename from the pipe and tries to remove them.

3.26.1 Function Documentation

3.26.1.1 print_async()

NoReturn threaded_funcs.print_async (
str info_form_str,
tuple tup)

Test function used for async printing.

str info_form_str: formatting string.
tuple tup: data to print.

Returns

Simply prints the string.

Definition at line 29 of file threaded_funcs.py.

```
00029     """
00030     print(info_form_str.format(*tup))
00031
00032
```

Recieves filenames (A, B) from the pipe and tries to copy A into B.

pipe: connection with the parent

Returns

Copies files in the background.

Definition at line 102 of file threaded_funcs.py.

```
00102     """
00103     stmt = pipe.get(timeout=3600)
00104     while stmt is not None:
00105         # with COPY_LOCK:
00106         cp2(stmt[0], stmt[1])
00107         pipe.task_done()
00108         stmt = pipe.get(timeout=1800)
00109
00110
```

Runs DB operation in a separate process.

pipe: connection with the parent.
len_seeds: total number of seeds.

Returns

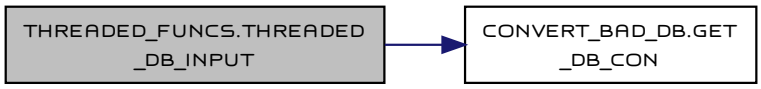
Executes the queries from the queue.

Definition at line 67 of file threaded_funcs.py.

```
00067     """
00068     con, dbname = get_db_con(len_seeds)
00069     stmt = pipe.get(timeout=3600)
00070     pid = None
00071     while stmt is not None:
00072         try:
00073             pid.join()
00074         except Exception as e:
00075             if pid:
00076                 print(e)
00077         # try:
00078         # con = con = lite.connect(dbname, timeout=3000, check_same_thread=False, isolation_level=None)
00079         # con.commit()
00080         pid = mp.Process(target=stmt[0], args=(con,)+stmt[1])
00081         pid.start()
00082         # except Exception as e:
00083         #     print('Found exception in db input:')
00084         #     print(e)
00085         #     print('Arguments that caused exception: ')
00086         #     print(stmt)
00087         # finally:
00088         pipe.task_done()
00089         stmt = pipe.get()
00090     print('DB thread exiting...')
00091     con.close()
00092
00093
```

References convert_bad_db.get_db_con().

Here is the call graph for this function:



3.26.1.2 threaded_print()

NoReturn threaded_funcs.threaded_print (mp.JoinableQueue pipe)

Prints statement provided from the pipe.
Typically, you supply formating string and options

mp.JoinableQueue pipe: source of the perforated strings and values (str, vals).

Returns

Simply prints the string.

Definition at line 43 of file threaded_funcs.py.

```
00043     """
00044     stmt = pipe.get(timeout=3600)
00045     while stmt is not None:
00046         try:
00047             # with PRINT_LOCK:
00048             #     print(stmt[0].format(*stmt[1]))
00049             print(stmt[0].format(*stmt[1]))
00050         except Exception as e:
00051             print(e)
00052         finally:
00053             pipe.task_done()
00054             stmt = pipe.get()
00055     print('Print thread exiting...')
00056
00057
```

Recieves filename from the pipe and tries to remove them.

pipe: connection with the parent

Returns

Removes files in the background.

Definition at line 119 of file threaded_funcs.py.

```
00119     """
00120     stmt = pipe.get(timeout=3600)
00121     while stmt is not None:
00122         # with RM_LOCK:
00123         try:
00124             os.remove(stmt)
00125         except Exception as e:
00126             print('Was not able to remove {}, Error: {}'.format(stmt, e))
00127         pipe.task_done()
00128         stmt = pipe.get(timeout=1800)
```

4 File Documentation

4.1 compare_db_perf_new_format.py File Reference

Namespaces

- compare_db_perf_new_format

Functions

- def compare_db_perf_new_format.main ()
- def compare_db_perf_new_format.gen_all (list filenames_db, list legend_names, str common_path)

  Takes the tasks and processes them either one by one or in parallel.
- def compare_db_perf_new_format.best_traj (int fig_num, list filenames_db, list legend_names, str guide_metr, str common_path)

  This is just a basic comparison among metrics.
- int compare_db_perf_new_format.plot_all_best_traj (int fig_num, list cur_arr, list filenames_db, list legend_names, str guide_metr, str common_path)
- def compare_db_perf_new_format.plot_sep_best_traj (fig_num, cur_arr, filenames_db, legend_names, guide_metr, common_path)
- int compare_db_perf_new_format.guide_metr_usage (int fig_num, list filenames_db, list legend_names, str guide_metr, str common_path)
- int compare_db_perf_new_format.plot_all_metrics (int fig_num, list cur_arr, list filenames_db, list legend_names, str guide_metr, str common_path)

  General force field comparison: sampling, best_so_far, dist traveled.
- int compare_db_perf_new_format.plot_only_one_metric (int fig_num, list cur_arr, list filenames_db, float init_rmsd, list legend_names, str metric_name, str guide_metr, str common_path)
- int compare_db_perf_new_format.plot_set (int fig_num, list to_goal_arr, list legend_names, float max_len, float max_non_init_rmsd, float init_metr, list bsf_arr, float common_point, float max_trav, list trav_arr, str full_cut, str metric, str metr_units, str same, str custom_path, bool shrink, list non_shrink_arr=None)
- int compare_db_perf_new_format.single_plot (int fig_num, dict ax_prop, list arr_A, list arr_B, list filenames_db, str marker, float mark↩ _size, bool bsf, bool rev, bool shrink, str xlab, str ylab, str title, str filename, list extra_line=None, int mdpi=400, dict second_↩ ax=None, list sec_arr=None)

  Main plotting function.

4.2 compare_db_perf_new_format.py

```
00001 #!/usr/bin/env python3
00002
00003 import os
00004 import sqlite3 as lite
00005 import matplotlib.pyplot as plt
00006 import numpy as np
00007 from matplotlib.figure import figaspect
00008 import multiprocessing as mp
00009 import math
00010
00011
00012 def main():
00013     """
00014     This function sets the task.
00015     Our task is to compare different runs by plotting plots.
00016     You specify DB names and proper legend entrees
00017     """
00018     batch_arr = list()
00019     ffs = ['amber', 'charm', 'gromos', 'opls']
00020     #########   TRP   ######################
00021     # for ff in ffs:
00022     #     filenames_db = ['results_{}_trp_300_fixed.sqlite3'.format(ff), 'results_{}_trp_300_2_fixed.sqlite3'.format(ff)]
00023     #     legend_names = ['TRP {}_1'.format(ff), 'TRP {}_2'.format(ff)]
00024     #     common_path = '../trp_{}_compar'.format(ff)
00025     #     batch_arr.append((filenames_db, legend_names, common_path))
```

```
00026
00027     filenames_db = ['results_amber_trp_300_2_fixed.sqlite3', 'results_charm_trp_300_2_fixed.sqlite3', 'results_gromos_trp_300_2_fixed.sqlite3',
      'results_opls_trp_300_2_fixed.sqlite3']
00028     # legend_names = ['TRP amber_2', 'TRP charm_2', 'TRP gromos_2', 'TRP opls_2']
00029     legend_names = ['1L2Y, 2nd run with AMBER ff', '1L2Y, 2nd run with CHARM ff', '1L2Y, 2nd run with GROMOS ff', '1L2Y, 2nd run with OPLS ff']
00030     common_path = '../trp_all_2_compar'
00031     batch_arr.append((filenames_db, legend_names, common_path))
00032
00033     filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_charm_trp_300_fixed.sqlite3', 'results_gromos_trp_300_fixed.sqlite3',
      'results_opls_trp_300_fixed.sqlite3']
00034     # legend_names = ['TRP amber_1', 'TRP charm_1', 'TRP gromos_1', 'TRP opls_1']
00035     legend_names = ['1L2Y, 1st run with AMBER ff', '1L2Y, 1st run with CHARM ff', '1L2Y, 1st run with GROMOS ff', '1L2Y, 1st run with OPLS ff']
00036     common_path = '../trp_all_1_compar'
00037     batch_arr.append((filenames_db, legend_names, common_path))
00038
00039     filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_amber_trp_300_2_fixed.sqlite3', 'results_charm_trp_300_fixed.sqlite3',
      'results_charm_trp_300_2_fixed.sqlite3', 'results_gromos_trp_300_fixed.sqlite3', 'results_gromos_trp_300_2_fixed.sqlite3',
      'results_opls_trp_300_fixed.sqlite3', 'results_opls_trp_300_2_fixed.sqlite3']
00040     legend_names = ['1L2Y, 1st run with AMBER ff', '1L2Y, 2nd run with AMBER ff', '1L2Y, 1st run with CHARM ff', '1L2Y, 2nd run with CHARM ff',
      '1L2Y, 1st run with GROMOS ff', '1L2Y, 2nd run with GROMOS ff', '1L2Y, 1st run with OPLS ff', '1L2Y, 2nd run with OPLS ff']
00041     # legend_names = ['TRP amber_1', 'TRP amber_2', 'TRP charm_1', 'TRP charm_2', 'TRP gromos_1', 'TRP gromos_2', 'TRP opls_1', 'TRP opls_2']
00042     legend_names = ['1L2Y, 1st run with AMBER ff', '1L2Y, 2nd run with AMBER ff', '1L2Y, 1st run with CHARM ff', '1L2Y, 2nd run with CHARM ff',
      '1L2Y, 1st run with GROMOS ff', '1L2Y, 2nd run with GROMOS ff', '1L2Y, 1st run with OPLS ff', '1L2Y, 2nd run with OPLS ff']
00043     common_path = '../trp_all_compar'
00044     batch_arr.append((filenames_db, legend_names, common_path))
00045
00046     # # ##################  VIL  ######################
00047
00048     filenames_db = ['results_amber_vil_300.sqlite3', 'results_charm_vil_300.sqlite3', 'results_gromos_vil_300.sqlite3',
      'results_opls_vil_300.sqlite3']
00049     # legend_names = ['VIL amber', 'VIL charm', 'VIL gromos', 'VIL opls']
00050     legend_names = ['1YRF with AMBER ff', '1YRF with CHARM ff', '1YRF with GROMOS ff', '1YRF with OPLS ff']
00051     common_path = '../vil_all_compar'
00052     batch_arr.append((filenames_db, legend_names, common_path))
00053
00054     # # ##################  GB1  ######################
00055     # #
00056     filenames_db = ['results_amber_gb1_300.sqlite3', 'results_charm_gb1_300.sqlite3', 'results_gromos_gb1_300.sqlite3',
      'results_opls_gb1_300.sqlite3']
00057     # legend_names = ['GB1 amber', 'GB1 charm', 'GB1 gromos', 'GB1 opls']
00058     legend_names = ['1GB1 with AMBER ff', '1GB1 with CHARM ff', '1GB1 with GROMOS ff', '1GB1 with OPLS ff']
00059     common_path = '../gb1_all_compar'
00060     batch_arr.append((filenames_db, legend_names, common_path))
00061
00062
00063     for filenames_db, legend_names, common_path in batch_arr:
00064         gen_all(filenames_db, legend_names, common_path)
00065
00066
00067
00068 def gen_all(filenames_db: list, legend_names: list, common_path: str):
00069     """Takes the tasks and processes them either one by one or in parallel.
00070
00071     Args:
00072         :param list filenames_db: list of databases
00073         :param list legend_names: correct names for DBs
00074         :param str common_path: where to store plots
00075     """
00076     fig_num = 0
00077     try:
00078         os.mkdir(common_path)
00079     except:
00080         pass
00081     # mdpi = 400
00082     #
00083     # font = {'family': 'serif',
00084     #         'color':  'darkred',
00085     #         'weight': 'normal',
00086     #         'size': 12,
00087     #         }
00088     parallel = True  # both work, use parallel to generate everything fast, use debug otherwise
00089     if parallel:
00090         pool = mp.Pool(len(['rmsd', 'angl', 'andh', 'and', 'xor']))  # we are IO bound in graphs, no need to use exact number of CPUs
      mp.cpu_count()
00091         results1 = pool.starmap_async(guide_metr_usage, [(fig_num, filenames_db, legend_names, guide_metr, common_path) for guide_metr in
      ['rmsd', 'angl', 'andh', 'and', 'xor']])
00092         results2 = pool.starmap_async(best_traj, [(fig_num, filenames_db, legend_names, guide_metr, common_path) for guide_metr in ['rmsd',
      'angl', 'andh', 'and', 'xor']])
00093         results1.get()
00094         results2.get()
00095         pool.close()
```

```
00096     else:  # then debug
00097         # for guide_metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00098         #     fig_num = guide_metr_usage(fig_num, filenames_db, legend_names, guide_metr, common_path)
00099 
00100         for guide_metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00101             best_traj(fig_num, filenames_db, legend_names, guide_metr, common_path)
00102 
00103 
00104 def best_traj(fig_num: int, filenames_db: list, legend_names: list, guide_metr: str, common_path: str):
00105     """This is just a basic comparison among metrics
00106 
00107     Args:
00108         :param list fig_num: figure number for matplotlib
00109         :param list filenames_db: databases with data
00110         :param list legend_names: database names
00111         :param str guide_metr:
00112         :param str common_path:
00113 
00114     """
00115     print('Working with ', filenames_db, ' guide metr: ', guide_metr, ' common path: ', common_path)
00116     con_arr = [lite.connect(db_name, check_same_thread=False, isolation_level=None) for db_name in filenames_db]
00117     cur_arr = [con.cursor() for con in con_arr]
00118 
00119     common_path = os.path.join(common_path, guide_metr)
00120     try:
00121         os.mkdir(common_path)
00122     except:
00123         pass
00124     plot_all_best_traj(fig_num, cur_arr, filenames_db, legend_names, guide_metr, common_path)
00125     plot_sep_best_traj(fig_num, cur_arr, filenames_db, legend_names, guide_metr, common_path)
00126 
00127 
00128 def plot_all_best_traj(fig_num: int, cur_arr: list, filenames_db: list, legend_names: list, guide_metr: str, common_path: str) -> int:
00129     """
00130 
00131     Args:
00132         :param int fig_num:
00133         :param list cur_arr:
00134         :param list filenames_db:
00135         :param list legend_names:
00136         :param str guide_metr:
00137         :param str common_path:
00138 
00139     Returns:
00140         :return: figure number
00141         :rtype: int
00142     """
00143     print('Working with ', filenames_db, ' guide metr: ', guide_metr, ' common path: ', common_path)
00144     qry = "select a.name from main_storage a where a.{0}_goal_dist= ( select min(b.{0}_goal_dist) from main_storage b)".format(guide_metr)
00145     result_arr = [cur.execute(qry) for cur in cur_arr]
00146     fetched_one_arr = [res.fetchone() for res in result_arr]
00147     names = [all_res[0] for all_res in fetched_one_arr]
00148     spnames = [name.split('_') for name in names]
00149     all_prev_names_s = [['\'{}\"'.format('_'.join(spname[:i])) for i in range(1, len(spname)+1)] for spname in spnames]
00150     long_lines = ["', "'.join(all_prev_names) for all_prev_names in all_prev_names_s]
00151     qrys = ["select a.rmsd_goal_dist, a.angl_goal_dist, a.andh_goal_dist, a.and_goal_dist, a.xor_goal_dist, a.rmsd_tot_dist, a.angl_tot_dist,
        a.andh_tot_dist, a.and_tot_dist, a.xor_tot_dist, a.name, a.hashed_name from main_storage a where a.name in ( {1} ) order by
        a.id".format(guide_metr, long_line) for long_line in long_lines]
00152     result_arr = list()
00153     for i, cur in enumerate(cur_arr):
00154         result_arr.append(cur.execute(qrys[i]))
00155     fetched_all_arr = [res.fetchall() for res in result_arr]
00156 
00157     rmsd_dist_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00158     angl_dist_arr = [[dist[1] for dist in goal_dist] for goal_dist in fetched_all_arr]
00159     andh_dist_arr = [[dist[2] for dist in goal_dist] for goal_dist in fetched_all_arr]
00160     and_dist_arr = [[dist[3] for dist in goal_dist] for goal_dist in fetched_all_arr]
00161     xor_dist_arr = [[dist[4] for dist in goal_dist] for goal_dist in fetched_all_arr]
00162 
00163 
00164     rmsd_tot_dist_arr = [[dist[5] for dist in goal_dist] for goal_dist in fetched_all_arr]
00165     angl_tot_dist_arr = [[dist[6] for dist in goal_dist] for goal_dist in fetched_all_arr]
00166     andh_tot_dist_arr = [[dist[7] for dist in goal_dist] for goal_dist in fetched_all_arr]
00167     and_tot_dist_arr = [[dist[8] for dist in goal_dist] for goal_dist in fetched_all_arr]
00168     xor_tot_dist_arr = [[dist[9] for dist in goal_dist] for goal_dist in fetched_all_arr]
00169 
00170     goal_dist = [rmsd_dist_arr, angl_dist_arr, andh_dist_arr, and_dist_arr, xor_dist_arr]
00171     tot_dist = [rmsd_tot_dist_arr, angl_tot_dist_arr, andh_tot_dist_arr, and_tot_dist_arr, xor_tot_dist_arr]
00172     metrics = ['rmsd', 'angl', 'andh', 'and', 'xor']
00173     metr_units = {'rmsd': 'Å', 'angl': '', 'andh': 'contacts', 'and': 'contacts', 'xor': 'contacts'}
00174 
```

```
00175
00176
00177     for i, dist_arr in enumerate(goal_dist):  # iterate over metric
00178         max_len = max([len(arr) for arr in dist_arr])
00179         max_pos_metr_val = max([max(arr) for arr in dist_arr])
00180         init_metr = dist_arr[0][0]
00181
00182         ax_prop = {"min_lim_x": 0 - max_len / 80, "max_lim_x": max_len + max_len / 80, "min_lim_y": 0 - max_pos_metr_val / 80, "max_lim_y":
    max_pos_metr_val + max_pos_metr_val / 80, "min_ax_x": 0,
00183                    "max_ax_x": max_len + max_len / 80, "min_ax_y": 0, "max_ax_y": max_pos_metr_val + max_pos_metr_val / 80, "ax_step_x":
    math.floor(max_len / 16), "ax_step_y": max_pos_metr_val / 20}
00184         if metr_units[metrics[i]] == 'contacts':
00185             extra_line = [{"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({} {})".format(metrics[i].upper(), int(init_metr),
    metr_units[metrics[i]]), "col": "darkmagenta"}]
00186         else:
00187             extra_line = [{"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metrics[i].upper(), init_metr,
    metr_units[metrics[i]]), "col": "darkmagenta"}]
00188         if metrics[i] == 'rmsd':
00189             extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units[metrics[i]]), "col":
    "midnightblue"})
00190         title = "{} version of the best trajectory | {} view".format(guide_metr, metrics[i])
00191         filename = "{}_version_of_best_traj_{}".format(guide_metr, metrics[i])
00192         filename = os.path.join(common_path, filename)
00193         fig_num = single_plot(fig_num, ax_prop, dist_arr, None, legend_names.copy(), '-', 1, bsf=False, rev=False, extra_line=extra_line,
    shrink=True, xlab="Steps (20ps each)", ylab="Distance to the goal, {}".format(metr_units[metrics[i]]), title=title, filename=filename)
00194
00195         max_tot_dist = max([dist[-1] for dist in tot_dist[i]])
00196         ax_prop = {"min_lim_x": max_pos_metr_val + max_pos_metr_val / 80, "max_lim_x": 0 - max_pos_metr_val / 80, "min_lim_y": 0 - max_tot_dist
    / 80, "max_lim_y": max_tot_dist + max_tot_dist / 80, "min_ax_x": 0, "max_ax_x": max_pos_metr_val + max_pos_metr_val / 80, "min_ax_y": 0,
    "max_ax_y": max_tot_dist + max_tot_dist / 80, "ax_step_x": max_pos_metr_val / 20, "ax_step_y": max_tot_dist / 20}
00197         if metr_units[metrics[i]] == 'contacts':
00198             extra_line = [{"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({} {})".format(metrics[i].upper(), int(init_metr),
    metr_units[metrics[i]]), "col": "darkmagenta"}]
00199         else:
00200             extra_line = [{"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metrics[i].upper(), init_metr,
    metr_units[metrics[i]]), "col": "darkmagenta"}]
00201         if metrics[i] == 'rmsd':
00202             extra_line.append({"ax_type": 'ver', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units[metrics[i]]), "col":
    "midnightblue"})
00203         title = "{} version of the best trajectory vs distance traveled | {} view".format(guide_metr, metrics[i])
00204         filename = '{}_version_of_best_traj_{}_vs_dist'.format(guide_metr, metrics[i])
00205         filename = os.path.join(common_path, filename)
00206         fig_num = single_plot(fig_num, ax_prop, dist_arr, tot_dist[i], legend_names.copy(), '-', 1, bsf=False, rev=True, extra_line=extra_line,
    shrink=False, xlab="Distance to the goal, {}".format(metr_units[metrics[i]]), ylab="Past distance, {}".format(metr_units[metrics[i]]),
    title=title, filename=filename)
00207
00208         for j in range(len(dist_arr)):  # iterate over dbs
00209             max_pos_metr_val = max(dist_arr[j])
00210             ax_prop = {"min_lim_x": 0 - max_len / 80, "max_lim_x": max_len + max_len / 80, "min_lim_y": 0, "max_lim_y": max_pos_metr_val +
    max_pos_metr_val / 80, "min_ax_x": 0,
00211                        "max_ax_x": max_len + max_len / 80, "min_ax_y": 0, "max_ax_y": max_pos_metr_val + max_pos_metr_val / 80, "ax_step_x":
    max_len / 16, "ax_step_y": max_pos_metr_val / 20}
00212             if metr_units[metrics[i]] == 'contacts':
00213                 extra_line = [{"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({} {})".format(metrics[i].upper(),
    int(init_metr), metr_units[metrics[i]]), "col": "darkmagenta"},
00214                               {"ax_type": 'hor', "val": min(dist_arr[j]), "name": "The lowest {} metric ({} {})".format(metrics[i].upper(),
    int(min(dist_arr[j])), metr_units[metrics[i]]), "col": "darkgreen"}]
00215             else:
00216                 extra_line = [{"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metrics[i].upper(),
    init_metr, metr_units[metrics[i]]), "col": "darkmagenta"},
00217                               {"ax_type": 'hor', "val": min(dist_arr[j]), "name": "The lowest {} metric ({:3.2f}
    {})".format(metrics[i].upper(), min(dist_arr[j]), metr_units[metrics[i]]), "col": "darkgreen"}]
00218
00219             if metrics[i] == 'rmsd':
00220                 extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units[metrics[i]]), "col":
    "midnightblue"})
00221             title = "{} version of the best trajectory | {} view".format(guide_metr, metrics[i])
00222             filename = "{}_version_of_best_traj_{}_only_{}".format(guide_metr, metrics[i], filenames_db[j].split('.')[0])
00223             filename = os.path.join(common_path, filename)
00224             fig_num = single_plot(fig_num, ax_prop, [dist_arr[j]], None, [legend_names[j]], '-', 1, bsf=False, rev=False,
    extra_line=extra_line, shrink=True, xlab="Steps (20ps each)", ylab="Distance to the goal, {}".format(metr_units[metrics[i]]), title=title,
    filename=filename)
00225
00226             max_tot_dist = max([dist[-1] for dist in [tot_dist[i][j]]])
00227             ax_prop = {"min_lim_x": max_pos_metr_val + max_pos_metr_val / 80, "max_lim_x": 0 - max_pos_metr_val / 80, "min_lim_y": 0 -
    max_tot_dist / 80, "max_lim_y": max_tot_dist + max_tot_dist / 80, "min_ax_x": 0,
00228                        "max_ax_x": max_pos_metr_val + max_pos_metr_val / 80, "min_ax_y": 0, "max_ax_y": max_tot_dist + max_tot_dist / 80,
    "ax_step_x": max_pos_metr_val / 20, "ax_step_y": max_tot_dist / 20}
00229             if metr_units[metrics[i]] == 'contacts':
00230                 extra_line = [{"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({} {})".format(metrics[i].upper(),
    int(init_metr), metr_units[metrics[i]]), "col": "darkmagenta"},
```

```
00231                           {"ax_type": 'ver', "val": min(dist_arr[j]), "name": "The lowest {} metric ({} {})".format(metrics[i].upper(),
    int(min(dist_arr[j])), metr_units[metrics[i]]), "col": "darkgreen"}]
00232           else:
00233               extra_line = [{"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metrics[i].upper(),
    init_metr, metr_units[metrics[i]]), "col": "darkmagenta"},
00234                           {"ax_type": 'ver', "val": min(dist_arr[j]), "name": "The lowest {} metric ({:3.2f}
    {})".format(metrics[i].upper(), min(dist_arr[j]), metr_units[metrics[i]]), "col": "darkgreen"}]
00235           if metrics[i] == 'rmsd':
00236               extra_line.append({"ax_type": 'ver', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units[metrics[i]]), "col":
    "midnightblue"})
00237           title = "{} version of the best trajectory vs distance traveled | {} view".format(guide_metr, metrics[i])
00238           filename = '{}_version_of_best_traj_{}_vs_dist_only_{}'.format(guide_metr, metrics[i], filenames_db[j].split('.')[0])
00239           filename = os.path.join(common_path, filename)
00240           fig_num = single_plot(fig_num, ax_prop, [dist_arr[j]], [tot_dist[i][j]], [legend_names[j]], '-', 1, bsf=False, rev=True,
    extra_line=extra_line, shrink=False, xlab="Distance to the goal, {}".format(metr_units[metrics[i]]), ylab="Past distance,
    {}".format(metr_units[metrics[i]]), title=title, filename=filename)
00241
00242           max_pos_metr_val = dist_arr[j][0]
00243           min_pos_metr_val = dist_arr[j][-1]
00244           if min_pos_metr_val > max_pos_metr_val:
00245               min_pos_metr_val, max_pos_metr_val = max_pos_metr_val, min_pos_metr_val
00246
00247
00248           loc_len = len(dist_arr[j])
00249           for k in range(len(goal_dist)):
00250               if i != k:
00251                   max_pos_metr2_val = goal_dist[k][j][0]
00252                   min_pos_metr2_val = goal_dist[k][j][-1]
00253                   if max_pos_metr2_val < min_pos_metr2_val:
00254                       max_pos_metr2_val, min_pos_metr2_val = min_pos_metr2_val, max_pos_metr2_val
00255
00256                   divider_min = 15.0
00257                   divider_max = 10.0
00258
00259                   while divider_min > 0.1:
00260                       if (min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / divider_min) < min(goal_dist[k][j]) and
    min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) / divider_min < min(
00261                               dist_arr[j]):
00262                           break
00263                       divider_min -= 0.05
00264
00265                   while divider_max > 0.1:
00266                       if (max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max) > max(goal_dist[k][j]) and
    max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max > max(
00267                               dist_arr[j]):
00268                           break
00269                       divider_max -= 0.05
00270
00271                   ax_prop = {"min_lim_x": 0 - loc_len / 80, "max_lim_x": loc_len + loc_len / 80, "min_lim_y": min_pos_metr_val -
    (max_pos_metr_val - min_pos_metr_val) / divider_min,
00272                              "max_lim_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max, "min_ax_x": 0,
00273                              "max_ax_x": loc_len + loc_len / 80, "min_ax_y": min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) /
    divider_min, "max_ax_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max,
00274                              "ax_step_x": math.floor(loc_len / 16), "ax_step_y": (max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) /
    divider_max - min_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_min) / 20}
00275                   ax2_prop = {"min_lim_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / divider_min, "max_lim_y":
    max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max,
00276                               "min_ax_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / divider_min, "max_ax_y":
    max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max,  "ax_step_y": (max_pos_metr2_val + (max_pos_metr2_val -
    min_pos_metr2_val) / divider_max - min_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_min) / 20,
00277                               "label": "Distance to the goal ({}), {}".format(metrics[k].upper(), metr_units[metrics[k]]), "line_name": '{}
    ({})'.format(legend_names[j], metrics[k].upper())}
00278                   if metr_units[metrics[i]] == 'contacts':
00279                       extra_line = [
00280                           {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({} {})".format(metrics[i].upper(), int(init_metr),
    metr_units[metrics[i]]), "col": "darkmagenta"},
00281                           {"ax_type": 'hor', "val": min(dist_arr[j]), "name": "The lowest {} metric ({} {})".format(metrics[i].upper(),
    int(min(dist_arr[j])), metr_units[metrics[i]]), "col": "darkgreen"}]
00282                   else:
00283                       extra_line = [
00284                           {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metrics[i].upper(), init_metr,
    metr_units[metrics[i]]), "col": "darkmagenta"},
00285                           {"ax_type": 'hor', "val": min(dist_arr[j]), "name": "The lowest {} metric ({:3.2f} {})".format(metrics[i].upper(),
    min(dist_arr[j]), metr_units[metrics[i]]), "col": "darkgreen"}]
00286                   if metrics[i] == 'rmsd':
00287                       extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7
    {})".format(metr_units[metrics[i]]), "col": "midnightblue"})
00288                   title = "{} version of the best trajectory | {} view vs {} view".format(guide_metr, metrics[i], metrics[k])
00289                   filename = "{}_version_of_best_traj_{}_only_{}_vs_{}".format(guide_metr, metrics[i], filenames_db[j].split('.')[0],
    metrics[k])
00290                   filename = os.path.join(common_path, filename)
```

```
00291                     try:
00292                         fig_num = single_plot(fig_num, ax_prop, [dist_arr[j]], None, ['{} ({})'.format(legend_names[j], metrics[i].upper())],
'-', 1, bsf=False, rev=False, extra_line=extra_line, shrink=True, xlab="Steps (20ps each)",
00293                                               ylab="Distance to the goal ({}), {}".format(metrics[i].upper(), metr_units[metrics[i]]), title=title,
filename=filename, second_ax=ax2_prop, sec_arr=goal_dist[k][j])
00294                     except Exception as e:
00295                         print('Error in generation of {}'.format(filename))
00296
00297             loc_len = len(dist_arr[j])
00298             # prot_name, ff = legend_names[j].split(' ')
00299             if 'AMBER' in legend_names[j].upper():
00300                 ff = 'amber'
00301             elif 'CHARM' in legend_names[j].upper():
00302                 ff = 'charm'
00303             elif 'GROMOS' in legend_names[j].upper():
00304                 ff = 'gromos'
00305             elif 'OPLS' in legend_names[j].upper():
00306                 ff = 'opls'
00307
00308             if 'TRP' in legend_names[j].upper() or '1L2Y' in legend_names[j].upper():
00309                 prot_name = 'TRP'
00310             elif 'VIL' in legend_names[j].upper() or '1YRF' in legend_names[j].upper():
00311                 prot_name = 'VIL'
00312             elif 'GB1' in legend_names[j].upper():
00313                 prot_name = 'GB1'
00314
00315             if '2ND' in legend_names[j].upper():
00316                 rn = 2
00317             elif '1ST' in legend_names[j].upper():
00318                 rn = 1
00319             else:
00320                 rn = None
00321             # if '_' in ff:
00322             #     ff, rn = ff.split('_')
00323             path_to_ener = "/home/vanya/Documents/Phillips/GMDA/Latest_results"
00324             path_to_ener1 = os.path.join(path_to_ener, prot_name)
00325             if rn is not None:
00326                 path_to_ener1 = os.path.join(path_to_ener1, "run_{}".format(rn))
00327             # path_to_ener2 = os.path.join(path_to_ener1, ff, 'LJ_energy')
00328             # np_ener_file = os.path.join(path_to_ener2, '{}_combined_energy_best_full_step.npy'.format(guide_metr))
00329             # ener_arr = np.load(np_ener_file).swapaxes(0, 1)[1]
00330             # ener_arr = ener_arr[-loc_len:]  # trim, so we have same number of steps
00331             # if len(ener_arr) != loc_len:
00332             #     print('kva')
00333             #
00334             # max_pos_metr2_val = ener_arr[0]
00335             # min_pos_metr2_val = ener_arr[-1]
00336             #
00337             # ax_prop = {"min_lim_x": 0 - loc_len / 80, "max_lim_x": loc_len + loc_len / 80, "min_lim_y": min_pos_metr_val - (max_pos_metr_val
- min_pos_metr_val) / 5.0,
00338             #            "max_lim_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / 10, "min_ax_x": 0,
00339             #            "max_ax_x": loc_len + loc_len / 80, "min_ax_y": min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) / 5.0,
00340             #            "max_ax_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / 10,
00341             #            "ax_step_x": loc_len / 16, "ax_step_y": (max_pos_metr_val - min_pos_metr_val) / 20}
00342             # ax2_prop = {"min_lim_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / 5.0, "max_lim_y": max_pos_metr2_val +
(max_pos_metr2_val - min_pos_metr2_val) / 10,
00343             #             "min_ax_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / 5.0, "max_ax_y": max_pos_metr2_val +
(max_pos_metr2_val - min_pos_metr2_val) / 10,
00344             #             "ax_step_y": (max_pos_metr2_val - min_pos_metr2_val) / 20,
00345             #             "label": "LJ energy, {}".format('kJ/mol'), "line_name": 'LJ:SR interaction energy ({})'.format('kJ/mol')}
00346             # extra_line = [{"ax_type": 'hor', "val": init_metr, "name": "initial {} metric ({:3.2f} {})".format(metrics[i], init_metr,
metr_units[metrics[i]]), "col": "darkmagenta"}]
00347             # if metrics[i] == 'rmsd':
00348             #     extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "typical folding mark (2.7 {})".format(metr_units[metrics[i]]),
"col": "midnightblue"})
00349             # title = "{} version of the best trajectory | {} view vs LJ:SR view".format(guide_metr, metrics[i])
00350             # filename = "{}_version_of_best_traj_{}_only_{}_vs_{}".format(guide_metr, metrics[i], filenames_db[j].split('.')[0], 'lj_energy')
00351             # filename = os.path.join(common_path, filename)
00352             # fig_num = single_plot(fig_num, ax_prop, [dist_arr[j]], None, ['{} ({})'.format(legend_names[j], metrics[i])], '-', 1, bsf=False,
rev=False, extra_line=extra_line, shrink=True,
00353             #                       xlab="steps (20ps each)",
00354             #                       ylab="to goal ({}), {}".format(metrics[i], metr_units[metrics[i]]), title=title, filename=filename,
second_ax=ax2_prop, sec_arr=ener_arr)
00355             #
00356             #
00357             # path_to_ener2 = os.path.join(path_to_ener1, ff, 'CL_energy')
00358             # np_ener_file = os.path.join(path_to_ener2, '{}_combined_energy_best_full_step.npy'.format(guide_metr))
00359             # ener_arr = np.load(np_ener_file).swapaxes(0, 1)[1]
00360             # ener_arr = ener_arr[-loc_len:]  # trim, so we have same number of steps
00361             #
00362             # max_pos_metr2_val = ener_arr[0]
```

```
00363             # min_pos_metr2_val = ener_arr[-1]
00364             #
00365             # ax_prop = {"min_lim_x": 0 - loc_len / 80, "max_lim_x": loc_len + loc_len / 80, "min_lim_y": min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) / 5.0,
00366             #            "max_lim_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / 10, "min_ax_x": 0,
00367             #            "max_ax_x": loc_len + loc_len / 80, "min_ax_y": min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) / 5.0,
00368             #            "max_ax_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / 10,
00369             #            "ax_step_x": loc_len / 16, "ax_step_y": (max_pos_metr_val - min_pos_metr_val) / 20}
00370             # ax2_prop = {"min_lim_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / 5.0, "max_lim_y": max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / 10,
00371             #             "min_ax_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / 5.0, "max_ax_y": max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / 10,
00372             #             "ax_step_y": (max_pos_metr2_val - min_pos_metr2_val) / 20,
00373             #             "label": "CL energy, {}".format('kJ/mol'), "line_name": 'CL:SR interaction energy ({})'.format('kJ/mol')}
00374             # extra_line = [{"ax_type": 'hor', "val": init_metr, "name": "initial {} metric ({:3.2f} {})".format(metrics[i], init_metr, metr_units[metrics[i]]), "col": "darkmagenta"}]
00375             # if metrics[i] == 'rmsd':
00376             #     extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "typical folding mark (2.7 {})".format(metr_units[metrics[i]]), "col": "midnightblue"})
00377             # title = "{} version of the best trajectory | {} view vs CL:SR view".format(guide_metr, metrics[i])
00378             # filename = "{}_version_of_best_traj_{}_only_{}_vs_{}".format(guide_metr, metrics[i], filenames_db[j].split('.')[0], 'cl_energy')
00379             # filename = os.path.join(common_path, filename)
00380             # fig_num = single_plot(fig_num, ax_prop, [dist_arr[j]], None, ['{} ({})'.format(legend_names[j], metrics[i])], '-', 1, bsf=False, rev=False, extra_line=extra_line, shrink=True,
00381             #                       xlab="steps (20ps each)",
00382             #                       ylab="to goal ({}), {}".format(metrics[i], metr_units[metrics[i]]), title=title, filename=filename, second_ax=ax2_prop, sec_arr=ener_arr)
00383
00384
00385
00386
00387             path_to_ener2 = os.path.join(path_to_ener1, ff, 'PT_energy')
00388             np_ener_file = os.path.join(path_to_ener2, '{}_correct_index_energy.npy'.format(guide_metr))
00389             ener_arr = np.load(np_ener_file).swapaxes(0, 1)[1]
00390             ener_arr = ener_arr[-loc_len:]  # trim, so we have same number of steps
00391
00392             max_pos_metr2_val = ener_arr[0]
00393             min_pos_metr2_val = ener_arr[-1]
00394
00395             divider_min = 5.0
00396             divider_max = 10.0
00397
00398             while divider_min > 0.1:
00399                 if (min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / divider_min) < min(ener_arr) and min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) / divider_min < min(
00400                         dist_arr[j]):
00401                     break
00402                 divider_min -= 0.05
00403
00404             while divider_max > 0.1:
00405                 if (max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max) > max(ener_arr) and max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max > max(
00406                         dist_arr[j]):
00407                     break
00408                 divider_max -= 0.05
00409
00410             ax_prop = {"min_lim_x": 0 - loc_len / 80, "max_lim_x": loc_len + loc_len / 80, "min_lim_y": min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) / divider_min,
00411                        "max_lim_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max, "min_ax_x": 0,
00412                        "max_ax_x": loc_len + loc_len / 80, "min_ax_y": min_pos_metr_val - (max_pos_metr_val - min_pos_metr_val) / divider_min,
00413                        "max_ax_y": max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max,
00414                        "ax_step_x": math.floor(loc_len / 16), "ax_step_y": (max_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_max - min_pos_metr_val + (max_pos_metr_val - min_pos_metr_val) / divider_min) / 20}
00415             ax2_prop = {"min_lim_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / divider_min, "max_lim_y": max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max,
00416                         "min_ax_y": min_pos_metr2_val - (max_pos_metr2_val - min_pos_metr2_val) / divider_min, "max_ax_y": max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max,
00417                         "ax_step_y": (max_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_max - min_pos_metr2_val + (max_pos_metr2_val - min_pos_metr2_val) / divider_min) / 20,
00418                         "label": "Potential energy, {}".format('kJ/mol'), "line_name": 'Potential energy ({})'.format('kJ/mol')}
00419             if metr_units[metrics[i]] == 'contacts':
00420                 extra_line = [
00421                     {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({} {})".format(metrics[i].upper(), int(init_metr), metr_units[metrics[i]]), "col": "darkmagenta"},
00422                     {"ax_type": 'hor', "val": min(dist_arr[j]), "name": "The lowest {} metric ({} {})".format(metrics[i].upper(), int(min(dist_arr[j])), metr_units[metrics[i]]), "col": "darkgreen"}]
00423             else:
00424                 extra_line = [
00425                     {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metrics[i].upper(), init_metr, metr_units[metrics[i]]), "col": "darkmagenta"},
```

```
00426                   {"ax_type": 'hor', "val": min(dist_arr[j]), "name": "The lowest {} metric ({:3.2f} {})".format(metrics[i].upper(),
      min(dist_arr[j]), metr_units[metrics[i]]), "col": "darkgreen"}]
00427             if metrics[i] == 'rmsd':
00428                 extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units[metrics[i]]), "col":
      "midnightblue"})
00429             title = "{} version of the best trajectory | {} view vs Potential energy view".format(guide_metr, metrics[i])
00430             filename = "{}_version_of_best_traj_{}_only_{}_vs_{}".format(guide_metr, metrics[i], filenames_db[j].split('.')[0], 'pt_energy')
00431             filename = os.path.join(common_path, filename)
00432             fig_num = single_plot(fig_num, ax_prop, [dist_arr[j]], None, ['{} ({})'.format(legend_names[j], metrics[i].upper())], '-', 1,
      bsf=False, rev=False, extra_line=extra_line, shrink=True,
00433                                   xlab="Steps (20ps each)",
00434                                   ylab="Distance to the goal ({}), {}".format(metrics[i].upper(), metr_units[metrics[i]]), title=title,
      filename=filename, second_ax=ax2_prop, sec_arr=ener_arr)
00435
00436
00437
00438     # max_len = max([len(arr) for arr in rmsd_dist_arr])
00439     # init_metr = rmsd_dist_arr[0][0]
00440     # metr_units = 'A'
00441     # ax_prop = {"min_lim_x": 0 - +max_len/80, "max_lim_x": max_len + max_len/80, "min_lim_y": 0 - init_metr/80, "max_lim_y": init_metr +
      init_metr/80, "min_ax_x": 0, "max_ax_x": max_len + max_len/80, "min_ax_y": 0, "max_ax_y": init_metr+init_metr/80, "ax_step_x": max_len / 16,
      "ax_step_y": init_metr / 20}
00442     # extra_line = {"ax_type": 'hor', "val": init_metr, "name": "initial {} metric ({:3.2f} {})".format('rmsd', init_metr, metr_units)}
00443     # # title = "{} | to goal vs traveled | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00444     # # filename = "{}_to_goal_vs_traveled_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00445     # # filename = os.path.join(custom_path, filename)
00446     # title = 'kva'
00447     # filename = 'test_best'
00448     # fig_num = single_plot(fig_num, ax_prop, rmsd_dist_arr, None, legend_names.copy(), '-', 1, bsf=False, rev=False, extra_line=extra_line,
      shrink=True, xlab="steps (20ps each)", ylab="to goal, {}".format(metr_units), title=title, filename=filename)
00449     #
00450     # max_tot_dist = max([dist[-1] for dist in rmsd_tot_dist_arr])
00451     # # ax_prop = {"min_lim_x": 0 - +max_len/80, "max_lim_x": max_tot_dist + max_tot_dist/80, "min_lim_y": 0 - init_metr/80, "max_lim_y":
      init_metr + init_metr/80, "min_ax_x": 0, "max_ax_x": max_tot_dist + max_tot_dist/80, "min_ax_y": 0, "max_ax_y": init_metr+init_metr/80,
      "ax_step_x": max_tot_dist / 16, "ax_step_y": init_metr / 20}
00452     # ax_prop = {"min_lim_x": init_metr + init_metr / 80, "max_lim_x": 0 - init_metr / 80, "min_lim_y": 0 - +max_len / 80, "max_lim_y":
      max_tot_dist + max_tot_dist / 80, "min_ax_x": 0,
00453     #            "max_ax_x": init_metr + init_metr / 80, "min_ax_y": 0, "max_ax_y": max_tot_dist + max_tot_dist / 80, "ax_step_x": init_metr /
      20, "ax_step_y": max_tot_dist / 16}
00454     # extra_line = {"ax_type": 'ver', "val": init_metr, "name": "initial {} metric ({:3.2f} {})".format('rmsd', init_metr, metr_units)}
00455     # # title = "{} | to goal vs traveled | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00456     # # filename = "{}_to_goal_vs_traveled_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00457     # # filename = os.path.join(custom_path, filename)
00458     # title = 'kva'
00459     # filename = 'test_best'
00460     # fig_num = single_plot(fig_num, ax_prop, rmsd_dist_arr, rmsd_tot_dist_arr, legend_names.copy(), '-', 1, bsf=False, rev=True,
      extra_line=extra_line, shrink=False, xlab="to goal, {}".format(metr_units), ylab="steps (20ps each)", title=title, filename=filename)
00461
00462
00463
00464
00465
00466 def plot_sep_best_traj(fig_num, cur_arr, filenames_db, legend_names, guide_metr, common_path):
00467     pass
00468
00469
00470 def guide_metr_usage(fig_num: int, filenames_db: list, legend_names: list, guide_metr: str, common_path: str) -> int:
00471     """
00472
00473     Args:
00474         :param int fig_num: figure number, it should not matter, since we close all figures regularly
00475         :param list filenames_db: database names
00476         :param list legend_names: proper database description
00477         :param str guide_metr: main metric for the plot
00478         :param str common_path: where to store plots
00479
00480     Returns:
00481         :return: figure number, it should not matter, since we close all figures regularly
00482     """
00483
00484     con_arr = [lite.connect(db_name, check_same_thread=False, isolation_level=None) for db_name in filenames_db]
00485     cur_arr = [con.cursor() for con in con_arr]
00486
00487     common_path = os.path.join(common_path, guide_metr)
00488     try:
00489         os.mkdir(common_path)
00490     except:
00491         pass
00492
00493     fig_num, init_rmsd = plot_all_metrics(fig_num, cur_arr, filenames_db, legend_names, guide_metr, common_path)
00494
```

```
00495     for partial_metr in ["RMSD", "ANGL", "AND_H", "AND", "XOR"]:
00496         pers_path = os.path.join(common_path, partial_metr)
00497         try:
00498             os.mkdir(pers_path)
00499         except:
00500             pass
00501         fig_num = plot_only_one_metric(fig_num, cur_arr, filenames_db, init_rmsd, legend_names, partial_metr, guide_metr, pers_path)
00502 
00503     [con.close() for con in con_arr]
00504     return fig_num
00505 
00506 
00507 def plot_all_metrics(fig_num: int, cur_arr: list, filenames_db: list, legend_names: list, guide_metr: str, common_path: str) -> int:
00508     """General force field comparison: sampling, best_so_far, dist traveled
00509 
00510     Args:
00511         :param int fig_num: figure number, it should not matter, since we close all figures regularly
00512         :param list cur_arr:
00513         :param list filenames_db:
00514         :param list legend_names:
00515         :param str guide_metr:
00516         :param str common_path:
00517 
00518     Returns:
00519         :return: figure number, it should not matter, since we close all figures regularly
00520     """
00521     best_metr_dic = {'rmsd': 'bsfr', 'angl': 'bsfn', 'andh': 'bsfh', 'and': 'bsfa', 'xor': 'bsfx'}
00522     metr_units = {'rmsd': 'Å', 'angl': '', 'andh': 'contacts', 'and': 'contacts', 'xor': 'contacts'}
00523     qry = 'SELECT a.{}_goal_dist FROM main_storage a join visited b on a.id=b.id order by b.vid'.format(guide_metr)
00524     result_arr = [cur.execute(qry) for cur in cur_arr]
00525     fetched_all_arr = [res.fetchall() for res in result_arr]
00526     filt_res_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00527     init_rmsd = filt_res_arr[0][0]
00528     max_non_init_rmsd = max(max(elem) for elem in filt_res_arr)
00529     common_point = max([min(elem) for elem in filt_res_arr])
00530 
00531     ind_arr = list()
00532     for rmsd_for_db in filt_res_arr:
00533         i = 0
00534         while common_point < rmsd_for_db[i]:
00535             i += 1
00536         ind_arr.append(i)
00537 
00538     # print('To reach common min point of {}A ({})'.format(common_point, guide_metr))
00539     # for i, db in enumerate(filenames_db):
00540     #     print('{} : {} steps'.format(db.split('.')[0], ind_arr[i]))
00541 
00542 
00543 
00544     #     ################ CUT ######################
00545 
00546     # qry = "select a.bsfr, b.rmsd_tot_dist, b.rmsd_goal_dist from log a join main_storage b on a.id=b.id where a.dst='VIZ' and a.bsfr>'{}' order by a.lid".format(common_point)
00547     qry = "select a.{0}, b.{1}_tot_dist, b.{1}_goal_dist, c.vid from main_storage b join visited c on c.id=b.id join (select id, {0} from log where dst='VIZ' group by id) a on a.id=b.id where a.{0}>'{2}'  order by c.vid".format(best_metr_dic[guide_metr], guide_metr, common_point)
00548     result_arr = [cur.execute(qry) for cur in cur_arr]
00549     [res.fetchone() for res in result_arr]
00550     fetched_all_arr = [res.fetchall() for res in result_arr]
00551     bsf_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00552     for i in range(len(bsf_arr)):
00553         bsf_arr[i].insert(0, init_rmsd)
00554     for j in range(len(bsf_arr)):
00555         for i in range(len(bsf_arr[j]) - 1):
00556             if bsf_arr[j][i] < bsf_arr[j][i + 1]:
00557                 bsf_arr[j][i+1] = bsf_arr[j][i]
00558     trav_arr = [[dist[1] for dist in goal_dist] for goal_dist in fetched_all_arr]
00559     to_goal_arr = [[dist[2] for dist in goal_dist] for goal_dist in fetched_all_arr]
00560 
00561     max_len = max([len(goal_dist) for goal_dist in fetched_all_arr])
00562     custom_path = '{}/ALL/'.format(common_path)
00563     try:
00564         os.mkdir(custom_path)
00565     except:
00566         pass
00567 
00568     try:
00569         max_trav = max([max(elem) for elem in trav_arr])
00570         custom_path = '{}/ALL/cut/'.format(common_path)
00571         try:
00572             os.mkdir(custom_path)
00573         except:
```

```
00574             pass
00575         # shrink is True since everything is in order, there is no difference whether to pass index or generate it
00576         fig_num = plot_set(fig_num, to_goal_arr, legend_names, max_len, max_non_init_rmsd, init_rmsd, bsf_arr, common_point, max_trav,
      trav_arr, "cut", guide_metr, metr_units[guide_metr], 'all', custom_path, shrink=True)
00577     except:
00578         print('Not all trajecotories have a common point', [len(elem) for elem in trav_arr])
00579
00580     #     ################ FULL ####################
00581
00582     # qry = "select a.bsfr, b.rmsd_tot_dist, b.rmsd_goal_dist from log a join main_storage b on a.id=b.id where a.dst='VIZ' order by a.lid"
00583     qry = "select a.{0}, b.{1}_tot_dist, b.{1}_goal_dist, c.vid from main_storage b join visited c on c.id=b.id join (select id, max({0}) as
      {0} from log where dst='VIZ' group by id) a on a.id=b.id order by c.vid".format(best_metr_dic[guide_metr], guide_metr)
00584     result_arr = [cur.execute(qry) for cur in cur_arr]
00585     [res.fetchone() for res in result_arr]
00586     fetched_all_arr = [res.fetchall() for res in result_arr]
00587     bsf_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00588     for i in range(len(bsf_arr)):
00589         bsf_arr[i].insert(0, init_rmsd)
00590     for j in range(len(bsf_arr)):
00591         for i in range(len(bsf_arr[j]) - 1):
00592             if bsf_arr[j][i] < bsf_arr[j][i + 1]:
00593                 bsf_arr[j][i+1] = bsf_arr[j][i]
00594
00595     trav_arr = [[dist[1] for dist in goal_dist] for goal_dist in fetched_all_arr]
00596     to_goal_arr = [[dist[2] for dist in goal_dist] for goal_dist in fetched_all_arr]
00597
00598     max_len = max([len(goal_dist) for goal_dist in fetched_all_arr])
00599     max_trav = max([max(elem) for elem in trav_arr])
00600     common_point = min([min(elem) for elem in filt_res_arr])
00601
00602     custom_path = '{}/ALL/full/'.format(common_path)
00603     try:
00604         os.mkdir(custom_path)
00605     except:
00606         pass
00607     # shrink is True since everything is in order, there is no difference whether to pass index or generate it
00608     fig_num = plot_set(fig_num, to_goal_arr, legend_names, max_len, max_non_init_rmsd, init_rmsd, bsf_arr, common_point, max_trav, trav_arr,
      "full", guide_metr, metr_units[guide_metr], 'all', custom_path, shrink=True)
00609
00610
00611     return fig_num, init_rmsd
00612
00613
00614 def plot_only_one_metric(fig_num: int, cur_arr: list, filenames_db: list, init_rmsd: float, legend_names: list, metric_name: str, guide_metr:
      str, common_path: str) -> int:
00615     """
00616
00617     Args:
00618         :param int fig_num:
00619         :param list cur_arr:
00620         :param list filenames_db:
00621         :param float init_rmsd:
00622         :param list legend_names:
00623         :param str metric_name:
00624         :param str guide_metr:
00625         :param str common_path:
00626
00627     Returns:
00628         :return: figure number
00629     """
00630     best_metr_dic = {'rmsd': 'bsfr', 'angl': 'bsfn', 'andh': 'bsfh', 'and': 'bsfa', 'xor': 'bsfx'}
00631     metr_units = {'rmsd': 'Å', 'angl': '°', 'andh': 'contacts', 'and': 'contacts', 'xor': 'contacts'}
00632     # qry = "SELECT a.rmsd_goal_dist, b.vid FROM main_storage a join visited b on a.id=b.id join log c on a.id=c.id where c.cur_metr='{}' order
      by b.vid".format(metric_name)
00633     qry = "select a.{0}_goal_dist, b.vid from main_storage a join visited b on a.id=b.id join (select id, cur_metr from log where dst='VIZ'
      group by id) c on c.id=b.id where c.cur_metr='{1}' order by b.vid".format(guide_metr, metric_name)
00634     result_arr = [cur.execute(qry) for cur in cur_arr]
00635     fetched_all_arr = [res.fetchall() for res in result_arr]
00636     filt_res_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00637     # init_rmsd = filt_res_arr[0][0]
00638     max_non_init_rmsd = max(max(elem) for elem in filt_res_arr)
00639     common_point = max([min(elem) for elem in filt_res_arr])
00640
00641     ind_arr = list()
00642     for rmsd_for_db in filt_res_arr:
00643         i = 0
00644         while common_point < rmsd_for_db[i]:
00645             i += 1
00646         ind_arr.append(i)
00647
00648     # print('To reach common min point of {}A (rmsd)'.format(common_point))
```

```
00649     # for i, db in enumerate(filenames_db):
00650     #     print('{} : {} steps'.format(db.split('.')[0], ind_arr[i]))
00651
00652
00653     #     ################ FULL ####################
00654
00655     # qry = "select a.bsfr, b.rmsd_tot_dist, b.rmsd_goal_dist, c.vid from log a join main_storage b on a.id=b.id join visited c on c.id=a.id
          where a.dst='VIZ' and a.cur_metr='{}' order by a.lid".format(metric_name)
00656     qry = "select c.{0}, a.{1}_tot_dist, a.{1}_goal_dist, b.vid from main_storage a join visited b on a.id=b.id join (select id, max({0}) as
          {0}, cur_metr from log where dst='VIZ' group by id) c on c.id=b.id where c.cur_metr='{2}' order by b.vid".format(best_metr_dic[guide_metr],
          guide_metr, metric_name)
00657     result_arr = [cur.execute(qry) for cur in cur_arr]
00658     [res.fetchone() for res in result_arr]
00659     fetched_all_arr = [res.fetchall() for res in result_arr]
00660     bsf_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00661     for i in range(len(bsf_arr)):
00662         bsf_arr[i].insert(0, init_rmsd)
00663     for j in range(len(bsf_arr)):
00664         for i in range(len(bsf_arr[j]) - 1):
00665             if bsf_arr[j][i] < bsf_arr[j][i + 1]:
00666                 bsf_arr[j][i+1] = bsf_arr[j][i]
00667     trav_arr = [[dist[1] for dist in goal_dist] for goal_dist in fetched_all_arr]
00668     to_goal_arr = [[dist[2] for dist in goal_dist] for goal_dist in fetched_all_arr]
00669     non_shr = [[dist[3] for dist in goal_dist] for goal_dist in fetched_all_arr]
00670     # for i in range(len(non_shr)):
00671     #     non_shr[i].insert(0, 0)
00672
00673     max_len = max([len(goal_dist) for goal_dist in fetched_all_arr])
00674     max_trav = max([max(elem) for elem in trav_arr])
00675     common_point = min([min(elem) for elem in filt_res_arr])
00676     custom_path = '{}/full/'.format(common_path)
00677     try:
00678         os.mkdir(custom_path)
00679     except:
00680         pass
00681
00682     fig_num = plot_set(fig_num, to_goal_arr, legend_names, max_len, max_non_init_rmsd, init_rmsd, bsf_arr, common_point, max_trav, trav_arr,
          "full", guide_metr, metr_units[guide_metr], metric_name, custom_path, shrink=True)
00683     max_len = max([max(arr) for arr in non_shr])
00684     fig_num = plot_set(fig_num, to_goal_arr, legend_names, max_len, max_non_init_rmsd, init_rmsd, bsf_arr, common_point, max_trav, trav_arr,
          "full", guide_metr, metr_units[guide_metr], metric_name, custom_path, shrink=False, non_shrink_arr=non_shr)
00685
00686     return fig_num
00687
00688
00689 def plot_set(fig_num: int, to_goal_arr: list, legend_names: list, max_len: float, max_non_init_rmsd: float,
00690              init_metr: float, bsf_arr: list, common_point: float, max_trav: float, trav_arr: list, full_cut: str,
00691              metric: str, metr_units: str, same: str, custom_path: str, shrink: bool, non_shrink_arr: list = None) -> int:
00692     """
00693
00694     Args:
00695         :param int fig_num:
00696         :param list to_goal_arr:
00697         :param list legend_names:
00698         :param float max_len:
00699         :param float max_non_init_rmsd:
00700         :param float init_metr:
00701         :param float list bsf_arr:
00702         :param float common_point:
00703         :param float max_trav:
00704         :param list trav_arr:
00705         :param str full_cut:
00706         :param str metric:
00707         :param str metr_units:
00708         :param str same:
00709         :param str custom_path:
00710         :param bool shrink:
00711         :param list non_shrink_arr:
00712
00713     Returns:
00714         :return: fig number
00715         :rtype: int
00716     """
00717     # # #### SHRINK
00718     # ax_prop = {"min_lim_x": -max_len/80, "max_lim_x": max_len+max_len/80, "min_lim_y": 0, "max_lim_y":
          max_non_init_rmsd+max_non_init_rmsd/80, "min_ax_x": 0, "max_ax_x": max_len+max_len/80, "min_ax_y": 0, "max_ax_y":
          max_non_init_rmsd+max_non_init_rmsd/80, "ax_step_x": max_len/16, "ax_step_y": max_non_init_rmsd/20}
00719     # extra_line = {"ax_type": 'hor', "val": init_rmsd, "name": "init {} ({:3.2f} {})".format(metric, init_rmsd, metr_units)}
00720     # fig_num = single_plot(fig_num, ax_prop, to_goal_arr, None,            legend_names, '.', 0.3, bsf=False, rev=False,
          extra_line=extra_line, xlab="steps (20ps each)", ylab="to goal, A", title="{} | to goal vs traveled | {} | {}".format(metric, full_cut, same),
          filename="{}_to_goal_vs_traveled_{}_{}".format(metric, full_cut, same))  # to goal vs traveled | cut
```

```
00721     #
00722     # ax_prop = {"min_lim_x": -max_len/80, "max_lim_x": max_len+max_len/80, "min_lim_y": 0, "max_lim_y":
      max_non_init_rmsd+max_non_init_rmsd/80, "min_ax_x": 0, "max_ax_x": max_len+max_len/80, "min_ax_y": 0, "max_ax_y":
      max_non_init_rmsd+max_non_init_rmsd/80, "ax_step_x": max_len/16, "ax_step_y": max_non_init_rmsd/20}
00723     # extra_line = {"ax_type": 'hor', "val": init_rmsd, "name": "init {} ({:3.2f} {})".format(metric, init_rmsd, metr_units)}
00724     # fig_num = single_plot(fig_num, ax_prop,  bsf_arr,    None,            legend_names, '-', 1,   bsf=True,  rev=False,
      extra_line=extra_line, xlab="steps (20ps each)", ylab="steps", title="{} | to goal vs best_so_far | {} | {}".format(metric, full_cut, same),
      filename="{}_to_goal_vs_best_so_far_{}_{}".format(metric, full_cut, same))  # to goal vs best_so_far | cut
00725     #
00726     # ax_prop = {"min_lim_x": max_non_init_rmsd, "max_lim_x": common_point-common_point/10, "min_lim_y": -max_len/80, "max_lim_y":
      max_len+max_len/80, "min_ax_x": common_point, "max_ax_x": max_non_init_rmsd, "min_ax_y": 0, "max_ax_y": max_len+max_len/80, "ax_step_x":
      (max_non_init_rmsd-common_point)/16, "ax_step_y": max_len/20}
00727     # extra_line = {"ax_type": 'ver', "val": init_rmsd, "name": "init {} ({:3.2f} {})".format(metric, init_rmsd, metr_units)}
00728     # fig_num = single_plot(fig_num, ax_prop,   bsf_arr,   None,            legend_names, '-',  1,   bsf=True,  rev=True,
      extra_line=extra_line, xlab="to goal, A", ylab="steps", title="{} | best_so_far vs steps | {} | {}".format(metric, full_cut, same),
      filename="{}_best_so_far_vs_steps_{}_{}".format(metric, full_cut, same))  # best_so_far vs steps | cut
00729
00730     # #### NO SHRINK
00731     custom_path = custom_path+'shrink' if shrink else custom_path+'unshrink'
00732     try:
00733         os.mkdir(custom_path)
00734     except:
00735         pass
00736     ax_prop = {"min_lim_x": -max_len/80, "max_lim_x": max_len+max_len/80, "min_lim_y": 0, "max_lim_y": max_non_init_rmsd+max_non_init_rmsd/80,
00737                "min_ax_x": 0, "max_ax_x": max_len+max_len/80, "min_ax_y": 0, "max_ax_y": max_non_init_rmsd+max_non_init_rmsd/80, "ax_step_x":
      math.floor(max_len/16), "ax_step_y": max_non_init_rmsd/20}
00738     if metr_units == 'contacts':
00739         extra_line = [
00740             {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({} {})".format(metric.upper(), int(init_metr), metr_units), "col":
      "darkmagenta"},
00741             {"ax_type": 'hor', "val": min(min(elem) for elem in to_goal_arr), "name": "The lowest {} metric ({} {})".format(metric.upper(),
      int(min(min(elem) for elem in to_goal_arr)), metr_units), "col": "darkgreen"}]
00742     else:
00743         extra_line = [
00744             {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metric.upper(), init_metr, metr_units), "col":
      "darkmagenta"},
00745             {"ax_type": 'hor', "val": min(min(elem) for elem in to_goal_arr), "name": "The lowest {} metric ({:3.2f}
      {})".format(metric.upper(), min(min(elem) for elem in to_goal_arr), metr_units), "col": "darkgreen"}]
00746     if metric == 'rmsd':
00747         extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units), "col": "midnightblue"})
00748     title = "{} | to goal vs traveled | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00749     filename = "{}_to_goal_vs_traveled_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00750     filename = os.path.join(custom_path, filename)
00751     fig_num = single_plot(fig_num, ax_prop, to_goal_arr, non_shrink_arr,  legend_names.copy(), '.', 0.3, bsf=False, rev=False,
      extra_line=extra_line, shrink=shrink, xlab="Steps (20ps each)", ylab="Distance to the goal, {}".format(metr_units), title=title,
      filename=filename)  # to goal vs traveled | cut
00752
00753     for i in range(len(to_goal_arr)):
00754         ff = legend_names[i].split('with')[1].split('ff')[0].strip()
00755         title = "{} | to goal vs traveled | {} | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00756         filename = "{}_to_goal_vs_traveled_{}_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00757         filename = os.path.join(custom_path, filename)
00758         extra_line[1]["val"] = min(to_goal_arr[i])
00759         if metr_units == 'contacts':
00760             extra_line[1]["name"] = "The lowest {} metric ({} {})".format(metric.upper(), int(min(to_goal_arr[i])), metr_units)
00761         else:
00762             extra_line[1]["name"] = "The lowest {} metric ({:3.2f} {})".format(metric.upper(), min(to_goal_arr[i]), metr_units)
00763         fig_num = single_plot(fig_num, ax_prop, [to_goal_arr[i],], [non_shrink_arr[i],] if non_shrink_arr is not None else None,
      [legend_names[i],].copy(), '.', 0.3, bsf=False, rev=False, extra_line=extra_line, shrink=shrink, xlab="Steps (20ps each)",
00764                               ylab="Distance to the goal, {}".format(metr_units), title=title, filename=filename)  # to goal vs traveled | cut
00765
00766     if shrink:
00767         ax_prop = {"min_lim_x": max_non_init_rmsd, "max_lim_x": common_point-common_point/20, "min_lim_y": -max_trav/80, "max_lim_y":
      max_trav+max_trav/80,
00768                    "min_ax_x": common_point, "max_ax_x": max_non_init_rmsd, "min_ax_y": 0, "max_ax_y": max_trav+max_trav/80, "ax_step_x":
      (max_non_init_rmsd-common_point)/20, "ax_step_y": max_trav/20}
00769         if metr_units == 'contacts':
00770             extra_line = [
00771                 {"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({} {})".format(metric.upper(), int(init_metr), metr_units),
      "col": "darkmagenta"},
00772                 {"ax_type": 'ver', "val": min(min(elem) for elem in to_goal_arr), "name": "The lowest {} metric ({} {})".format(metric.upper(),
      int(min(min(elem) for elem in to_goal_arr)), metr_units), "col": "darkgreen"}]
00773         else:
00774             extra_line = [
00775                 {"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metric.upper(), init_metr, metr_units),
      "col": "darkmagenta"},
00776                 {"ax_type": 'ver', "val": min(min(elem) for elem in to_goal_arr), "name": "The lowest {} metric ({:3.2f}
      {})".format(metric.upper(), min(min(elem) for elem in to_goal_arr), metr_units), "col": "darkgreen"}]
00777         if metric == 'rmsd':
00778             extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units), "col":
      "midnightblue"})
```

```
00779         title = "{} | traveled vs to_goal | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00780         filename = "{}_traveled_vs_to_goal_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00781         filename = os.path.join(custom_path, filename)
00782         fig_num = single_plot(fig_num, ax_prop, to_goal_arr, trav_arr,    legend_names.copy(), '.', 1,   bsf=False, rev=True,
      extra_line=extra_line, shrink=shrink, xlab="Distance to the goal, {}".format(metr_units), ylab="Past dist, {}".format(metr_units),
      title=title, filename=filename)  # traveled vs to_goal | cut
00783
00784         for i in range(len(to_goal_arr)):
00785             ff = legend_names[i].split('with')[1].split('ff')[0].strip()
00786             title = "{} | traveled vs to_goal | {} | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00787             filename = "{}_traveled_vs_to_goal_{}_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00788             filename = os.path.join(custom_path, filename)
00789             extra_line[1]["val"] = min(to_goal_arr[i])
00790             if metr_units == 'contacts':
00791                 extra_line[1]["name"] = "The lowest {} metric ({} {})".format(metric.upper(), int(min(to_goal_arr[i])), metr_units)
00792             else:
00793                 extra_line[1]["name"] = "The lowest {} metric ({:3.2f} {})".format(metric.upper(), min(to_goal_arr[i]), metr_units)
00794             fig_num = single_plot(fig_num, ax_prop, [to_goal_arr[i],], [trav_arr[i],], [legend_names[i],].copy(), '.', 1, bsf=False, rev=True,
      extra_line=extra_line, shrink=shrink,
00795                                   xlab="Distance to the goal, {}".format(metr_units), ylab="Past dist, {}".format(metr_units), title=title,
      filename=filename)  # traveled vs to_goal | cut
00796
00797     if not shrink:
00798         for i in range(len(non_shrink_arr)):
00799             non_shrink_arr[i].insert(0, 0)
00800     ax_prop = {"min_lim_x": -max_len / 80, "max_lim_x": max_len + max_len / 80, "min_lim_y": 0, "max_lim_y": init_metr + init_metr / 80,  #
      max_non_init_rmsd + max_non_init_rmsd / 80,
00801                "min_ax_x": 0, "max_ax_x": max_len + max_len / 80, "min_ax_y": 0, "max_ax_y": init_metr + init_metr / 80, "ax_step_x":
      math.floor(max_len / 16), "ax_step_y": init_metr / 20}
00802     if metr_units == 'contacts':
00803         extra_line = [
00804             {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({} {})".format(metric.upper(), int(init_metr), metr_units), "col":
      "darkmagenta"},
00805             {"ax_type": 'hor', "val": min(min(elem) for elem in bsf_arr), "name": "The lowest {} metric ({} {})".format(metric.upper(),
      int(min(min(elem) for elem in bsf_arr)), metr_units), "col": "darkgreen"}]
00806     else:
00807         extra_line = [
00808             {"ax_type": 'hor', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metric.upper(), init_metr, metr_units), "col":
      "darkmagenta"},
00809             {"ax_type": 'hor', "val": min(min(elem) for elem in bsf_arr), "name": "The lowest {} metric ({:3.2f} {})".format(metric.upper(),
      min(min(elem) for elem in bsf_arr), metr_units), "col": "darkgreen"}]
00810     if metric == 'rmsd':
00811         extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units), "col": "midnightblue"})
00812     title = "{} | to goal vs best_so_far | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00813     filename = "{}_to_goal_vs_best_so_far_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00814     filename = os.path.join(custom_path, filename)
00815     fig_num = single_plot(fig_num, ax_prop, bsf_arr, non_shrink_arr, legend_names.copy(), '-', 1, bsf=True, rev=False, extra_line=extra_line,
      shrink=shrink, xlab="Steps (20ps each)", ylab="Distance to the goal, {}".format(metr_units), title=title, filename=filename)  # to goal vs
      best_so_far | cut
00816     for i in range(len(bsf_arr)):
00817         ff = legend_names[i].split('with')[1].split('ff')[0].strip()
00818         title = "{} | to goal vs best_so_far | {} | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00819         filename = "{}_to_goal_vs_best_so_far_{}_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00820         extra_line[1]["val"] = min(bsf_arr[i])
00821         if metr_units == 'contacts':
00822             extra_line[1]["name"] = "The lowest {} metric ({} {})".format(metric.upper(), int(min(bsf_arr[i])), metr_units)
00823         else:
00824             extra_line[1]["name"] = "The lowest {} metric ({:3.2f} {})".format(metric.upper(), min(bsf_arr[i]), metr_units)
00825         filename = os.path.join(custom_path, filename)
00826         fig_num = single_plot(fig_num, ax_prop, [bsf_arr[i],], [non_shrink_arr[i],] if non_shrink_arr is not None else None,
      [legend_names[i],].copy(), '-', 1, bsf=True, rev=False, extra_line=extra_line, shrink=shrink, xlab="Steps (20ps each)",
00827                               ylab="Distance to the goal, {}".format(metr_units), title=title, filename=filename)  # to goal vs best_so_far |
      cut
00828
00829     ax_prop = {"min_lim_x": max_non_init_rmsd, "max_lim_x": common_point-common_point/10, "min_lim_y": -max_len/80, "max_lim_y":
      max_len+max_len/80,
00830                "min_ax_x": common_point, "max_ax_x": max_non_init_rmsd, "min_ax_y": 0, "max_ax_y": max_len+max_len/80, "ax_step_x":
      (max_non_init_rmsd-common_point)/20, "ax_step_y": math.floor(max_len/20)}
00831
00832     if metr_units == 'contacts':
00833         extra_line = [
00834             {"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({} {})".format(metric.upper(), int(init_metr), metr_units), "col":
      "darkmagenta"},
00835             {"ax_type": 'ver', "val": min(min(elem) for elem in bsf_arr), "name": "The lowest {} metric ({} {})".format(metric.upper(),
      int(min(min(elem) for elem in bsf_arr)), metr_units), "col": "darkgreen"}]
00836     else:
00837         extra_line = [
00838             {"ax_type": 'ver', "val": init_metr, "name": "Initial {} metric ({:3.2f} {})".format(metric.upper(), init_metr, metr_units), "col":
      "darkmagenta"},
00839             {"ax_type": 'ver', "val": min(min(elem) for elem in bsf_arr), "name": "The lowest {} metric ({:3.2f} {})".format(metric.upper(),
      min(min(elem) for elem in bsf_arr), metr_units), "col": "darkgreen"}]
```

```
00840     if metric == 'rmsd':
00841         extra_line.append({"ax_type": 'hor', "val": 2.7, "name": "Typical folding mark (2.7 {})".format(metr_units), "col": "midnightblue"})
00842     title = "{} | best_so_far vs steps | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00843     filename = "{}_best_so_far_vs_steps_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink')
00844     filename = os.path.join(custom_path, filename)
00845     fig_num = single_plot(fig_num, ax_prop, bsf_arr,     non_shrink_arr, legend_names.copy(), '-',  1,   bsf=True,  rev=True,
      extra_line=extra_line, shrink=shrink, xlab="Distance to the goal, {}".format(metr_units), ylab="Steps (20 ps each)", title=title,
      filename=filename)  # best_so_far vs steps | cut
00846     for i in range(len(bsf_arr)):
00847         ff = legend_names[i].split('with')[1].split('ff')[0].strip()
00848         title = "{} | best_so_far vs steps | {} | {} | {} | {}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00849         filename = "{}_best_so_far_vs_steps_{}_{}_{}_{}".format(metric, full_cut, same, 'shrink' if shrink else 'unshrink', ff)
00850         extra_line[1]["val"] = min(bsf_arr[i])
00851         if metr_units == 'contacts':
00852             extra_line[1]["name"] = "The lowest {} metric ({} {})".format(metric.upper(), int(min(bsf_arr[i])), metr_units)
00853         else:
00854             extra_line[1]["name"] = "The lowest {} metric ({:3.2f} {})".format(metric.upper(), min(bsf_arr[i]), metr_units)
00855         filename = os.path.join(custom_path, filename)
00856         fig_num = single_plot(fig_num, ax_prop, [bsf_arr[i],], [non_shrink_arr[i],] if non_shrink_arr is not None else None,
      [legend_names[i],].copy(), '-', 1, bsf=True, rev=True, extra_line=extra_line, shrink=shrink,
00857                               xlab="Distance to the goal, {}".format(metr_units), ylab="Steps (20 ps each)", title=title, filename=filename)  #
      best_so_far vs steps | cut
00858
00859     return fig_num
00860
00861
00862 def single_plot(fig_num: int, ax_prop: dict, arr_A: list, arr_B: list, filenames_db: list, marker: str, mark_size: float,
00863                 bsf: bool, rev: bool, shrink: bool, xlab: str, ylab: str, title: str, filename: str,
00864                 extra_line: list = None, mdpi: int = 400, second_ax: dict = None, sec_arr: list = None) -> int:
00865     """Main plotting function
00866
00867     Args:
00868         :param int fig_num: figure number, it should not matter, since we close all figures regularly
00869         :param dict ax_prop: axis properties
00870         :param list arr_A: typically Y values
00871         :param list arr_B: typically X values
00872         :param list filenames_db: line names
00873         :param str marker: type of the marker
00874         :param float mark_size: size of the marker
00875         :param bool bsf: best so far version
00876         :param bool rev: reversed
00877         :param bool shrink: whether to ignore x values, and just plot all y values
00878         :param str xlab: x label
00879         :param str ylab:  y label
00880         :param str title: plot title
00881         :param str filename: output filename
00882         :param list extra_line: whether to plot extra line, if so contains its properties
00883         :param int mdpi: plot resolution
00884         :param dict second_ax: whether to plot second Y axis, if so this contains dict with properties
00885         :param list sec_arr: value for the second axis
00886
00887     Returns:
00888         :return: figure number, it should not matter, since we close all figures regularly
00889     """
00890     fig_num += 1
00891     # for fname in ['angl_version_of_best_traj_angl_only_results_gromos_trp_300_2_fixed_vs_pt_energy',
00892     #  'rmsd_version_of_best_traj_rmsd_only_results_gromos_trp_300_2_fixed_vs_pt_energy',
00893     #  'rmsd_version_of_best_traj_rmsd_vs_dist',
00894     #  'xor_version_of_best_traj_rmsd_only_results_opls_trp_300_2_fixed_vs_angl',
00895     #  'xor_version_of_best_traj_rmsd_only_results_opls_trp_300_2_fixed_vs_pt_energy',
00896     #  'xor_version_of_best_traj_angl_only_results_opls_trp_300_2_fixed_vs_pt_energy',
00897     #              'rmsd_to_goal_vs_best_so_far_full_RMSD_unshrink']:
00898     #     if fname in filename:
00899     #         print('found')
00900
00901     w, h = figaspect(0.5)
00902     fig = plt.figure(fig_num, figsize=(w, h))
00903     #
00904     ax = fig.gca()
00905     fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(w, h), sharex=True, squeeze=False)
00906     plt.xlim(ax_prop["min_lim_x"], ax_prop["max_lim_x"])
00907     plt.ylim(ax_prop["min_lim_y"], ax_prop["max_lim_y"])
00908
00909     major_xticks = np.arange(ax_prop["min_ax_x"], ax_prop["max_ax_x"], ax_prop["ax_step_x"])
00910     major_yticks = np.arange(ax_prop["min_ax_y"], ax_prop["max_ax_y"], ax_prop["ax_step_y"])
00911
00912     if ax_prop["ax_step_y"] is not None:
00913         if major_yticks[-1] > ax_prop["max_lim_y"]:  # fix inconsistency in real numbers
00914             major_yticks[-1] = ax_prop["max_lim_y"]
00915         if ax_prop["max_lim_y"] - major_yticks[-1] > ax_prop["ax_step_y"]:  # this should not happen, but just in case..
00916             major_yticks = np.append(major_yticks, major_yticks[-1] + ax_prop["ax_step_y"])
```

```
00917         elif ax_prop["max_lim_y"] - major_yticks[-1] > 0.7*ax_prop["ax_step_y"]:
00918             major_yticks = np.append(major_yticks, ax_prop["max_lim_y"])
00919 
00920     if ax_prop["ax_step_x"] is not None:
00921         if ax_prop["max_lim_x"] - major_xticks[-1] > ax_prop["ax_step_x"]:  # this should not happen, but just in case..
00922             print('2', filename)
00923             major_xticks = np.append(major_xticks, int(major_xticks[-1] + ax_prop["ax_step_x"]) if isinstance(ax_prop["ax_step_x"], int) else
      (major_xticks[-1] + ax_prop["ax_step_x"]))
00924         elif ax_prop["max_lim_x"] - major_xticks[-1] > 0.7 * ax_prop["ax_step_x"]:
00925             print('1', filename)
00926             major_xticks = np.append(major_xticks, int(ax_prop["max_lim_x"]) if isinstance(ax_prop["ax_step_x"], int) else
      ax_prop["max_lim_x"])
00927 
00928         if arr_B is not None and abs(arr_B[0][-1] - major_xticks[-1]) < 0.5 * ax_prop["ax_step_x"]:
00929             major_xticks[-1] = arr_B[0][-1]
00930         elif abs(max(len(elem) for elem in arr_A) - major_xticks[-1]) < 0.5 * ax_prop["ax_step_x"]:
00931             major_xticks[-1] = max(len(elem) for elem in arr_A)
00932 
00933     if major_xticks is not None:
00934         ax[0][0].set_xticks(major_xticks)
00935     if major_yticks is not None:
00936         ax[0][0].set_yticks(major_yticks)
00937     # if minor_xticks is not None:
00938     #     ax.set_xticks(minor_xticks, minor=True)
00939     # if minor_yticks is not None:
00940     #     ax.set_yticks(minor_yticks, minor=True)
00941     top_ax = ax[0][0]
00942     if second_ax is not None:
00943         ax2 = ax[0][0].twinx()
00944         major_yticks2 = np.arange(second_ax["min_ax_y"], second_ax["max_ax_y"], second_ax["ax_step_y"])
00945 
00946         if major_yticks2[-1] > second_ax["max_lim_y"]:  # fix inconsistency in real numbers
00947             major_yticks2[-1] = second_ax["max_lim_y"]
00948 
00949         if second_ax["max_lim_y"] - major_yticks2[-1] > second_ax["ax_step_y"]:
00950             major_yticks2 = np.append(major_yticks2, major_yticks2[-1] + second_ax["ax_step_y"])
00951         elif second_ax["max_lim_y"] - major_yticks2[-1] > 0.7*second_ax["ax_step_y"]:
00952             major_yticks2 = np.append(major_yticks2, second_ax["max_lim_y"])
00953 
00954         ax2.set_yticks(major_yticks2)
00955         ax2.tick_params(direction='out', length=6, width=1, grid_alpha=0.5)
00956         # ax[0].right_ax.set_ylim(second_ax["min_lim_y"], second_ax["max_lim_y"])
00957         ax2.set_ylim(second_ax["min_lim_y"], second_ax["max_lim_y"])
00958         ax2.plot(range(len(sec_arr)), sec_arr, color='r', alpha=0.75)
00959         ax2.set_ylabel(second_ax["label"] if second_ax["label"][-2] != ',' else second_ax["label"][0:-2])
00960         top_ax = ax2
00961 
00962 
00963 
00964     ax[0][0].tick_params(direction='out', length=6, width=1, grid_alpha=0.5)
00965     ax[0][0].grid(which='both', linestyle='dotted')
00966     plt.xticks(rotation=30)
00967     plt.subplots_adjust(top=0.95, bottom=0.16, left=0.09, right=0.90)
00968 
00969     lines_b = []
00970     for i, bsf_trav_to_goal in enumerate(arr_A):
00971         if not shrink:  # use provided array arr_B
00972             if rev:
00973                 line_b, = ax[0][0].plot(arr_A[i], arr_B[i], marker, markersize=mark_size, alpha=0.75)
00974             else:
00975                 line_b, = ax[0][0].plot(arr_B[i], arr_A[i], marker, markersize=mark_size, alpha=0.75)
00976         else:  # generate array from 0 to len(arr_A)
00977             if rev:
00978                 if bsf:
00979                     line_b, = ax[0][0].plot(arr_A[i], range(len(arr_A[i])), marker, markersize=mark_size, alpha=0.75)
00980                 else:
00981                     line_b, = ax[0][0].plot(arr_A[i], arr_B[i], marker, markersize=mark_size, alpha=0.75)
00982             else:
00983                 line_b, = ax[0][0].plot(range(len(arr_A[i])), arr_A[i], marker, markersize=mark_size, alpha=0.75)
00984         lines_b.append(line_b)
00985 
00986     if extra_line is not None:
00987         for el in extra_line:
00988             if el["ax_type"] == 'ver':
00989                 straight_line = ax[0][0].axvline(x=el["val"], color=el["col"], linestyle='--', alpha=0.75)  #
00990             elif el["ax_type"] == 'hor':
00991                 straight_line = ax[0][0].axhline(y=el["val"], color=el["col"], linestyle='--', alpha=0.75)
00992             else:
00993                 raise Exception('Wrong ax type')
00994             lines_b.append(straight_line)
00995             filenames_db.append(el["name"])
```

```
00996         if el["ax_type"] == 'ver':
00997             if not rev:
00998                 ax[0][0].annotate('Folding direction', xytext=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
      ax_prop["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 5 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 * ax_prop["ax_step_y"]),
      arrowprops={'arrowstyle': '->', 'lw': 1.3, 'color': 'mediumblue'}, va='center')  # -->
00999             else:
01000                 ax[0][0].annotate('Folding direction', xytext=(ax_prop["max_ax_x"] - 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
      ax_prop["ax_step_y"]), xy=(ax_prop["max_ax_x"] - 5 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 * ax_prop["ax_step_y"]),
      arrowprops={'arrowstyle': '->', 'lw': 1.3, 'color': 'mediumblue'}, va='center')  # -->
01001         else:
01002             if not rev:
01003                 if second_ax is not None:
01004                     ax2.annotate('Folding direction', xytext=(ax_prop["min_ax_x"] + 3.5 * ax_prop["ax_step_x"], second_ax["max_lim_y"] - 1 *
      second_ax["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 3.5 * ax_prop["ax_step_x"], second_ax["max_lim_y"] - 4 * second_ax["ax_step_y"]),
      arrowprops={'arrowstyle': '->', 'lw': 1.3, 'color': 'mediumblue'}, ha='center')  # <--
01005                 else:
01006                     ax[0][0].annotate('Folding direction', xytext=(ax_prop["min_ax_x"] + 3.5 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
      ax_prop["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 3.5 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 4 * ax_prop["ax_step_y"]),
      arrowprops={'arrowstyle': '->', 'lw': 1.3, 'color': 'mediumblue'}, ha='center')  # <--
01007             else:
01008                 pass # does not exist
01009                 # ax.annotate('folding direction', xytext=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
      ax_prop["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 4 * ax_prop["ax_step_y"]),
      arrowprops={'arrowstyle': '->', 'lw': 1.5, 'color': 'mediumblue'}, ha='center')  # -->
01010
01011     if second_ax is not None:
01012         lines_b.append(ax[0][0].plot([], [], marker, color='r', markersize=mark_size)[0])
01013         filenames_db.append(second_ax["line_name"])
01014
01015     ax[0][0].set_xlabel(xlab)
01016     ax[0][0].set_ylabel(ylab if ylab[-2] != ',' else ylab[0:-2])
01017     top_ax.legend(lines_b, filenames_db)
01018     plt.title(title)
01019     try:
01020         plt.savefig(filename, dpi=mdpi, transparent=True, bbox_inches='tight', pad_inches=0.02)
01021     except:
01022         plt.show()
01023     plt.close('all')
01024     return fig_num
01025
01026
01027 if __name__ == '__main__':
01028     main()
```

4.3 compute_corr_between_metr.py File Reference

Namespaces

- compute_corr_between_metr

Functions

- def compute_corr_between_metr.main ()
- def compute_corr_between_metr.myr (y, f)
- def compute_corr_between_metr.myr_rev (y, f)
- def compute_corr_between_metr.fill_stat_dict (filenames_db, legend_names, guide_metr)

Variables

- compute_corr_between_metr.main_dict = dict ()
- compute_corr_between_metr.full_dict = dict ()

4.4 compute_corr_between_metr.py

```
00001 #!/usr/bin/env python3
00002
00003 import os
00004 import sqlite3 as lite
00005 import matplotlib.pyplot as plt
00006 import numpy as np
00007 from matplotlib.figure import figaspect
00008 import multiprocessing as mp
00009 from sklearn import preprocessing
00010 from sklearn.metrics import r2_score
00011
00012
00013 main_dict = dict()
00014 full_dict = dict()
00015
```

```
00016 def main():
00017     global main_dict, full_dict
00018     batch_arr = list()
00019 
00020     # #########   TRP   ######################
00021     filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_amber_trp_300_2_fixed.sqlite3', 'results_charm_trp_300_fixed.sqlite3',
      'results_charm_trp_300_2_fixed.sqlite3', 'results_gromos_trp_300_fixed.sqlite3', 'results_gromos_trp_300_2_fixed.sqlite3',
      'results_opls_trp_300_fixed.sqlite3', 'results_opls_trp_300_2_fixed.sqlite3']
00022     legend_names = ['TRP amber_1', 'TRP amber_2', 'TRP charm_1', 'TRP charm_2', 'TRP gromos_1', 'TRP gromos_2', 'TRP opls_1', 'TRP opls_2']
00023     common_path = '../trp_all_compar'
00024     batch_arr.append((filenames_db, legend_names, common_path))
00025     for fname in filenames_db:
00026         main_dict[fname] = dict()
00027         for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00028             main_dict[fname][metr] = {'rmsd': [0, 0, 0], 'angl': [0, 0, 0], 'andh': [0, 0, 0], 'and': [0, 0, 0], 'xor': [0, 0, 0], 'pt': [0, 0,
      0]}
00029         full_dict[fname] = dict()
00030         for g_metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00031             full_dict[fname][g_metr] = dict()
00032             for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00033                 full_dict[fname][g_metr][metr] = {'rmsd': [0, 0, 0], 'angl': [0, 0, 0], 'andh': [0, 0, 0], 'and': [0, 0, 0], 'xor': [0, 0, 0],
      'pt': [0, 0, 0]}
00034 
00035 
00036     # # ##################  VIL  ######################
00037 
00038     filenames_db = ['results_amber_vil_300.sqlite3', 'results_charm_vil_300.sqlite3', 'results_gromos_vil_300.sqlite3',
      'results_opls_vil_300.sqlite3']
00039     legend_names = ['VIL amber', 'VIL charm', 'VIL gromos', 'VIL opls']
00040     common_path = '../vil_all_compar'
00041     batch_arr.append((filenames_db, legend_names, common_path))
00042     for fname in filenames_db:
00043         main_dict[fname] = dict()
00044         for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00045             main_dict[fname][metr] = {'rmsd': [0, 0, 0], 'angl': [0, 0, 0], 'andh': [0, 0, 0], 'and': [0, 0, 0], 'xor': [0, 0, 0], 'pt': [0, 0,
      0]}
00046         full_dict[fname] = dict()
00047         for g_metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00048             full_dict[fname][g_metr] = dict()
00049             for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00050                 full_dict[fname][g_metr][metr] = {'rmsd': [0, 0, 0], 'angl': [0, 0, 0], 'andh': [0, 0, 0], 'and': [0, 0, 0], 'xor': [0, 0, 0],
      'pt': [0, 0, 0]}
00051 
00052 
00053     # # ##################  GB1  ######################
00054     # #
00055     filenames_db = ['results_amber_gb1_300.sqlite3', 'results_charm_gb1_300.sqlite3', 'results_gromos_gb1_300.sqlite3',
      'results_opls_gb1_300.sqlite3']
00056     legend_names = ['GB1 amber', 'GB1 charm', 'GB1 gromos', 'GB1 opls']
00057     common_path = '../gb1_all_compar'
00058     batch_arr.append((filenames_db, legend_names, common_path))
00059     for fname in filenames_db:
00060         main_dict[fname] = dict()
00061         for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00062             main_dict[fname][metr] = {'rmsd': [0, 0, 0], 'angl': [0, 0, 0], 'andh': [0, 0, 0], 'and': [0, 0, 0], 'xor': [0, 0, 0], 'pt': [0, 0,
      0]}
00063         full_dict[fname] = dict()
00064         for g_metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00065             full_dict[fname][g_metr] = dict()
00066             for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00067                 full_dict[fname][g_metr][metr] = {'rmsd': [0, 0, 0], 'angl': [0, 0, 0], 'andh': [0, 0, 0], 'and': [0, 0, 0], 'xor': [0, 0, 0],
      'pt': [0, 0, 0]}
00068 
00069 
00070 
00071     for filenames_db, legend_names, common_path in batch_arr:
00072         for guide_metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00073             fill_stat_dict(filenames_db, legend_names, guide_metr)
00074 
00075     with open('correlation.tex', 'w') as tex_table:
00076         # for db_name in main_dict.keys():
00077         #     tex_table.writelines(['\n\\begin{table}[h]\n', '\centering\n', '\sisetup{table-align-text-post=false}\n',
00078         #                           '\\begin{tabular}{@{}|l|\
00079         #                          |S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]|\
00080         #                          |S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]|\
00081         #                          |S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]|\
00082         #                          |S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]|\
00083         #                          |S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]|\
00084         #                          |S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]|\
00085         #                          |@{}}\n \\hline\n']) 
```

```
00086         #     tex_table.write(' \multirow{2}{*}{metric\_y} & \\multicolumn{3}{c@{}|}{rmsd} & \\multicolumn{3}{c@{}|}{angl} & \\multicolumn{3}{c@{}|}{andh} & \\multicolumn{3}{c@{}|}{and} & \\multicolumn{3}{c@{}|}{xor} & \\multicolumn{3}{c@{}|}{pot ener} \\\\ \\cline{2-19}\n')
00087         #     tex_table.write(' & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} \\\\ \\hline\n'.format('{cor\_xy}', '{d\_xy}', '{d\_yx}', '{cor\_xy}', '{d\_xy}', '{d\_yx}', '{cor\_xy}', '{d\_xy}', '{d\_yx}', '{cor\_xy}', '{d\_xy}', '{d\_yx}', '{cor\_xy}', '{d\_xy}', '{d\_yx}', '{cor\_xy}', '{d\_xy}', '{d\_yx}'))
00088         #     for gm in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00089         #                         if gm == 'andh':
00090         #                     tw = 'and_h'
00091         #                 else:
00092         #                     tw = gm
00093         #                 tex_table.write('{} '.format(tw.upper()))
00094         #         for chm in ['rmsd', 'angl', 'andh', 'and', 'xor', 'pt']:
00095         #             val1 = main_dict[db_name][gm][chm][0]
00096         #             val2 = main_dict[db_name][gm][chm][1]
00097         #             val3 = main_dict[db_name][gm][chm][2]
00098         #             if abs(val1) > 99.999:
00099         #                 tex_table.write(' & {{$<-99$}} ')
00100         #             elif abs(val1) > 10.0:
00101         #                 tex_table.write(' & {{${}$}} '.format(int(round(val1))))
00102         #             else:
00103         #                 tex_table.write(' & {:3.2f} '.format(val1))
00104         #
00105         #             if abs(val2) > 99.999:
00106         #                 tex_table.write(' & {{$<-99$}} ')
00107         #             elif abs(val2) > 10.0:
00108         #                 tex_table.write(' & {{${}$}} '.format(int(round(val2))))
00109         #             else:
00110         #                 tex_table.write(' & {:3.2f} '.format(val2))
00111         #
00112         #             if abs(val3) > 99.999:
00113         #                 tex_table.write(' & {{$<-99$}} ')
00114         #             elif abs(val3) > 10.0:
00115         #                 tex_table.write(' & {{${}$}} '.format(int(round(val3))))
00116         #             else:
00117         #                 tex_table.write(' & {:3.2f} '.format(val3))
00118         #             # tex_table.write(' & {:3.2f} & {:3.2f} & {:3.2f} '.format(main_dict[db_name][gm][chm][0], main_dict[db_name][gm][chm][1], main_dict[db_name][gm][chm][2]))
00119         #         tex_table.write('\\\\ \\hline\n')
00120         # tex_table.writelines(['\\end {tabular}\n', '\\caption{{{}}}\n'.format('DB: {}'.format(db_name.translate(str.maketrans({"_": r"\_"})))), '\\end {table}\n'])
00121         # tex_table.write('\n\n\n')
00122
00123
00124 #         ############ CORR ONLY  ############################
00125
00126         for db_name in main_dict.keys():
00127             tex_table.writelines(['\n\\begin{table}[t]\n', '\sisetup{table-align-text-post=false}\n',
00128                                   '\\begin{tabular}{@{}|l|S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]|S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]|@{}}\n\\rowcolor{lightgray}\n'])
00129             tex_table.write(' {} & {\\glsentryshort{rmsd}} & {\\glsentryshort{angl}} & {\\glsentryshort{andh}} & {\\glsentryshort{and}} & {\\glsentryshort{xor}} & {Potential energy} \\\\ \\hline \n')
00130             # tex_table.write(' {} & {RMSD} & {ANGL} & {AND\_H} & {AND} & {XOR} & {Potential energy} \\\\ \\hline \n')
00131             # tex_table.write(' & {} & {} & {} & {} & {} & {}  \\\\ \\hline\n'.format('{cor\_xy}', '{cor\_xy}', '{cor\_xy}', '{cor\_xy}', '{cor\_xy}', '{cor\_xy}'))
00132             for gm in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00133                 if gm == 'andh':
00134                     tw = '\\glsentryshort{andh}'
00135                 else:
00136                     tw = '\\glsentryshort{{{}}}'.format(gm)
00137                 tex_table.write('{} '.format(tw))
00138                 for chm in ['rmsd', 'angl', 'andh', 'and', 'xor', 'pt']:
00139                     val1 = main_dict[db_name][gm][chm][0]
00140                     if abs(val1) > 99.999:
00141                         tex_table.write(' & {{$<-99$}} ')
00142                     elif abs(val1) > 10.0:
00143                         tex_table.write(' & {{${}$}} '.format(int(round(val1))))
00144                     else:
00145                         tex_table.write(' & {:3.2f} '.format(val1))
00146
00147                 tex_table.write('\\\\ \\hline\n')
00148             db_name1 = db_name.split('.')[0]
00149             pr_1 = db_name1.split('_')[2]
00150             ff_2 = db_name1.split('_')[1]
00151             if pr_1 == 'trp':
00152                 if '2' in db_name:
00153                     tex_table.writelines(['\\end {tabular}\n', '\\label {{cor_{}}}\n'.format(db_name1),
00154                                           '\\caption{{{}}}\n'.format(
00155                                               'orrelation coefficients among metrics and potential energy for the second simulation of \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field. Rows simultaneously represent the best trajectory according to the listed metric and correlation between this metric and other metrics and potential energy.'.format(
```

```
00156                                           pr_1, ff_2)), '\\end {table}\n'])
00157                 else:
00158                     tex_table.writelines(['\\end {tabular}\n', '\\label {{cor_{}}}\n'.format(db_name1),
00159                                           '\\caption{{{}}}\n'.format(
00160                                               'orrelation coefficients among metrics and potential energy for the first simulation of \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field. Rows simultaneously represent the best trajectory according to the listed metric and correlation between this metric and other metrics and potential energy.'.format(
00161                                           pr_1, ff_2)), '\\end {table}\n'])
00162             else:
00163                 tex_table.writelines(['\\end {tabular}\n', '\\label {{cor_{}}}\n'.format(db_name1),
00164                                       '\\caption{{{}}}\n'.format(
00165                     'orrelation coefficients among metrics and potential energy for simulation of \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field. Rows simultaneously represent the best trajectory according to the listed metric and correlation between this metric and other metrics and potential energy.'.format(pr_1, ff_2)), '\\end {table}\n'])
00166
00167             tex_table.write('\n\n\n')
00168
00169
00170
00171 #         ################### DET ONLY  ############################
00172         tex_table.write('\\begin{landscape}')
00173         for db_name in main_dict.keys():
00174             tex_table.writelines(['\n\\begin{table}\n', '\sisetup{table-align-text-post=false}\n',
00175                                   '\\begin{tabular}{@{}l|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|@{}}\n\\rowcolor{lightgray}\n'])
00176             tex_table.write('\multirow{2}{*}{} & \multicolumn{2}{c@{}|}{\glsentryshort{rmsd}} & \multicolumn{2}{c@{}|}{\glsentryshort{angl}} & \multicolumn{2}{c@{}|}{\glsentryshort{andh}} & \multicolumn{2}{c@{}|}{\glsentryshort{and}} & \multicolumn{2}{c@{}|}{\glsentryshort{xor}} & \multicolumn{2}{c@{}|}{Potential energy} \\\\ \cline{2-13}\n')
00177             tex_table.write(' & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} \\\\ \\hline\n'.format('{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}'))
00178             for gm in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00179                 if gm == 'andh':
00180                     tw = '\\glsentryshort{andh}'
00181                 else:
00182                     tw = '\\glsentryshort{{{}}}'.format(gm)
00183                 tex_table.write('{} '.format(tw))
00184                 for chm in ['rmsd', 'angl', 'andh', 'and', 'xor', 'pt']:
00185                     val2 = main_dict[db_name][gm][chm][1]
00186                     val3 = main_dict[db_name][gm][chm][2]
00187
00188                     if abs(val2) > 99.999:
00189                         tex_table.write(' & {{$<-99$}} ')
00190                     elif abs(val2) > 10.0:
00191                         tex_table.write(' & {{${}$}} '.format(int(round(val2))))
00192                     else:
00193                         tex_table.write(' & {:3.2f} '.format(val2))
00194
00195                     if abs(val3) > 99.999:
00196                         tex_table.write(' & {{$<-99$}} ')
00197                     elif abs(val3) > 10.0:
00198                         tex_table.write(' & {{${}$}} '.format(int(round(val3))))
00199                     else:
00200                         tex_table.write(' & {:3.2f} '.format(val3))
00201                     # tex_table.write(' & {:3.2f} & {:3.2f} & {:3.2f} '.format(main_dict[db_name][gm][chm][0], main_dict[db_name][gm][chm][1], main_dict[db_name][gm][chm][2]))
00202                 tex_table.write('\\\\ \\hline\n')
00203
00204             db_name1 = db_name.split('.')[0]
00205             pr_1 = db_name1.split('_')[2]
00206             ff_2 = db_name1.split('_')[1]
00207             if pr_1 == 'trp':
00208                 if '2' in db_name:
00209                     tex_table.writelines(['\\end {tabular}\n', '\\label {{det_{}}}\n'.format(db_name1),
00210                                           '\\caption{{{}}}\n'.format(
00211                                               'Determination coefficients among metrics and potential energy for the second simulation of \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field. Rows simultaneously represent the best trajectory according to the listed metric and determination between this metric and other metrics and potential energy.'.format(
00212                                           pr_1, ff_2)), '\\end {table}\n'])
00213                 else:
00214                     tex_table.writelines(['\\end {tabular}\n', '\\label {{det_{}}}\n'.format(db_name1),
00215                                           '\\caption{{{}}}\n'.format(
00216                                               'Determination coefficients among metrics and potential energy for the first simulation of \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field. Rows simultaneously represent the best trajectory according to the listed metric and determination between this metric and other metrics and potential energy.'.format(
00217                                           pr_1, ff_2)), '\\end {table}\n'])
00218             else:
00219                 tex_table.writelines(['\\end {tabular}\n', '\\label {{det_{}}}\n'.format(db_name1),
00220                                       '\\caption{{{}}}\n'.format(
```

```
00221                   'Determination coefficients among metrics and potential energy for simulation of \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field. Rows simultaneously represent the best trajectory according to the listed metric and determination between this metric and other metrics and potential energy.'.format(pr_1, ff_2)), '\\end {table}\n'])
00222             tex_table.write('\n\n\n')
00223         tex_table.write('\\end{landscape}')
00224 
00225     with open('full_correlation.tex', 'w') as tex_table:
00226 
00227         #         ############ CORR ONLY  ############################
00228 
00229         for db_name in main_dict.keys():
00230             for guid_m in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00231                 tex_table.writelines(['\n\\begin{table}[t]\n', '\sisetup{table-align-text-post=false}\n',
00232                                       '\\begin{tabular}{@{}|l|S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]|S[table-format=2.2] |S[table-format=2.2] |S[table-format=2.2]|@{}}\n\\rowcolor{lightgray}\n'])
00233                 tex_table.write(' {} & {\\glsentryshort{rmsd}} & {\\glsentryshort{angl}} & {\\glsentryshort{andh}} & {\\glsentryshort{and}} & {\\glsentryshort{xor}} & {Potential energy} \\\\ \\hline \n')
00234                 # tex_table.write(' {} & {RMSD} & {ANGL} & {AND\_H} & {AND} & {XOR} & {Potential energy} \\\\ \\hline \n')
00235                 # tex_table.write(' & {} & {} & {} & {} & {} & {}  \\\\ \\hline\n'.format('{cor\_xy}', '{cor\_xy}', '{cor\_xy}', '{cor\_xy}', '{cor\_xy}', '{cor\_xy}'))
00236                 for gm in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00237                     if gm == 'andh':
00238                         tw = '\\glsentryshort{andh}'
00239                     else:
00240                         tw = '\\glsentryshort{{{}}}'.format(gm)
00241                     tex_table.write('{} '.format(tw))
00242                     for chm in ['rmsd', 'angl', 'andh', 'and', 'xor', 'pt']:
00243                         try:
00244                             val1 = full_dict[db_name][guid_m][gm][chm][0]
00245                         except:
00246                             a = 8
00247                         if abs(val1) > 99.999:
00248                             tex_table.write(' & {{$<-99$}} ')
00249                         elif abs(val1) > 10.0:
00250                             tex_table.write(' & {{${}$}} '.format(int(round(val1))))
00251                         else:
00252                             tex_table.write(' & {:3.2f} '.format(val1))
00253 
00254                     tex_table.write('\\\\ \\hline\n')
00255                 db_name1 = db_name.split('.')[0]
00256                 pr_1 = db_name1.split('_')[2]
00257                 ff_2 = db_name1.split('_')[1]
00258                 if pr_1 == 'trp':
00259                     if '2' in db_name:
00260                         tex_table.writelines(['\\end {tabular}\n', '\\label {{cor_{}_{}}}\n'.format(guid_m, db_name1),
00261                                               '\\caption{{{}}}\n'.format(
00262                                                   'Correlation coefficients among metrics and potential energy for the second simulation of \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field for \\glsentryshort{{{}}} guide metric.'.format(
00263                                                       pr_1, ff_2, guid_m)), '\\end {table}\n'])
00264                     else:
00265                         tex_table.writelines(['\\end {tabular}\n', '\\label {{cor_{}_{}}}\n'.format(guid_m, db_name1),
00266                                               '\\caption{{{}}}\n'.format(
00267                                                   'Correlation coefficients among metrics and potential energy for the first simulation of \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field for \\glsentryshort{{{}}} guide metric.'.format(
00268                                                       pr_1, ff_2, guid_m)), '\\end {table}\n'])
00269                 else:
00270                     tex_table.writelines(['\\end {tabular}\n', '\\label {{cor_{}_{}}}\n'.format(guid_m, db_name1),
00271                                           '\\caption{{{}}}\n'.format(
00272                                               'Correlation coefficients among metrics and potential energy for simulation of \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field for \\glsentryshort{{{}}} guide metric.'.format(
00273                                                   pr_1, ff_2, guid_m)), '\\end {table}\n'])
00274                 tex_table.write('\n\n\n')
00275 
00276         #         ################### DET ONLY  ############################
00277         tex_table.write('\\begin{landscape}')
00278         for db_name in main_dict.keys():
00279             for guid_m in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00280                 tex_table.writelines(['\n\\begin{table}\n', '\sisetup{table-align-text-post=false}\n',
00281                                       '\\begin{tabular}{@{}|l|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|S[table-format=3.2] |S[table-format=3.2] |S[table-format=3.2]|@{}}\n\\rowcolor{lightgray}\n'])
00282                 tex_table.write('\multirow{2}{*}{} & \multicolumn{2}{c@{}|}{\glsentryshort{rmsd}} & \multicolumn{2}{c@{}|}{\glsentryshort{angl}} & \multicolumn{2}{c@{}|}{\glsentryshort{andh}} & \multicolumn{2}{c@{}|}{\glsentryshort{and}} & \multicolumn{2}{c@{}|}{\glsentryshort{xor}} & \multicolumn{2}{c@{}|}{Potential energy} \\\\ \\cline{2-13}\n')
00283                 tex_table.write(' & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} & {} \\\\ \\hline\n'.format('{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}', '{$r^2_{xy}$}', '{$r^2_{yx}$}'))
00284                 for gm in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00285                     if gm == 'andh':
00286                         tw = '\\glsentryshort{andh}'
```

```
00287                     else:
00288                         tw = '\\glsentryshort{{{}}}'.format(gm)
00289                     tex_table.write('{} '.format(tw))
00290                     for chm in ['rmsd', 'angl', 'andh', 'and', 'xor', 'pt']:
00291                         val2 = full_dict[db_name][guid_m][gm][chm][1]
00292                         val3 = full_dict[db_name][guid_m][gm][chm][2]
00293 
00294                         if abs(val2) > 99.999:
00295                             tex_table.write(' & {{$<-99$}} ')
00296                         elif abs(val2) > 10.0:
00297                             tex_table.write(' & {{${}$}} '.format(int(round(val2))))
00298                         else:
00299                             tex_table.write(' & {:3.2f} '.format(val2))
00300 
00301                         if abs(val3) > 99.999:
00302                             tex_table.write(' & {{$<-99$}} ')
00303                         elif abs(val3) > 10.0:
00304                             tex_table.write(' & {{${}$}} '.format(int(round(val3))))
00305                         else:
00306                             tex_table.write(' & {:3.2f} '.format(val3))
00307                         # tex_table.write(' & {:3.2f} & {:3.2f} & {:3.2f} '.format(main_dict[db_name][gm][chm][0],
    main_dict[db_name][gm][chm][1], main_dict[db_name][gm][chm][2]))
00308                     tex_table.write('\\\\ \\hline\n')
00309 
00310                 db_name1 = db_name.split('.')[0]
00311                 pr_1 = db_name1.split('_')[2]
00312                 ff_2 = db_name1.split('_')[1]
00313                 if pr_1 == 'trp':
00314                     if '2' in db_name:
00315                         tex_table.writelines(['\\end {tabular}\n', '\\label {{det_{}_{}}}\n'.format(guid_m, db_name1),
00316                                               '\\caption{{{}}}\n'.format(
00317                                                   'Determination coefficients among metrics and potential energy for the second simulation of
    \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field for \\glsentryshort{{{}}} guide metric.'.format(
00318                                                       pr_1, ff_2, guid_m)), '\\end {table}\n'])
00319                     else:
00320                         tex_table.writelines(['\\end {tabular}\n', '\\label {{det_{}_{}}}\n'.format(guid_m, db_name1),
00321                                               '\\caption{{{}}}\n'.format(
00322                                                   'Determination coefficients among metrics and potential energy for the first simulation of
    \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field for \\glsentryshort{{{}}} guide metric.'.format(
00323                                                       pr_1, ff_2, guid_m)), '\\end {table}\n'])
00324                 else:
00325                     tex_table.writelines(['\\end {tabular}\n', '\\label {{det_{}_{}}}\n'.format(guid_m, db_name1),
00326                                           '\\caption{{{}}}\n'.format(
00327                                               'Determination coefficients among metrics and potential energy for simulation of
    \\glsentryshort{{{}}} protein with \\glsentryshort{{{}}} force field for \\glsentryshort{{{}}} guide metric.'.format(
00328                                                   pr_1, ff_2, guid_m)), '\\end {table}\n'])
00329                 tex_table.write('\n\n\n')
00330         tex_table.write('\\end{landscape}')
00331 
00332 
00333 
00334 
00335 def myr(y, f):
00336     SSres = sum(map(lambda x: (x[0] - x[1]) ** 2, zip(y, f)))
00337     SStot = sum([(x - np.mean(y)) ** 2 for x in y])
00338     return 1-(SSres/SStot)
00339 
00340 
00341 def myr_rev(y, f):
00342     SSres = sum(map(lambda x: (x[0] - x[1]) ** 2, zip(y, f)))
00343     SStot = sum([(x - np.mean(f)) ** 2 for x in f])
00344     return 1-(SSres/SStot)
00345 
00346 
00347 def fill_stat_dict(filenames_db, legend_names, guide_metr):
00348     global main_dict, full_dict
00349     con_arr = [lite.connect(db_name, check_same_thread=False, isolation_level=None) for db_name in filenames_db]
00350     cur_arr = [con.cursor() for con in con_arr]
00351 
00352     print('Working with ', filenames_db, ' guide metr: ', guide_metr)
00353     qry = "select a.name from main_storage a where a.{0}_goal_dist= ( select min(b.{0}_goal_dist) from main_storage b)".format(guide_metr)
00354     result_arr = [cur.execute(qry) for cur in cur_arr]
00355     fetched_one_arr = [res.fetchone() for res in result_arr]
00356     names = [all_res[0] for all_res in fetched_one_arr]
00357     spnames = [name.split('_') for name in names]
00358     all_prev_names_s = [['\'{}\"'.format('_'.join(spname[:i])) for i in range(1, len(spname)+1)] for spname in spnames]
00359     long_lines = ["', "'.join(all_prev_names) for all_prev_names in all_prev_names_s]
00360     qrys = ["select a.rmsd_goal_dist, a.angl_goal_dist, a.andh_goal_dist, a.and_goal_dist, a.xor_goal_dist, a.rmsd_tot_dist, a.angl_tot_dist,
    a.andh_tot_dist, a.and_tot_dist, a.xor_tot_dist, a.name, a.hashed_name from main_storage a where a.name in ( {1} ) order by
    a.id".format(guide_metr, long_line) for long_line in long_lines]
00361     result_arr = list()
```

```
00362     for i, cur in enumerate(cur_arr):
00363         result_arr.append(cur.execute(qrys[i]))
00364     fetched_all_arr = [res.fetchall() for res in result_arr]
00365 
00366     rmsd_dist_arr = [[dist[0] for dist in goal_dist] for goal_dist in fetched_all_arr]
00367     angl_dist_arr = [[dist[1] for dist in goal_dist] for goal_dist in fetched_all_arr]
00368     andh_dist_arr = [[dist[2] for dist in goal_dist] for goal_dist in fetched_all_arr]
00369     and_dist_arr = [[dist[3] for dist in goal_dist] for goal_dist in fetched_all_arr]
00370     xor_dist_arr = [[dist[4] for dist in goal_dist] for goal_dist in fetched_all_arr]
00371 
00372     goal_dist = [rmsd_dist_arr, angl_dist_arr, andh_dist_arr, and_dist_arr, xor_dist_arr]
00373     metrics = ['rmsd', 'angl', 'andh', 'and', 'xor']
00374     # metr_units = {'rmsd': 'Å', 'angl': 'n/a', 'andh': 'contacts', 'and': 'contacts', 'xor': 'contacts'}
00375 
00376     # with open('correlation.tex', 'a+') as tex_table:
00377 
00378 
00379     print('Guide metric {}'.format(guide_metr))
00380 
00381     for j in range(len(goal_dist[0])):  # iterate over dbs
00382         for i, dist_arr in enumerate(goal_dist):  # iterate over metric
00383             # tex_table.writelines(['\n\\begin{table}[h]\n', '\centering\n', '\sisetup{table-align-text-post=false}\n',
      '\\begin{tabular}{@{}|l|l|\
00384             #
          |S[table-format=3.5]\
00385             #
          |S[table-format=3.5]\
00386             #
          |S[table-format=3.5]\
00387             #
          |@{}}\n'])
00388             # tex_table.write(
00389             #     '{} & {} & {} & {} & {}  \\\\ \\hline\n'.format('{metric_x}', '{metric_y}', '{corr_xy}', '{det_xy}', '{det_yx}'))
00390 
00391             prot_name, ff = legend_names[j].split(' ')
00392             rn = None
00393             if '_' in ff:
00394                 ff, rn = ff.split('_')
00395             path_to_ener = "/home/vanya/Documents/Phillips/GMDA/Latest_results"
00396             path_to_ener1 = os.path.join(path_to_ener, prot_name)
00397             if rn is not None:
00398                 path_to_ener1 = os.path.join(path_to_ener1, "run_{}".format(rn))
00399 
00400             print('Prot: {} ff: {} run: {}'.format(prot_name, ff, rn if rn is not None else 'n/a'))
00401 
00402             # Reduced correlation matrices
00403             a = np.asarray(goal_dist[metrics.index(guide_metr)][j])
00404             a = (a - a.min()) / (a.max() - a.min())
00405             b = np.asarray(goal_dist[i][j])
00406             b = (b - b.min()) / (b.max() - b.min())
00407             main_dict[filenames_db[j]][guide_metr][metrics[i]][0] = np.corrcoef(a, b)[0][1]
00408             main_dict[filenames_db[j]][guide_metr][metrics[i]][1] = r2_score(a, b)
00409             main_dict[filenames_db[j]][guide_metr][metrics[i]][2] = r2_score(b, a)
00410 
00411             if metrics.index(guide_metr) == i:
00412                 loc_len = len(goal_dist[i][j])
00413 
00414                 path_to_ener2 = os.path.join(path_to_ener1, ff, 'PT_energy')
00415                 np_ener_file = os.path.join(path_to_ener2, '{}_correct_index_energy.npy'.format(guide_metr))
00416                 ener_arr = np.load(np_ener_file).swapaxes(0, 1)[1]
00417                 ener_arr = ener_arr[-loc_len:]  # trim, so we have same number of steps
00418                 a = np.asarray(ener_arr)
00419                 a = (a - a.min()) / (a.max() - a.min())
00420                 b = np.asarray(goal_dist[i][j])
00421                 b = (b - b.min()) / (b.max() - b.min())
00422 
00423                 if main_dict[filenames_db[j]][metrics[i]]['pt'][0] != 0:
00424                     print('warning here')
00425                 if main_dict[filenames_db[j]][metrics[i]]['pt'][1] != 0:
00426                     print('warning here')
00427                 if main_dict[filenames_db[j]][metrics[i]]['pt'][2] != 0:
00428                     print('warning here')
00429 
00430                 main_dict[filenames_db[j]][guide_metr]['pt'][0] = np.corrcoef(a, b)[0][1]
00431                 main_dict[filenames_db[j]][guide_metr]['pt'][1] = r2_score(a, b)
00432                 main_dict[filenames_db[j]][guide_metr]['pt'][2] = r2_score(b, a)
00433 
00434 
00435             # Full correlation matrices
00436 
00437             for k in range(len(goal_dist)):
```

```
00438                 # if i != k:
00439                 a = np.asarray(goal_dist[i][j])
00440                 a = (a - a.min()) / (a.max() - a.min())
00441                 b = np.asarray(goal_dist[k][j])
00442                 b = (b - b.min()) / (b.max() - b.min())
00443                 full_dict[filenames_db[j]][guide_metr][metrics[i]][metrics[k]][0] = np.corrcoef(a, b)[0][1]
00444                 full_dict[filenames_db[j]][guide_metr][metrics[i]][metrics[k]][1] = r2_score(a, b)
00445                 full_dict[filenames_db[j]][guide_metr][metrics[i]][metrics[k]][2] = r2_score(b, a)
00446 
00447             loc_len = len(goal_dist[i][j])
00448 
00449             path_to_ener2 = os.path.join(path_to_ener1, ff, 'PT_energy')
00450             np_ener_file = os.path.join(path_to_ener2, '{}_correct_index_energy.npy'.format(guide_metr))
00451             ener_arr = np.load(np_ener_file).swapaxes(0, 1)[1]
00452             ener_arr = ener_arr[-loc_len:]  # trim, so we have same number of steps
00453             a = np.asarray(ener_arr)
00454             a = (a - a.min()) / (a.max() - a.min())
00455             b = np.asarray(goal_dist[i][j])
00456             b = (b - b.min()) / (b.max() - b.min())
00457 
00458             full_dict[filenames_db[j]][guide_metr][metrics[i]]['pt'][0] = np.corrcoef(a, b)[0][1]
00459             full_dict[filenames_db[j]][guide_metr][metrics[i]]['pt'][1] = r2_score(a, b)
00460             full_dict[filenames_db[j]][guide_metr][metrics[i]]['pt'][2] = r2_score(b, a)
00461 
00462 
00463 
00464 if __name__ == '__main__':
00465     main()
00466 # from scipy.stats import pearsonr
```

4.5 compute_sincos_dist.py File Reference

Namespaces

- compute_sincos_dist

Functions

- def compute_sincos_dist.compute_sincos_dist (num_el, filename_nat='sincos_goal.dat', filename_check='sincos_bb_300.dat')

4.6 compute_sincos_dist.py

```
00001 import numpy as np
00002 import matplotlib.pyplot as plt
00003 from matplotlib.figure import figaspect
00004 import numpy as np
00005 from functools import reduce
00006 
00007 
00008 def compute_sincos_dist(num_el, filename_nat = 'sincos_goal.dat', filename_check = 'sincos_bb_300.dat'):
00009     with open(filename_nat, 'rb') as file:
00010         initial_1d_array = np.frombuffer(file.read(), dtype=np.float64, count=-1)
00011     nat_arr = np.reshape(initial_1d_array, (-1, num_el*2))
00012     with open(filename_check, 'rb') as file:
00013         initial_1d_array = np.frombuffer(file.read(), dtype=np.float64, count=-1)
00014     check_arr = np.reshape(initial_1d_array, (-1, num_el*2))
00015     del initial_1d_array
00016 
00017     res_arr = [None]*check_arr.shape[0]
00018     for i in range(check_arr.shape[0]):
00019         res_arr[i] = np.sum(abs(check_arr[i] - nat_arr))
00020     # res_arr = [res_arr[i*2] + res_arr[i*2+1] for i in range(len(res_arr)/2)]
00021 
00022     max_val = max(res_arr)
00023     min_val = min(res_arr)
00024     fig_num = 0
00025     mdpi = 400
00026     major_xticks = None
00027     minor_xticks = None
00028     major_yticks = None
00029     minor_yticks = None
00030     w, h = figaspect(0.5)
00031     fig = plt.figure(fig_num, figsize=(w, h))
00032     plt.xlim(0, len(res_arr))
00033     ax = fig.gca()
00034     major_xticks = np.arange(0, len(res_arr) + len(res_arr) / 10, len(res_arr) / 10)
00035     major_yticks = np.arange(min_val, max_val + max_val / 16, (max_val - min_val) / 16)
00036     if major_xticks is not None:
00037         ax.set_xticks(major_xticks)
00038     if minor_xticks is not None:
```

```
00039         ax.set_xticks(minor_xticks, minor=True)
00040     if major_yticks is not None:
00041         ax.set_yticks(major_yticks)
00042     if minor_yticks is not None:
00043         ax.set_yticks(minor_yticks, minor=True)
00044     plt.grid(which='both')
00045     lines = []
00046 
00047     line, = plt.plot(range(len(res_arr)), res_arr, '-', markersize=1)
00048     lines.append(line)
00049     ax.legend(lines, 'full cont')
00050     plt.xlabel("frame")
00051     plt.ylabel("sin/cos")
00052     plt.title('sin/cos (difference, error) for 20ns gb1 simulatoin and goal at 300K (lower is better)')
00053     plt.savefig('sincos_20ns_300.png', dpi=mdpi)
00054 
00055 compute_sincos_dist(110, 'sincos_goal.dat')
```

4.7 concat_all_xtc.py File Reference

Namespaces

- concat_all_xtc

Functions

- def concat_all_xtc.get_all_xtc (past_dir)

Variables

- int concat_all_xtc.elem_at_once = 128
- def concat_all_xtc.all_xtc = get_all_xtc('./past/')
- int concat_all_xtc.tot_iter = 0
- int concat_all_xtc.cur_name = 0
- concat_all_xtc.new_names = list()
- def concat_all_xtc.cur_files = all_xtc[tot_iter:tot_iter+elem_at_once]
- concat_all_xtc.f
- concat_all_xtc.o
- concat_all_xtc.n
- concat_all_xtc.new_names1 = list()
- concat_all_xtc.new_names2 = list()
- concat_all_xtc.new_names3 = list()

4.8 concat_all_xtc.py

```
00001 #!/usr/bin/env python3
00002 
00003 import os
00004 from gmx_wrappers import gmx_trjcat
00005 
00006 def get_all_xtc(past_dir):
00007     filenames_found = [f.split("/")[-1] for f in os.listdir(past_dir)]
00008     filenames_found_important = [f for f in filenames_found if f.split('.')[1] == 'xtc']
00009     del filenames_found
00010     print('Found files: {} with .xtc'.format(len(filenames_found_important)))
00011     return filenames_found_important
00012 
00013 elem_at_once = 128
00014 
00015 all_xtc = get_all_xtc('./past/')
00016 all_xtc.sort()
00017 with open('index_file.txt', 'w') as f:
00018     for elem in all_xtc:
00019         f.write("{}\n".format(elem))
00020 
00021 tot_iter = 0
00022 cur_name = 0
00023 new_names = list()
00024 while tot_iter < len(all_xtc):
00025     cur_files = all_xtc[tot_iter:tot_iter+elem_at_once]
00026     tot_iter += elem_at_once
00027     cur_name += 1
00028     gmx_trjcat(f=[os.path.join('./past', file) for file in cur_files], o=str(cur_name), n=None)
00029     new_names.append(str(cur_name))
00030 
00031 if len(new_names) > 1:
00032     tot_iter = 0
00033     cur_name = 0
```

```
00034     new_names1 = list()
00035     while tot_iter < len(new_names):
00036         cur_files = new_names[tot_iter:tot_iter + elem_at_once]
00037         tot_iter += elem_at_once
00038         cur_name += 1
00039         gmx_trjcat(f=cur_files, o='a' + str(cur_name), n=None)
00040         new_names1.append('a' + str(cur_name))
00041 else:
00042     os.rename(new_names[0], 'final_fat.xtc')
00043     exit('Done')
00044
00045 for file in new_names:
00046     os.remove('./{}.xtc'.format(file))
00047
00048 if len(new_names1) > 1:
00049     tot_iter = 0
00050     cur_name = 0
00051     new_names2 = list()
00052     while tot_iter < len(new_names):
00053         cur_files = new_names1[tot_iter:tot_iter + elem_at_once]
00054         tot_iter += elem_at_once
00055         cur_name += 1
00056         gmx_trjcat(f=cur_files, o='b' + str(cur_name), n=None)
00057         new_names2.append('b' + str(cur_name))
00058 else:
00059     os.rename(new_names1[0], 'final_fat.xtc')
00060     exit('Done')
00061
00062 for file in new_names1:
00063     os.remove('./{}.xtc'.format(file))
00064
00065 if len(new_names2) > 1:
00066     tot_iter = 0
00067     cur_name = 0
00068     new_names3 = list()
00069     while tot_iter < len(new_names):
00070         cur_files = new_names2[tot_iter:tot_iter + elem_at_once]
00071         tot_iter += elem_at_once
00072         cur_name += 1
00073         gmx_trjcat(f=cur_files, o='c' + str(cur_name), n=None)
00074         new_names3.append('c' + str(cur_name))
00075 else:
00076     os.rename(new_names2[0], 'final_fat.xtc')
00077     exit('Done')
00078
00079
00080 if len(new_names3) > 1:
00081     print('Need more iterations!')
00082 else:
00083     os.rename(new_names3[0], 'final_fat.xtc')
00084
00085 for file in new_names2:
00086     os.remove('./{}.xtc'.format(file))
```

4.9 convert_bad_db.py File Reference

Namespaces

- convert_bad_db

Functions

- def convert_bad_db.get_db_con (db_name='fixed_db', tot_seeds=4)

Variables

- string convert_bad_db.in_db = "results_opls_trp_300"
- convert_bad_db.con_bad = lite.connect(in_db+'.sqlite3', check_same_thread=False, isolation_level=None)
- convert_bad_db.cur_bad = con_bad.cursor()
- string convert_bad_db.qry = "SELECT rmsd_goal_dist, rmsd_prev_dist, rmsd_tot_dist, angl_goal_dist, angl_prev_dist, angl_tot_dist, andh_goal↩
  _dist," \
- convert_bad_db.res = cur_bad.execute(qry)
- convert_bad_db.res_first = res.fetchone()
- convert_bad_db.res_arr = res.fetchall()
- convert_bad_db.log_res = res.fetchall()
- convert_bad_db.vis_res = res.fetchall()
- convert_bad_db.good_arr = list()
- convert_bad_db.elem = res_first
- def convert_bad_db.con_fixed = get_db_con(in_db+'_fixed')
- def convert_bad_db.cur_good = con_fixed.cursor()

4.10 convert_bad_db.py

```
00001 import os
00002 import sqlite3 as lite
00003 import numpy as np
00004 import struct
00005 lite.register_adapter(np.int64, lambda val: int(val))
00006 lite.register_adapter(np.int32, lambda val: int(val))
00007 lite.register_adapter(np.float, lambda val: float(val))
00008 lite.register_adapter(np.float32, lambda val: float(val))
00009
00010
00011 def get_db_con(db_name ='fixed_db', tot_seeds=4):
00012     counter = 0
00013     # db_path = '/dev/shm/GMDApy'
00014     db_path = os.getcwd()
00015     full_path = os.path.join(db_path, db_name + '.sqlite3')
00016
00017     con = lite.connect(full_path, check_same_thread=False, isolation_level=None)
00018
00019     cur = con.cursor()
00020     cur.execute("""CREATE TABLE main_storage (
00021         id               INTEGER   PRIMARY KEY AUTOINCREMENT,
00022
00023         rmsd_goal_dist   FLOAT    NOT NULL,
00024         rmsd_prev_dist   FLOAT    NOT NULL,
00025         rmsd_tot_dist    FLOAT    NOT NULL,
00026
00027         angl_goal_dist   FLOAT    NOT NULL,
00028         angl_prev_dist   FLOAT    NOT NULL,
00029         angl_tot_dist    FLOAT    NOT NULL,
00030
00031         andh_goal_dist   INTEGER   NOT NULL,
00032         andh_prev_dist   INTEGER   NOT NULL,
00033         andh_tot_dist    INTEGER   NOT NULL,
00034
00035         and_goal_dist    INTEGER   NOT NULL,
00036         and_prev_dist    INTEGER   NOT NULL,
00037         and_tot_dist     INTEGER   NOT NULL,
00038
00039         xor_goal_dist    INTEGER   NOT NULL,
00040         xor_prev_dist    INTEGER   NOT NULL,
00041         xor_tot_dist     INTEGER   NOT NULL,
00042
00043         curr_gc          INTEGER   NOT NULL,
00044         Timestamp        DATETIME DEFAULT (CURRENT_TIMESTAMP),
00045         hashed_name      CHAR (32) NOT NULL UNIQUE,
00046         name             TEXT
00047         );""")
00048     con.commit()
00049     cur.execute("""CREATE TABLE visited (
00050         vid        INTEGER   PRIMARY KEY AUTOINCREMENT, \
00051         id        REFERENCES main_storage (id),
00052         cur_gc    INTEGER,
00053         Timestamp DATETIME DEFAULT (CURRENT_TIMESTAMP)
00054     );""")
00055     con.commit()
00056
00057     add_ind_q = 'CREATE INDEX viz_id_idx ON visited (id);'
00058     cur.execute(add_ind_q)
00059     con.commit()
00060
00061     # id         REFERENCES main_storage (id), \
00062     init_query = 'CREATE TABLE log ( \
00063         lid        INTEGER   PRIMARY KEY AUTOINCREMENT, \
00064         operation  INTEGER, \
00065         id         INTEGER, \
00066         src        CHAR (8), \
00067         dst        CHAR(8), \
00068         cur_metr   CHAR(5), \
00069         gc         INTEGER , \
00070         mul        FLOAT, \
00071         bsfr        FLOAT, \
00072         bsfn        FLOAT, \
00073         bsfh        FLOAT, \
00074         bsfa        FLOAT, \
00075         bsfx        FLOAT, \
00076         Timestamp  DATETIME DEFAULT (CURRENT_TIMESTAMP)'  # no this is not an error
00077     for i in range(tot_seeds):
00078         init_query += ", \
00079         dist_from_prev_{0} FLOAT, \
```

```
00080         dist_to_goal_{0}   FLOAT ".format(i+1)
00081     init_query += ');'
00082 
00083     cur.execute(init_query)
00084     con.commit()
00085     add_ind_q = 'CREATE INDEX log_id_idx ON log (id);'
00086     cur.execute(add_ind_q)
00087     con.commit()
00088 
00089     cur.execute('PRAGMA mmap_size=-64000')  # 32M
00090     cur.execute('PRAGMA journal_mode =  OFF')
00091     cur.execute('PRAGMA synchronous = OFF')
00092     cur.execute('PRAGMA temp_store = MEMORY')
00093     cur.execute('PRAGMA threads = 32')
00094 
00095     return con
00096 
00097 in_db = "results_opls_trp_300"
00098 
00099 con_bad = lite.connect(in_db+'.sqlite3', check_same_thread=False, isolation_level=None)
00100 
00101 
00102 cur_bad = con_bad.cursor()
00103 
00104 qry = "SELECT rmsd_goal_dist, rmsd_prev_dist, rmsd_tot_dist, angl_goal_dist, angl_prev_dist, angl_tot_dist, andh_goal_dist," \
00105 " andh_prev_dist, andh_tot_dist, and_goal_dist, and_prev_dist, and_tot_dist, xor_goal_dist, xor_prev_dist, xor_tot_dist, curr_gc, " \
00106 "Timestamp, hashed_name,  name   FROM main_storage;"
00107 
00108 res = cur_bad.execute(qry)
00109 res_first = res.fetchone()
00110 res_arr = res.fetchall()
00111 
00112 qry = "SELECT lid, operation, id, src, dst, cur_metr, gc, mul, bsfr, bsfn, bsfh, bsfa, bsfx, Timestamp, dist_from_prev_1, dist_to_goal_1,
       dist_from_prev_2, dist_to_goal_2, dist_from_prev_3, dist_to_goal_3, dist_from_prev_4, dist_to_goal_4 FROM log;"
00113 res = cur_bad.execute(qry)
00114 log_res = res.fetchall()
00115 qry = "SELECT vid, id, cur_gc, Timestamp FROM visited;"
00116 res = cur_bad.execute(qry)
00117 vis_res = res.fetchall()
00118 
00119 con_bad.close()
00120 
00121 good_arr = list()
00122 elem = res_first
00123 good_arr.append(tuple([elem[0], 0, 0, elem[3], 0, 0,
00124                     struct.unpack('i', elem[6])[0], 0, 0, struct.unpack('i', elem[9])[0], 0, 0, struct.unpack('i', elem[12])[0], 0, 0,
00125                     elem[15], elem[16], elem[17], elem[18]]))
00126 
00127 for elem in res_arr:
00128     good_arr.append(tuple([elem[0], elem[1], elem[2], elem[3], elem[4], elem[5],
00129                     struct.unpack('Q', elem[6])[0], struct.unpack('Q', elem[7])[0], struct.unpack('Q', elem[8])[0], struct.unpack('Q',
       elem[9])[0], struct.unpack('Q', elem[10])[0],
00130                     struct.unpack('Q', elem[11])[0], struct.unpack('Q', elem[12])[0], struct.unpack('Q', elem[13])[0], struct.unpack('Q',
       elem[14])[0],
00131                     elem[15], elem[16], elem[17], elem[18]]))
00132 
00133 qry = "INSERT INTO main_storage ( rmsd_goal_dist, rmsd_prev_dist, rmsd_tot_dist, angl_goal_dist, angl_prev_dist, angl_tot_dist,
       andh_goal_dist," \
00134 " andh_prev_dist, andh_tot_dist, and_goal_dist, and_prev_dist, and_tot_dist, xor_goal_dist, xor_prev_dist, xor_tot_dist, curr_gc, Timestamp,
       hashed_name,  name  ) " \
00135       "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);"
00136 
00137 con_fixed = get_db_con(in_db+'_fixed')
00138 cur_good = con_fixed.cursor()
00139 cur_good.executemany(qry, good_arr)
00140 con_fixed.commit()
00141 cur_good.executemany("INSERT INTO log VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", log_res)
00142 con_fixed.commit()
00143 cur_good.executemany("INSERT INTO visited VALUES (?, ?, ?, ?)", vis_res)
00144 con_fixed.commit()
00145 con_fixed.close()
00146 
```

4.11 db_proc.py File Reference

Namespaces

- db_proc

Functions

- tuple db_proc.get_db_con (int tot_seeds=4)

  Creates the database with structure that fits exact number of seeds.
- NoReturn db_proc.log_error (lite.Connection con, str type, int id)

  Writes an error message into the log table.
- int db_proc.get_id_for_hash (lite.Connection con, str h_name)

  Searches main storage for id with given hash.
- tuple db_proc.get_corr_vid_for_id (lite.Connection con, int max_id, list prev_ids, float last_gc)

  Used for recovery procedure.
- int db_proc.get_corr_lid_for_id (lite.Connection con, int next_id, int vid_ts, int last_vis_id)

  Used for recovery procedure.
- list db_proc.get_all_hashed_names (lite.Connection con)

  Fetches all hashes from the main_storage.
- NoReturn db_proc.insert_into_main_stor (lite.Connection con, dict node_info, int curr_gc, str digest_name, str name)

  Inserts main information into the DB.
- NoReturn db_proc.insert_into_visited (lite.Connection con, str hname, int gc)

  Inserts node processing event.
- NoReturn db_proc.insert_into_log (lite.Connection con, str operation, str hname, str src, str dst, list bsf, int gc, float mul, list prev_arr, list goal_arr, str cur_metr_name)

  Inserts various information, like new best_so_far events, insertions into the open queue, etc.
- NoReturn db_proc.copy_old_db (list main_dict_keys, list last_visited, str next_in_oq, float last_gc)

  Used during the recovery procedure.

4.12 db_proc.py

```
00001 """
00002 This file contains DB related functions.
00003 .. module:: GMDA_main
00004     :platform: linux
00005
00006 .. moduleauthor:: Ivan Syzonenko <is2k@mtmail.mtsu.edu>
00007 """
00008 __license__ = "MIT"
00009 __docformat__ = 'reStructuredText'
00010
00011 import os
00012 import sqlite3 as lite
00013 import numpy as np
00014 lite.register_adapter(np.int64, lambda val: int(val))
00015 lite.register_adapter(np.int32, lambda val: int(val))
00016 lite.register_adapter(np.float, lambda val: float(val))
00017 lite.register_adapter(np.float32, lambda val: float(val))
00018 # import numpy as np
00019 from typing import NoReturn, Mapping, Sequence, List, Set
00020
00021
00022 def get_db_con(tot_seeds: int = 4) -> tuple:
00023     """Creates the database with structure that fits exact number of seeds.
00024
00025     Filename for DB is generated as next number after the highest consequent found.
00026     If there is results_0.sqlite3, then next will be results_1.sqlite3 if it did not exist.
00027
00028     Args:
00029         :param int tot_seeds: number of seeds used in the current run
00030         :type tot_seeds: int
00031
00032     Returns:
00033         :return: database connection and name
00034
00035     Connection to the new database and it's name.
00036     """
00037     counter = 0
00038     # db_path = '/dev/shm/GMDApy'
00039     db_path = os.getcwd()
00040     db_name = 'results_{}.sqlite3'.format(counter)
00041     full_path = os.path.join(db_path, 'results_{}.sqlite3'.format(counter))
00042     while os.path.exists(full_path):
00043         counter += 1
00044         full_path = os.path.join(db_path, 'results_{}.sqlite3'.format(counter))
00045
```

```
00046     con = lite.connect(full_path, check_same_thread=False, isolation_level=None)
00047
00048     cur = con.cursor()
00049     cur.execute("""CREATE TABLE main_storage (
00050         id               INTEGER   PRIMARY KEY AUTOINCREMENT,
00051
00052         rmsd_goal_dist   FLOAT    NOT NULL,
00053         rmsd_prev_dist   FLOAT    NOT NULL,
00054         rmsd_tot_dist    FLOAT    NOT NULL,
00055
00056         angl_goal_dist   FLOAT    NOT NULL,
00057         angl_prev_dist   FLOAT    NOT NULL,
00058         angl_tot_dist    FLOAT    NOT NULL,
00059
00060         andh_goal_dist   INTEGER   NOT NULL,
00061         andh_prev_dist   INTEGER   NOT NULL,
00062         andh_tot_dist    INTEGER   NOT NULL,
00063
00064         and_goal_dist    INTEGER   NOT NULL,
00065         and_prev_dist    INTEGER   NOT NULL,
00066         and_tot_dist     INTEGER   NOT NULL,
00067
00068         xor_goal_dist    INTEGER   NOT NULL,
00069         xor_prev_dist    INTEGER   NOT NULL,
00070         xor_tot_dist     INTEGER   NOT NULL,
00071
00072         curr_gc          INTEGER   NOT NULL,
00073         Timestamp        DATETIME DEFAULT (CURRENT_TIMESTAMP),
00074         hashed_name      CHAR (32) NOT NULL UNIQUE,
00075         name             TEXT
00076         );""")
00077     con.commit()
00078     cur.execute("""CREATE TABLE visited (
00079         vid        INTEGER   PRIMARY KEY AUTOINCREMENT, \
00080         id        REFERENCES main_storage (id),
00081         cur_gc    INTEGER,
00082         Timestamp DATETIME DEFAULT (CURRENT_TIMESTAMP)
00083     );""")
00084     con.commit()
00085
00086     add_ind_q = 'CREATE INDEX viz_id_idx ON visited (id);'
00087     cur.execute(add_ind_q)
00088     con.commit()
00089
00090     # id         REFERENCES main_storage (id), \
00091     init_query = 'CREATE TABLE log ( \
00092         lid        INTEGER   PRIMARY KEY AUTOINCREMENT, \
00093         operation  INTEGER, \
00094         id         INTEGER, \
00095         src        CHAR (8), \
00096         dst        CHAR(8), \
00097         cur_metr   CHAR(5), \
00098         gc         INTEGER , \
00099         mul        FLOAT, \
00100         bsfr        FLOAT, \
00101         bsfn        FLOAT, \
00102         bsfh        FLOAT, \
00103         bsfa        FLOAT, \
00104         bsfx        FLOAT, \
00105         Timestamp  DATETIME DEFAULT (CURRENT_TIMESTAMP)'  # no this is not an error
00106     for i in range(tot_seeds):
00107         init_query += ", \
00108         dist_from_prev_{0} FLOAT, \
00109         dist_to_goal_{0}   FLOAT ".format(i+1)
00110     init_query += ');'
00111
00112     cur.execute(init_query)
00113     con.commit()
00114     add_ind_q = 'CREATE INDEX log_id_idx ON log (id);'
00115     cur.execute(add_ind_q)
00116     con.commit()
00117
00118     cur.execute('PRAGMA mmap_size=-64000')  # 32M
00119     cur.execute('PRAGMA journal_mode =  OFF')
00120     cur.execute('PRAGMA synchronous = OFF')
00121     cur.execute('PRAGMA temp_store = MEMORY')
00122     cur.execute('PRAGMA threads = 32')
00123
00124     return con, db_name
00125
00126
```

```
00127 def log_error(con: lite.Connection, type: str, id: int) -> NoReturn:
00128     """Writes an error message into the log table
00129 
00130     Args:
00131         :param con: current DB connection
00132         :param type: error type
00133         :param id: id associated with the error
00134 
00135     Returns:
00136     Adds one row in the log table.
00137     """
00138     qry = 'INSERT INTO log (id, operation, dst)  VALUES ({}, "ERROR", "{}")'.format(id, type)
00139     try:
00140         con.cursor().execute(qry)
00141         con.commit()
00142     except Exception as e:
00143         print(e)
00144         print('Error in "log_error": {}'.format(qry))
00145 
00146 
00147 # def get_id_for_name(con, name):
00148 #     con.commit()
00149 #     qry = "SELECT id FROM main_storage WHERE name='{}'".format(name)
00150 #     cur = con.cursor()
00151 #     result = cur.execute(qry)
00152 #     num = int(result.fetchone()[0])
00153 #     if not isinstance(num, int):
00154 #         raise Exception("ID was not found in main stor")
00155 #     return num
00156 
00157 
00158 def get_id_for_hash(con: lite.Connection, h_name: str) -> int:
00159     """Searches main storage for id with given hash
00160 
00161     Args:
00162         :param lite.Connection con: DB connection
00163         :param str h_name: hashname to use during the search
00164 
00165     Returns:
00166         :return: id or None if not found
00167     """
00168     con.commit()
00169     qry = "SELECT id FROM main_storage WHERE hashed_name='{}'".format(h_name)
00170     cur = con.cursor()
00171     result = cur.execute(qry)
00172     row = result.fetchone()
00173     if row is not None:
00174         num = int(row[0])
00175     else:
00176         num = None
00177     # if not isinstance(num, int):
00178     #     print("ID was not found in main stor")
00179     return num
00180 
00181 
00182 def get_corr_vid_for_id(con: lite.Connection, max_id: int, prev_ids: list, last_gc: float) -> tuple:
00183     """Used for recovery procedure. Tries to find matching sequence of nodes in the visited table
00184 
00185     Args:
00186         :param lite.Connection con: DB connection
00187         :param int max_id: maximum value of the id (defined by previous search as the common latest id)
00188         :param list prev_ids: several ids that should match
00189         :param float last_gc: extra check, whether greed counters also match
00190 
00191     Returns:
00192         :return: last common visited id, timestamp, and id
00193         :rtype: tuple
00194     """
00195     qry = "SELECT vid, id, CAST(strftime('%s', Timestamp) AS INT), cur_gc FROM visited WHERE id<'{}' AND id in ({}, {}, {}) order by vid
       desc".format(max_id, prev_ids[0], prev_ids[1], prev_ids[2])
00196     cur = con.cursor()
00197     result = cur.execute(qry)
00198     rows = result.fetchall()
00199     i = 0
00200     while i+2 < len(rows):  # 3 for next version
00201         if rows[i][0] - rows[i+1][0] == 1 and rows[i+1][0] - rows[i+2][0] == 1:
00202             break
00203         i += 1
00204     if i+2 >= len(rows):
00205         raise Exception("Sequence of events from pickle dump not found in DB")
00206     last_good_vid = rows[i][0]
```

```
00207     last_good_ts = rows[i][2]
00208     last_good_id = rows[i][1]
00209     if last_gc != int(rows[i][3]):
00210         raise Exception('Everything looked good, but greed counters did not match.\n Check manually and comment this exception if you are sure
      that this is normal.\n')
00211
00212     return last_good_vid, last_good_ts, last_good_id
00213
00214
00215 def get_corr_lid_for_id(con: lite.Connection, next_id: int, vid_ts: int, last_vis_id: int) -> int:
00216     """
00217     Used for recovery procedure. Tries to find matching sequence of nodes in the log table
00218
00219     Args:
00220         :param lite.Connection con: DB connection
00221         :param int next_id: next id we expect to see in the log, used for double check
00222         :param int vid_ts: visited timestampt
00223         :param int last_vis_id: last visited id
00224
00225     Returns:
00226         :return: the latest valid log_id
00227     """
00228     qry = "SELECT lid, CAST(strftime('%s', Timestamp) AS INT) FROM log WHERE id='{}' AND src='WQ' AND dst='VIZ' order by
    lid".format(last_vis_id)
00229     cur = con.cursor()
00230     result = cur.execute(qry)
00231     rows = result.fetchall()
00232     if len(rows) > 1:
00233         # find the smallest dist between vid_ts and all ts
00234         dist = abs(rows[0][1] - vid_ts)
00235         good_lid = int(rows[0][0])
00236         i = 1
00237         while i < len(rows):
00238             if abs(rows[i][1] - vid_ts) <= dist:
00239                 dist = abs(rows[i][1] - vid_ts)
00240                 good_lid = int(rows[i][0])
00241             i += 1
00242     else:
00243         good_lid = int(rows[0][0])
00244
00245     # so now we have good_lid which is very close, but may be not exact
00246
00247     qry = "SELECT lid, operation, id, src, dst FROM log WHERE lid > {} order by lid limit 4".format(good_lid)
00248     result = cur.execute(qry)
00249     rows = result.fetchall()
00250     i = 0
00251     if (rows[i][1] == 'current' and rows[i][4] == 'WQ') or rows[i][1] == 'skip':
00252         good_lid += 1
00253         i += 1
00254         if rows[i][1] == 'prom_O':
00255             good_lid += 1
00256             i += 1
00257
00258     if rows[i][1] == 'result' and rows[i][4] == 'VIZ' and int(rows[i][2]) == next_id:
00259         print("Log table ID computed perfectly.")
00260
00261     return good_lid
00262
00263
00264 # I am not using it
00265 # def get_max_id_from_main(con):
00266 #     qry = "SELECT max(id) FROM main_storage"
00267 #     cur = con.cursor()
00268 #     result = cur.execute(qry)
00269 #     row = result.fetchone()
00270 #     if row is not None:
00271 #         num = int(row[0])
00272 #     else:
00273 #         num = None
00274 #     return num
00275
00276
00277 def get_all_hashed_names(con: lite.Connection) -> list:
00278     """Fetches all hashes from the main_storage
00279
00280     Args:
00281         :param lite.Connection con: DB connection
00282
00283     Returns:
00284         :return: list of all hashes in the main_storage
00285         :rtype: list
```

```
00286     """
00287     qry = "SELECT hashed_name FROM main_storage order by id desc"
00288     cur = con.cursor()
00289     result = cur.execute(qry)
00290     rows = result.fetchall()
00291     return rows
00292
00293
00294 def insert_into_main_stor(con: lite.Connection, node_info: dict, curr_gc: int, digest_name: str, name: str) -> NoReturn:
00295     """Inserts main information into the DB.
00296
00297     Args:
00298         :param lite.Connection con: DB connection
00299         :param dict node_info: all metric values associated with the node
00300         :param int curr_gc: current greedy counter
00301         :param str digest_name: hash name for the path, same as filenames for MD simulations
00302         :param str name: path from the origin separated by _
00303
00304     Returns:
00305     Stores data in the DB in a main_storage table.
00306     """
00307     # con = lite.connect('results_8.sqlite3', timeout=300, check_same_thread=False, isolation_level=None)
00308     # qry = "INSERT OR IGNORE INTO main_storage(rmsd_goal_dist, rmsd_prev_dist, rmsd_tot_dist, angl_goal_dist,
00309     # angl_prev_dist, angl_tot_dist," \
00310     qry = "INSERT INTO main_storage(rmsd_goal_dist, rmsd_prev_dist, rmsd_tot_dist, angl_goal_dist, angl_prev_dist, angl_tot_dist," \
00311           "                         andh_goal_dist, andh_prev_dist, andh_tot_dist, and_goal_dist, and_prev_dist, and_tot_dist," \
00312           "                         xor_goal_dist, xor_prev_dist, xor_tot_dist, curr_gc, hashed_name, name) " \
00313           "VALUES (?, ?, ?, ?, ?, ?,   ?, ?, ?, ?, ?, ?,   ?, ?, ?, ?, ?, ?)"
00314     cur = con.cursor()
00315     try:
00316         cur.execute(qry, [str(elem) for elem in (node_info['RMSD_to_goal'], node_info['RMSD_from_prev'], node_info['RMSD_dist_total'],
00317                           node_info['ANGL_to_goal'], node_info['ANGL_from_prev'], node_info['ANGL_dist_total'],
00318                           node_info['AND_H_to_goal'], node_info['AND_H_from_prev'], node_info['AND_H_dist_total'],
00319                           node_info['AND_to_goal'], node_info['AND_from_prev'], node_info['AND_dist_total'],
00320                           node_info['XOR_to_goal'], node_info['XOR_from_prev'], node_info['XOR_dist_total'],
00321                           curr_gc, digest_name, name)])
00322         con.commit()
00323     except Exception as e:
00324         nid = get_id_for_hash(con, digest_name)
00325         log_error(con, 'MAIN', nid)
00326         qry = "SELECT * FROM main_storage WHERE id=?"
00327         cur = con.cursor()
00328         result = cur.execute(qry, nid)
00329         row = result.fetchone()
00330         print('Original elment in MAIN:', row)
00331         qry = "SELECT * FROM log WHERE id=?"
00332         cur = con.cursor()
00333         result = cur.execute(qry, nid)
00334         rows = result.fetchall()
00335         print('Printing all I found in the log about this ID:')
00336         for row in rows:
00337             print(row)
00338         print('Error element message: ', e, '\nqry: ', node_info, curr_gc, digest_name, name)
00339
00340
00341 def insert_into_visited(con: lite.Connection, hname: str, gc: int) -> NoReturn:
00342     """
00343     Inserts node processing event.
00344
00345     Args:
00346         :param lite.Connection con: DB connection
00347         :param str hname: hashname, same as MD filenames
00348         :param int gc: greedy counter
00349
00350     Returns:
00351     Stores data in the DB in a visited table.
00352     """
00353     nid = get_id_for_hash(con, hname)
00354     qry = 'INSERT INTO visited( id, cur_gc ) VALUES (?, ?)'
00355     cur = con.cursor()
00356     try:
00357         cur.execute(qry, (nid, gc))
00358         con.commit()
00359     except Exception as e:
00360         print(e, '\nqry: ', hname, gc)
00361         log_error(con, 'VIZ', nid)
00362
00363
00364 def insert_into_log(con: lite.Connection, operation: str, hname: str, src: str, dst: str, bsf: list, gc: int, mul: float, prev_arr: list,
00365                     goal_arr: list, cur_metr_name: str) -> NoReturn:
00366     """Inserts various information, like new best_so_far events, insertions into the open queue, etc.
```

```
00367
00368     Args:
00369         :param lite.Connection con: DB connection
00370         :param str operation: result, current, prom_0, skip
00371         :param str hname: hash name, same as MD filenames
00372         :param str src: from WQ (open queue)
00373         :param str dst: to VIZ (visited)
00374         :param list bsf: all best_so_far values for each metric
00375         :param int gc: greedy counter - affects events like seed change
00376         :param float mul: greedy multiplier - controls greediness
00377         :param list prev_arr: distance from the previous node
00378         :param list goal_arr: distance to the goal
00379         :param str cur_metr_name: name of the current metric
00380
00381     Returns:
00382     Stores data in the DB in a log table.
00383     """
00384     src = 'None' if src == '' else src
00385     dst = 'None' if dst == '' else dst
00386     nid = get_id_for_hash(con, hname)
00387     nid = 'None' if nid is None else nid
00388     columns = 'operation, id, src, dst, cur_metr, bsfr, bsfn, bsfh, bsfa, bsfx, gc, mul, '
00389
00390     if not isinstance(goal_arr, (list,)):  # short version for skip operation
00391         columns += 'dist_from_prev_1, dist_to_goal_1'
00392         final_str = ', '.join('"{}"'.format(elem) if isinstance(elem, str) else str(elem)
00393                               for elem in (operation, nid, src, dst, cur_metr_name, bsf[0], bsf[1], bsf[2], bsf[3],
00394                                            bsf[4], gc, mul, prev_arr, goal_arr))
00395     else:
00396         nseeds = len(prev_arr)  # long version for append operation
00397         columns += ', '.join(('dist_from_prev_{0}'.format(i+1) for i in range(nseeds))) + ', '
00398         columns += ', '.join(('dist_to_goal_{0}'.format(i+1) for i in range(nseeds)))
00399         prev_arr_str = ', '.join((str(elem) for elem in prev_arr))
00400         goal_arr_str = ', '.join((str(elem) for elem in goal_arr))
00401         final_str = ', '.join('"{}"'.format(elem) if isinstance(elem, str) else str(elem)
00402                               for elem in (operation, nid, src, dst, cur_metr_name, bsf[0], bsf[1], bsf[2], bsf[3], bsf[4], gc, mul))
00403         final_str += ", ".join(('', prev_arr_str, goal_arr_str))
00404
00405     qry = 'INSERT INTO log({}) VALUES ({})'.format(columns, final_str)
00406     cur = con.cursor()
00407     try:
00408         cur.execute(qry)
00409         con.commit()
00410     except Exception as e:
00411         print(e, '\nqry: ', operation, hname, src, dst, bsf, gc, mul, prev_arr, goal_arr)
00412         print('Extra info: ', qry)
00413         print('Type of function : {}'.format('Short' if not isinstance(goal_arr, (list,)) else 'Long'))
00414         log_error(con, 'LOG', nid)
00415
00416
00417 # def prep_insert_into_log(con, operation, name, src, dst, bsf, gc, mul, prev_arr, goal_arr):
00418 #     src = 'None' if src == '' else src
00419 #     nid = get_id_for_name(con, name)
00420 #     columns = 'operation, id, src, dst, bsf, gc, mul, '
00421 #
00422 #     if isinstance(goal_arr, (float, int)):  # short version
00423 #         columns += 'dist_from_prev_1, dist_to_goal_1'
00424 #         final_str = ', '.join('"{}"'.format(elem) if isinstance(elem, str) else str(elem)
00425 #                               for elem in (operation, nid, src, dst, bsf, gc, mul, prev_arr, goal_arr))
00426 #     else:
00427 #         nseeds = len(prev_arr)
00428 #         columns += ', '.join(('dist_from_prev_{0}, dist_to_goal_{0}'.format(i+1) for i in range(nseeds)))
00429 #         prev_arr_str = ', '.join((str(elem) for elem in prev_arr))
00430 #         goal_arr_str = ', '.join((str(elem) for elem in goal_arr))
00431 #         final_str = ', '.join('"{}"'.format(elem) if isinstance(elem, str) else str(elem)
00432 #                               for elem in (operation, nid, src, dst, bsf, gc, mul))
00433 #         final_str += ", ".join(('', prev_arr_str, goal_arr_str))
00434 #
00435 #     return final_str
00436
00437
00438 def copy_old_db(main_dict_keys: list, last_visited: list, next_in_oq: str, last_gc: float) -> NoReturn:
00439     """Used during the recovery procedure.
00440
00441     Args:
00442         :param list main_dict_keys: all hash values from the main_dict - storage of all metric information
00443         :param list last_visited: several (3) recent values from the visited queue
00444         :param str next_in_oq: next hash (id) in the open queue, used for double check
00445         :param float last_gc: last greedy counter observed in the information from the pickle
00446
00447     Returns:
```

```
00448     Conditionally copies data from the previous DB into a new one as a part of the restore process.
00449     """
00450     counter = 0
00451     db_path = os.getcwd()
00452     # db_name = 'results_{}.sqlite3'.format(counter)
00453     full_path = os.path.join(db_path, 'results_{}.sqlite3'.format(counter))
00454 
00455     while os.path.exists(full_path):
00456         prev_db = full_path
00457         counter += 1
00458         full_path = os.path.join(db_path, 'results_{}.sqlite3'.format(counter))
00459 
00460     #  yes, prev_db - the last one which exists
00461     cur_con = lite.connect(prev_db, check_same_thread=False, isolation_level=None)
00462 
00463     current_db_cur = cur_con.cursor()
00464 
00465     current_db_cur.execute("DELETE FROM log")
00466     current_db_cur.execute("DELETE FROM visited")
00467     current_db_cur.execute("DELETE FROM main_storage")
00468     cur_con.commit()
00469 
00470     prev_db_con = lite.connect(os.path.join(db_path, 'results_{}.sqlite3'.format(counter - 2)), check_same_thread=False, isolation_level=None)
00471 
00472     hashes = get_all_hashed_names(prev_db_con)
00473     for hash_hame in hashes:
00474         if hash_hame[0] in main_dict_keys:
00475             break
00476 
00477     max_id = get_id_for_hash(prev_db_con, hash_hame[0])
00478     prev_ids = [get_id_for_hash(prev_db_con, last_visited[0][2]), get_id_for_hash(prev_db_con, last_visited[1][2]),
      get_id_for_hash(prev_db_con, last_visited[2][2])]
00479     next_id = get_id_for_hash(prev_db_con, next_in_oq)
00480     # del last_visited, next_in_oq
00481     max_vid, vid_ts, last_vis_id = get_corr_vid_for_id(prev_db_con, max_id, prev_ids, last_gc)
00482     max_lid = get_corr_lid_for_id(prev_db_con, next_id, vid_ts, last_vis_id)
00483 
00484     prev_db_con.close()
00485     del prev_db_con, hash_hame, hashes, main_dict_keys
00486 
00487     current_db_cur.execute("ATTACH DATABASE ? AS prev_db", ('results_{}.sqlite3'.format(counter-2),))  # -1 - cur, -2 - prev
00488 
00489     current_db_cur.execute("INSERT INTO main.main_storage SELECT * FROM prev_db.main_storage WHERE prev_db.main_storage.id <= ?", (max_id,))
00490     cur_con.commit()
00491     current_db_cur.execute("INSERT INTO main.visited SELECT * FROM prev_db.visited WHERE prev_db.visited.vid <= ?", (max_vid,))
00492     cur_con.commit()
00493     current_db_cur.execute("INSERT INTO main.log SELECT * FROM prev_db.log WHERE prev_db.log.lid <= ?", (max_lid,))
00494     cur_con.commit()
00495 
00496 #
00497 # def sync_state_with_db(state):
00498 #     counter = 0
00499 #     db_path = os.getcwd()
00500 #     db_name = 'results_{}.sqlite3'.format(counter)
00501 #     full_path = os.path.join(db_path, 'results_{}.sqlite3'.format(counter))
00502 #
00503 #     while os.path.exists(full_path):
00504 #         prev_db = full_path
00505 #         counter += 1
00506 #         full_path = os.path.join(db_path, 'results_{}.sqlite3'.format(counter))
00507 #
00508 #     #  yes, prev_db - last one which exists
00509 #     cur_con = lite.connect(prev_db, check_same_thread=False, isolation_level=None)
00510 #
00511 #     current_db_cur = cur_con.cursor()
00512 #
00513 #     current_db_cur.execute("DELETE FROM log")
00514 #     # get_conn
00515 #     # get indexes
00516 #     # drop all log with
00517 #     # drop all vis with
00518 #     # drop all main with
00519 #     # vacuum
00520 #     return True
```

4.13 fix_filenames.py File Reference

Namespaces

- fix_filenames

Variables

- fix_filenames.files = os.walk('.').__next__()[2]
- int fix_filenames.counter = 0

4.14 fix_filenames.py

```
00001 #!/bin/env python3
00002 import os
00003 # import sys
00004
00005 files = os.walk('.').__next__()[2]
00006
00007 counter = 0
00008 # for file in files:
00009 #     if len(file) > 8:
00010 #         newname = file[int((len(file)-6)/2-1):]
00011 #         # if newname[:1] != '_':
00012 #         #     print('Found bug in renaming {} to {}'.format(file, newname))
00013 #         # print('File "{}" will be renamed to "{}"'.format(file, newname))
00014 #         os.rename(file, newname)
00015 #         counter += 1
00016     # else:
00017         # print('File "{}" will not be changed'.format(file))
00018     # if counter > 40:
00019     #     break
00020
00021 for file in files:
00022     os.rename(file, 's'+file)
00023     counter += 1
00024
00025 print('Files checked', counter)
```

4.15 gen_mdp.py File Reference

Namespaces

- gen_mdp

Functions

- str gen_mdp.get_mdp (int seed, int temp, str name='default')

  Generates text for .mdp file with simulation settings.

4.16 gen_mdp.py

```
00001 """
00002 This file contains only one function that generates configuration for the MD simulation.
00003     :platform: linux
00004
00005 .. moduleauthor:: Ivan Syzonenko <is2k@mtmail.mtsu.edu>
00006 """
00007 __license__ = "MIT"
00008 __docformat__ = 'reStructuredText'
00009
00010
00011 def get_mdp(seed: int, temp: int, name: str = 'default') -> str:
00012     """Generates text for .mdp file with simulation settings
00013
00014     Args:
00015         :param int seed: seed to be used for initial velocities generation
00016         :param int temp: temperature of the experiment
00017         :param str name: name of the experiment inside the .mdp file
00018
00019     Returns:
00020         :return: string with .mdp text
00021         :rtype: str
00022     """
00023     calibration_mdp = "\
00024 ; Run parameters\n\
00025 integrator  = md        ; leap-frog integrator\n\
00026 nsteps      = 10000     ; 2 * 10000 = 20 ps\n\
00027 dt          = 0.002     ; 2 fs\n\
00028 ld-seed     = {2:d}     ; \n\
00029 ; Output control\n\
00030 nstxout     = 0     ; save coordinates every 0.0 ps\n\
00031 nstvout     = 0     ; save velocities every 0.0 ps\n\
00032 nstenergy   = 10000 ; save energies every 0.0 ps\n\
00033 nstlog      = 0     ; update log file every 0.0 ps\n\
```

```
00034 nstxout-compressed  = 10000 ; save coordinates every 0.0 ps\n\
00035 energygrps  = Protein SOL\n\
00036 ; Bond parameters\n\
00037 continuation            = no        ; first dynamics run\n\
00038 constraint_algorithm    = lincs     ; holonomic constraints \n\
00039 constraints             = h-bonds   ; all bonds (even heavy atom-H bonds) constrained\n\
00040 lincs_iter              = 1         ; accuracy of LINCS\n\
00041 lincs_order             = 4         ; also related to accuracy\n\
00042 ; Neighborsearching\n\
00043 cutoff-scheme   = Verlet\n\
00044 ns_type         = grid      ; search neighboring grid cells\n\
00045 nstlist         = 10        ; 20 fs, largely irrelevant with Verlet\n\
00046 rcoulomb        = 1.0       ; short-range electrostatic cutoff (in nm)\n\
00047 rvdw            = 1.0       ; short-range van der Waals cutoff (in nm)\n\
00048 ; Electrostatics\n\
00049 coulombtype     = PME   ; Particle Mesh Ewald for long-range electrostatics\n\
00050 pme_order       = 4     ; cubic interpolation\n\
00051 fourierspacing  = 0.16  ; grid spacing for FFT\n\
00052 ; Temperature coupling is on\n\
00053 tcoupl      = V-rescale             ; modified Berendsen thermostat\n\
00054 tc-grps     = Protein Non-Protein   ; two coupling groups - more accurate\n\
00055 tau_t       = 0.1     0.1           ; time constant, in ps\n\
00056 ref_t       = {1:d}       {1:d}           ; reference temperature, one for each group, in K\n\
00057 ; Pressure coupling is off\n\
00058 pcoupl      = no        ; no pressure coupling in NVT\n\
00059 ; Periodic boundary conditions\n\
00060 pbc     = xyz           ; 3-D PBC\n\
00061 ; Dispersion correction\n\
00062 DispCorr    = EnerPres  ; account for cut-off vdW scheme\n\
00063 ; Velocity generation\n\
00064 gen-vel     = yes       ; assign velocities from Maxwell distribution\n\
00065 gen-temp    = {1:d}     ; temperature for Maxwell distribution\n\
00066 gen-seed    = {2:d}     ; generate a random seed".format(name, temp, seed)
00067     return calibration_mdp
```

4.17 generate_REMD_dirs.py File Reference

Namespaces

- generate_REMD_dirs

Functions

- def generate_REMD_dirs.gen_dirs ()
- def generate_REMD_dirs.get_mdp_str_ener_gr (str name, float temp, int seed, int steps)
- def generate_REMD_dirs.get_mdp_str_gpu (str name, float temp, int seed, int steps)

4.18 generate_REMD_dirs.py

```
00001 #!/usr/bin/env python3
00002
00003 import os
00004 from shutil import copy2 as cp2
00005
00006
00007 def gen_dirs():
00008     root_dir = 'REMD_profiles'
00009     cur_prot = 'TRP'
00010     tot_steps = 31250000  # trp 100 000
00011     # tot_steps = 166670000  # vil 100 000
00012     # tot_steps = 250000000  # gb1 800 000
00013
00014     full_path = os.path.join(root_dir, cur_prot)
00015     ffs = ['amber', 'charm', 'gromos', 'opls']
00016
00017     trp_profile_1 = [300.00, 302.87, 305.77, 308.69, 311.63, 314.59, 317.57, 320.58, 323.62, 326.67, 329.75, 332.86, 335.98, 339.13, 342.31,
00018                 345.51, 348.74, 351.99, 355.26, 358.56, 361.90, 365.25, 368.63, 372.04, 375.48, 378.93, 382.42, 385.94, 389.48, 393.05,
00019                 396.65, 400.00]  # amber, charm, opls
00020     trp_profile_2 = [300.00, 302.90, 305.83, 308.78, 311.76, 314.76, 317.78, 320.82, 323.89, 326.98, 330.10, 333.25, 336.41, 339.61, 342.82,
00021                 346.07, 349.34, 352.63, 355.95, 359.30, 362.67, 366.07, 369.50, 372.94, 376.42, 379.92, 383.46, 387.02, 390.62, 394.23,
00022                 397.89, 400.00]  # gromos
00023
00024     vil_profile_1 = [300.00, 303.07, 306.17, 309.30, 312.46, 315.64,
00025 318.85, 322.09, 325.35, 328.63, 331.95, 335.28, 338.65, 342.05, 345.48, 348.93,
00026 352.42, 355.93, 359.48, 363.05, 366.65, 370.29, 373.95, 377.64, 381.37, 385.13,
00027 388.91, 392.73, 396.59, 400.00
00028 ]  # amber, charm, opls
00029     vil_profile_2 = [300.00, 303.15, 306.32, 309.52, 312.75, 316.01, 319.29,
00030 322.58, 325.92, 329.29, 332.68, 336.11, 339.57, 343.05, 346.57, 350.11, 353.69,
00031 357.29, 360.93, 364.59, 368.29, 372.02, 375.79, 379.58, 383.41, 387.27, 391.17,
```

```
00032 395.10, 399.06, 400.00
00033 ]  # gromos
00034
00035     gb1_profile_1 = [300.00, 302.57, 305.16, 307.76, 310.39, 313.03,
00036                      315.69, 318.37, 321.07, 323.78, 326.52, 329.27, 332.05, 334.84, 337.62, 340.45,
00037                      343.30, 346.17, 349.07, 351.98, 354.91, 357.86, 360.84, 363.83, 366.84, 369.88,
00038                      372.94, 376.01, 379.11, 382.22, 385.37, 388.53, 391.72, 394.93, 398.16, 400.00
00039 ]  # amber, charm, opls
00040     gb1_profile_2 = [300.00, 302.57, 305.15, 307.76, 310.38, 313.03, 315.69,
00041                      318.37, 321.07, 323.78, 326.52, 329.27, 332.05, 334.84, 337.62, 340.45, 343.30,
00042                      346.17, 349.06, 351.98, 354.91, 357.86, 360.84, 363.83, 366.84, 369.88, 372.94,
00043                      376.01, 379.11, 382.23, 385.37, 388.54, 391.70, 394.91, 398.14, 400.00
00044 ]  # gromos
00045
00046     profile_1 = trp_profile_1
00047     profile_2 = trp_profile_2
00048
00049     temperartures = [
00050         profile_1,
00051         profile_1,
00052         profile_2,
00053         profile_1
00054     ]
00055
00056     try:
00057         os.mkdir(root_dir)
00058     except:
00059         print('Failed to create directory {}.'.format(root_dir))
00060
00061     try:
00062         os.mkdir(full_path)
00063     except:
00064         print('Failed to create directory {}.'.format(full_path))
00065
00066     gpu_flag = True
00067
00068     for i, ff in enumerate(ffs):
00069         work_dir = os.path.join(full_path, ff)
00070         try:
00071             os.mkdir(work_dir)
00072         except:
00073             print('Failed to create directory {}.'.format(os.path.join(full_path, ff)))
00074         for j, temp in enumerate(temperartures[i]):
00075             if gpu_flag:
00076                 mdp_content = get_mdp_str_gpu(name='REMD {}@{}'.format(cur_prot, ff), temp=temp, seed=1, steps=tot_steps)
00077             else:
00078                 mdp_content = get_mdp_str_ener_gr(name='REMD {}@{}'.format(cur_prot, ff), temp=temp, seed=1, steps=tot_steps)
00079             temp_dir = os.path.join(work_dir, '{}_{}_{}'.format(cur_prot, ff, j+1))
00080             try:
00081                 os.mkdir(temp_dir)
00082             except:
00083                 pass
00084             with open(os.path.join(temp_dir, 'md.mdp'), 'w') as mdp_file:
00085                 mdp_file.write(mdp_content)
00086
00087         # cp2(os.path.join(conf_files_dir, 'prot.ndx'), work_dir)
00088         # if ff == 'charm':
00089         #     cp2(os.path.join(conf_files_dir, 'charmm36-nov2018.ff'), work_dir)
00090
00091
00092 def get_mdp_str_ener_gr(name: str, temp: float, seed: int, steps: int):
00093     """
00094
00095         :param str name:
00096         :param float temp:
00097         :param int seed:
00098         :param int steps:
00099     """
00100     mdp_str = "\
00101         ; Run parameters\n\
00102         integrator  = md        ; leap-frog integrator\n\
00103         nsteps      = {3:d}     ; 2 * 10000 = 20 ps\n\
00104         dt          = 0.002     ; 2 fs\n\
00105         ld-seed     = {2:d}     ; \n\
00106         ; Output control\n\
00107         nstxout     = 0     ; save coordinates every 0.0 ps\n\
00108         nstvout     = 0     ; save velocities every 0.0 ps\n\
00109         nstenergy   = 10000 ; save energies every 0.0 ps\n\
00110         nstlog      = 10000     ; update log file every 0.0 ps\n\
00111         nstxout-compressed  = 10000 ; save coordinates every 0.0 ps\n\
00112         energygrps  = Protein SOL\n\
```

```
00113         ; Bond parameters\n\
00114         continuation            = no        ; first dynamics run\n\
00115         constraint_algorithm    = lincs     ; holonomic constraints \n\
00116         constraints             = h-bonds   ; all bonds (even heavy atom-H bonds) constrained\n\
00117         lincs_iter              = 1         ; accuracy of LINCS\n\
00118         lincs_order             = 4         ; also related to accuracy\n\
00119         ; Neighborsearching\n\
00120         cutoff-scheme   = Verlet\n\
00121         ns_type         = grid      ; search neighboring grid cells\n\
00122         nstlist         = 10        ; 20 fs, largely irrelevant with Verlet\n\
00123         rcoulomb        = 1.0       ; short-range electrostatic cutoff (in nm)\n\
00124         rvdw            = 1.0       ; short-range van der Waals cutoff (in nm)\n\
00125         ; Electrostatics\n\
00126         coulombtype     = PME   ; Particle Mesh Ewald for long-range electrostatics\n\
00127         pme_order       = 4     ; cubic interpolation\n\
00128         fourierspacing  = 0.16  ; grid spacing for FFT\n\
00129         ; Temperature coupling is on\n\
00130         tcoupl      = V-rescale             ; modified Berendsen thermostat\n\
00131         tc-grps     = Protein Non-Protein   ; two coupling groups - more accurate\n\
00132         tau_t       = 0.1     0.1           ; time constant, in ps\n\
00133         ref_t       = {1:f}       {1:f}           ; reference temperature, one for each group, in K\n\
00134         ; Pressure coupling is off\n\
00135         pcoupl      = no        ; no pressure coupling in NVT\n\
00136         ; Periodic boundary conditions\n\
00137         pbc     = xyz           ; 3-D PBC\n\
00138         ; Dispersion correction\n\
00139         DispCorr    = EnerPres  ; account for cut-off vdW scheme\n\
00140         ; Velocity generation\n\
00141         gen-vel     = yes       ; assign velocities from Maxwell distribution\n\
00142         gen-temp    = {1:f}     ; temperature for Maxwell distribution\n\
00143         gen-seed    = {2:d}     ; generate a random seed".format(name, temp, seed, steps)
00144     return mdp_str
00145 
00146 
00147 def get_mdp_str_gpu(name: str, temp: float, seed: int, steps: int):
00148     """
00149 
00150         :param str name:
00151         :param float temp:
00152         :param int seed:
00153         :param int steps:
00154     """
00155     mdp_str = "\
00156         ; Run parameters\n\
00157         integrator  = md        ; leap-frog integrator\n\
00158         nsteps      = {3:d}     ; 2 * 10000 = 20 ps\n\
00159         dt          = 0.002     ; 2 fs\n\
00160         ld-seed     = {2:d}     ; \n\
00161         ; Output control\n\
00162         nstxout     = 0     ; save coordinates every 0.0 ps\n\
00163         nstvout     = 0     ; save velocities every 0.0 ps\n\
00164         nstenergy   = 0 ; save energies every 0.0 ps\n\
00165         nstlog      = 10000     ; update log file every 0.0 ps\n\
00166         nstxout-compressed  = 10000 ; save coordinates every 0.0 ps\n\
00167         ; Bond parameters\n\
00168         continuation            = no        ; first dynamics run\n\
00169         constraint_algorithm    = lincs     ; holonomic constraints \n\
00170         constraints             = h-bonds   ; all bonds (even heavy atom-H bonds) constrained\n\
00171         lincs_iter              = 1         ; accuracy of LINCS\n\
00172         lincs_order             = 4         ; also related to accuracy\n\
00173         ; Neighborsearching\n\
00174         cutoff-scheme   = Verlet\n\
00175         ns_type         = grid      ; search neighboring grid cells\n\
00176         nstlist         = 10        ; 20 fs, largely irrelevant with Verlet\n\
00177         rcoulomb        = 1.0       ; short-range electrostatic cutoff (in nm)\n\
00178         rvdw            = 1.0       ; short-range van der Waals cutoff (in nm)\n\
00179         ; Electrostatics\n\
00180         coulombtype     = PME   ; Particle Mesh Ewald for long-range electrostatics\n\
00181         pme_order       = 4     ; cubic interpolation\n\
00182         fourierspacing  = 0.16  ; grid spacing for FFT\n\
00183         ; Temperature coupling is on\n\
00184         tcoupl      = V-rescale             ; modified Berendsen thermostat\n\
00185         tc-grps     = Protein Non-Protein   ; two coupling groups - more accurate\n\
00186         tau_t       = 0.1     0.1           ; time constant, in ps\n\
00187         ref_t       = {1:f}       {1:f}           ; reference temperature, one for each group, in K\n\
00188         ; Pressure coupling is off\n\
00189         pcoupl      = no        ; no pressure coupling in NVT\n\
00190         ; Periodic boundary conditions\n\
00191         pbc     = xyz           ; 3-D PBC\n\
00192         ; Dispersion correction\n\
00193         DispCorr    = EnerPres  ; account for cut-off vdW scheme\n\
```

```
00194         ; Velocity generation\n\
00195         gen-vel     = yes       ; assign velocities from Maxwell distribution\n\
00196         gen-temp    = {1:f}     ; temperature for Maxwell distribution\n\
00197         gen-seed    = {2:d}     ; generate a random seed".format(name, temp, seed, steps)
00198     return mdp_str
00199
00200
00201 if __name__ == '__main__':
00202     gen_dirs()
```

4.19 generate_total_best_tables.py File Reference

Namespaces

- generate_total_best_tables

Functions

- def generate_total_best_tables.main ()
- def generate_total_best_tables.plot_tables (list filenames_db, str out_file, list table_names)

4.20 generate_total_best_tables.py

```
00001 #!/usr/bin/env python3
00002
00003 import os
00004 import sqlite3 as lite
00005 import matplotlib.pyplot as plt
00006 import numpy as np
00007 from matplotlib.figure import figaspect
00008 import multiprocessing as mp
00009
00010 # #########   TRP   ######################
00011 # for ff in ffs:
00012 #     filenames_db = ['results_{}_trp_300_fixed.sqlite3'.format(ff), 'results_{}_trp_300_2_fixed.sqlite3'.format(ff)]
00013 #     legend_names = ['TRP {}_1'.format(ff), 'TRP {}_2'.format(ff)]
00014 #     common_path = '../trp_{}_compar'.format(ff)
00015 #     batch_arr.append((filenames_db, legend_names, common_path))
00016 #
00017 # filenames_db = ['results_amber_trp_300_2_fixed.sqlite3', 'results_charm_trp_300_2_fixed.sqlite3', 'results_gromos_trp_300_2_fixed.sqlite3',
       'results_opls_trp_300_2_fixed.sqlite3']
00018 # legend_names = ['TRP amber_2', 'TRP charm_2', 'TRP gromos_2', 'TRP opls_2']
00019 # common_path = '../trp_all_2_compar'
00020 # batch_arr.append((filenames_db, legend_names, common_path))
00021 #
00022 # filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_charm_trp_300_fixed.sqlite3', 'results_gromos_trp_300_fixed.sqlite3',
       'results_opls_trp_300_fixed.sqlite3']
00023 # legend_names = ['TRP amber', 'TRP charm', 'TRP gromos', 'TRP opls']
00024 # common_path = '../trp_all_1_compar'
00025 # batch_arr.append((filenames_db, legend_names, common_path))
00026 #
00027 # filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_amber_trp_300_2_fixed.sqlite3', 'results_charm_trp_300_fixed.sqlite3',
       'results_charm_trp_300_2_fixed.sqlite3', 'results_gromos_trp_300_fixed.sqlite3', 'results_gromos_trp_300_2_fixed.sqlite3',
       'results_opls_trp_300_fixed.sqlite3', 'results_opls_trp_300_2_fixed.sqlite3']
00028 # legend_names = ['TRP amber', 'TRP amber_2', 'TRP charm', 'TRP charm_2', 'TRP gromos', 'TRP gromos_2', 'TRP opls', 'TRP opls_2']
00029 # common_path = '../trp_all_compar'
00030 # batch_arr.append((filenames_db, legend_names, common_path))
00031
00032 # #################  VIL  ######################
00033 #
00034 # filenames_db = ['results_amber_vil_300.sqlite3', 'results_charm_vil_300.sqlite3', 'results_gromos_vil_300.sqlite3',
       'results_opls_vil_300.sqlite3']
00035 # legend_names = ['VIL amber', 'VIL charm', 'VIL gromos', 'VIL opls']
00036 # common_path = '../vil_all_compar'
00037 # batch_arr.append((filenames_db, legend_names, common_path))
00038
00039
00040 # #################  GB1  ######################
00041 #
00042 # filenames_db = ['results_amber_gb1_300.sqlite3', 'results_charm_gb1_300.sqlite3', 'results_gromos_gb1_300.sqlite3',
       'results_opls_gb1_300.sqlite3']
00043 # legend_names = ['GB1s amber', 'GB1 charm', 'GB1 gromos', 'GB1 opls']
00044 # common_path = '../gb1_all_compar'
00045 # batch_arr.append((filenames_db, legend_names, common_path))
00046
00047
00048
00049 def main():
00050
00051     # ############ TRP #########
```

```
00052     # filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_amber_trp_300_2_fixed.sqlite3', 'results_charm_trp_300_fixed.sqlite3',
      'results_charm_trp_300_2_fixed.sqlite3', 'results_gromos_trp_300_fixed.sqlite3', 'results_gromos_trp_300_2_fixed.sqlite3',
      'results_opls_trp_300_fixed.sqlite3', 'results_opls_trp_300_2_fixed.sqlite3']
00053     # table_names = ['amber trp 1', 'amber trp 2', 'charm trp 1', 'charm trp 2', 'gromos trp 1', 'gromos trp 2', 'opls trp 1', 'opls trp 2']
00054     # outfile = 'all_trp_all'
00055     # plot_tables(filenames_db, outfile, table_names)
00056
00057     # filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_charm_trp_300_fixed.sqlite3', 'results_gromos_trp_300_fixed.sqlite3',
      'results_opls_trp_300_fixed.sqlite3']
00058     # table_names = ['amber trp 1', 'charm trp 1', 'gromos trp 1', 'opls trp 1']
00059     # outfile = 'all_trp_1'
00060     # plot_tables(filenames_db, outfile, table_names)
00061
00062     # filenames_db = ['results_amber_trp_300_2_fixed.sqlite3', 'results_charm_trp_300_2_fixed.sqlite3',
      'results_gromos_trp_300_2_fixed.sqlite3', 'results_opls_trp_300_2_fixed.sqlite3']
00063     # table_names = ['amber trp 2', 'charm trp 2', 'gromos trp 2', 'opls trp 2']
00064     # outfile = 'all_trp_2'
00065     # plot_tables(filenames_db, outfile, table_names)
00066
00067     # filenames_db = ['results_amber_trp_300_fixed.sqlite3', 'results_amber_trp_300_2_fixed.sqlite3']
00068     # table_names = ['amber trp 1', 'amber trp 2']
00069     # outfile = 'amber_trp'
00070     # plot_tables(filenames_db, outfile, table_names)
00071
00072     # filenames_db = ['results_charm_trp_300_fixed.sqlite3', 'results_charm_trp_300_2_fixed.sqlite3']
00073     # table_names = ['charm trp 1', 'charm trp 2']
00074     # outfile = 'charm_trp'
00075     # plot_tables(filenames_db, outfile, table_names)
00076
00077     # filenames_db = ['results_gromos_trp_300_fixed.sqlite3', 'results_gromos_trp_300_2_fixed.sqlite3']
00078     # table_names = ['gromos trp 1', 'gromos trp 2']
00079     # outfile = 'gromos_trp'
00080     # plot_tables(filenames_db, outfile, table_names)
00081
00082     # filenames_db = ['results_opls_trp_300_fixed.sqlite3', 'results_opls_trp_300_2_fixed.sqlite3']
00083     # table_names = ['opls trp 1', 'opls trp 2']
00084     # outfile = 'opls_trp'
00085     # plot_tables(filenames_db, outfile, table_names)
00086
00087
00088     # # ############ VIL #########
00089     # filenames_db = ['results_amber_vil_300.sqlite3', 'results_charm_vil_300.sqlite3', 'results_gromos_vil_300.sqlite3',
      'results_opls_vil_300.sqlite3']
00090     # table_names = ['amber vil', 'charm vil', 'gromos vil', 'opls vil']
00091     # outfile = 'all_vil'
00092     # plot_tables(filenames_db, outfile, table_names)
00093
00094
00095     # ############ GB1 #########
00096     filenames_db = ['results_amber_gb1_300.sqlite3', 'results_charm_gb1_300.sqlite3', 'results_gromos_gb1_300.sqlite3',
      'results_opls_gb1_300.sqlite3']
00097     table_names = ['amber gb1', 'charm gb1', 'gromos gb1', 'opls gb1']
00098     outfile = 'all_gb1'
00099     plot_tables(filenames_db, outfile, table_names)
00100
00101
00102 def plot_tables(filenames_db: list, out_file: str, table_names: list):
00103     """
00104
00105     Args:
00106         :param list filenames_db:
00107         :param str out_file:
00108         :param list table_names:
00109     """
00110     out_file = '{}.tex'.format(out_file)
00111     con_arr = [lite.connect(db_name, check_same_thread=False, isolation_level=None) for db_name in filenames_db]
00112     cur_arr = [con.cursor() for con in con_arr]
00113     metrics = ["RMSD", "ANGL", "AND_H", "AND", "XOR"]
00114     metrics_tab = ["RMSD", "ANGL", "AND\\_H", "AND", "XOR"]
00115     allowed_faild = [20, 10, 5, 5, 10]
00116
00117     total_promotions = list()
00118     prom_during_metric = list()
00119     total_steps_during_metric = list()
00120     for db_name in filenames_db:
00121         con = lite.connect(db_name, check_same_thread=False, isolation_level=None)
00122         cur = con.cursor()
00123         qry = "select count(1) from log where operation='prom_0' "  # total
00124         result = cur.execute(qry)
00125         total_promotions.append(result.fetchone()[0])
00126         personal_res = list()
```

```
00127         personal_total_steps = list()
00128         for partial_metr in metrics:
00129             qry = "select count(1) from log where operation='prom_O' and cur_metr='{}'".format(partial_metr)
00130             result = cur.execute(qry)
00131             personal_res.append(result.fetchone()[0])
00132         prom_during_metric.append(personal_res)
00133         for partial_metr in metrics:
00134             qry = "select count(1) from log where dst='VIZ' and cur_metr='{}'".format(partial_metr)
00135             result = cur.execute(qry)
00136             personal_total_steps.append(result.fetchone()[0])
00137         total_steps_during_metric.append(personal_total_steps)
00138         del personal_res
00139         con.close()
00140     del result, qry, partial_metr, db_name, cur, con, con_arr, cur_arr, personal_total_steps
00141     a = 8
00142     # for i in range(len(total_promotions)):
00143     #     print('{}\t{}\t{}\t{}\t{}\t{}\t{}\t{}'.format('metr', 'per_s', 'tot_s', 'pe/to', 'pe/al   ', 'to/al', 'to/al ', 'fa'))
00144     #     for j in range(len(prom_during_metric[i])):
00145     #         print('{}:\t{:d}  \t{:d}\t{:3.2f}\t{:3.2f} \t{:3.2f}\t{:3.2f}\t{:d}'.format(metrics[j], prom_during_metric[i][j],
    total_steps_during_metric[i][j], 100*prom_during_metric[i][j]/total_steps_during_metric[i][j], 100 * total_steps_during_metric[i][j] /
    total_promotions[i], 100 * prom_during_metric[i][j] / total_promotions[i], 100 * prom_during_metric[i][j] / sum(total_promotions),
    allowed_faild[j]))
00146     #     print('t: {}\t{}'.format(total_promotions[i], sum(total_steps_during_metric[i])))
00147     #     print()
00148
00149
00150     with open(out_file, 'w') as tex_table:
00151         tex_table.writelines(['\\begin{table}[h]\n', '\centering\n', '\sisetup{table-align-text-post=false}\n',
    '\\begin{tabular}{@{}|l|S[table-format=2.0]\
00152
    |S[table-format=3.0]\
00153
    |S[table-format=6]\
00154
    |S[table-format=3.3]\
00155
    |S[table-format=3.2]\
00156
    |S[table-format=3.3]\
00157
    |S[table-format=1.2]\
00158
    |@{}}\n']
00159         for i in range(len(total_promotions)):
00160             tex_table.write('')
00161
00162             tex_table.write('\multicolumn{{8}}{{c}}{{{}}} \\\\ \n'.format(table_names[i]))
00163             tex_table.write('\\hline\n')
00164             tex_table.write('{} & {} & {} & {} & {} & {} & {} & {} & {} \\\\ \n'.format(          '',      '{allowed}', '{percent}', '{metric}',
    '{percent}', '{promotions}', '{percent of}', '{promotions}'))
00165             tex_table.write('{} & {} & {} & {} & {} & {} & {} & {} & {} \\\\ \\hline\n'.format('{metric}', '{fails}', '{allowed}', '{total
    steps}', '{steps}', '{per metric}', '{promotions}', '{per 1000 steps}'))
00166             for j in range(len(prom_during_metric[i])):
00167                 tex_table.write('{:s} & {:d} & {:3.0f}\\si{{\\percent}} & {:d} & {:3.2f}\\si{{\\percent}} & {} & {:3.2f}\\si{{\\percent}}  &
    {:3.2f} \\\\ \\hline\n'.format(
00168                     metrics_tab[j],
00169                     allowed_faild[j],
00170                     100*allowed_faild[j]/sum(allowed_faild),
00171                     total_steps_during_metric[i][j],
00172                     100*total_steps_during_metric[i][j]/sum(total_steps_during_metric[i]),
00173                     prom_during_metric[i][j],
00174                     100 * prom_during_metric[i][j]/sum(prom_during_metric[i]),
00175                     1000 * prom_during_metric[i][j]/total_steps_during_metric[i][j]))
00176             tex_table.write('{:s} & {:d} & {:3.0f}\\si{{\\percent}} & {:d} & {:3.2f}\\si{{\\percent}} & {} & {:3.2f}\\si{{\\percent}} &
    {:3.2f}\\\\ \\hline \\hline \n'.format('total', sum(allowed_faild), 100, sum(total_steps_during_metric[i]), 100, sum(prom_during_metric[i]),
    100, 1000 * sum(prom_during_metric[i])/sum(total_steps_during_metric[i])))
00177         tex_table.writelines(['\\end {tabular}\n', '\\caption{{{}}}\n'.format('{}'.format(', '.join(table_names))), '\\end {table}\n'])
00178
00179         tex_table.write('\n\n\n')
00180
00181
00182         total_steps_during_metric_comb = [sum(x) for x in zip(*total_steps_during_metric)]
00183         prom_during_metric_comb = [sum(x) for x in zip(*prom_during_metric)]
00184
00185         tex_table.writelines(['\\begin{table}[h]\n', '\centering\n', '\sisetup{table-align-text-post=false}\n',
    '\\begin{tabular}{@{}|l|S[table-format=2.0]\
00186
    |S[table-format=3.0]\
00187
    |S[table-format=6]\
```

```
00188
    |S[table-format=3.3]\
00189
    |S[table-format=3.2]\
00190
    |S[table-format=3.3]\
00191
    |S[table-format=1.2]\
00192
    |@{}}\n', '\\hline\n'])
00193       tex_table.write('{} & {} & {} & {} & {} & {} & {} & {} \\\\ \n'.format('', '{allowed}', '{percent}', '{metric}', '{percent}',
    '{promotions}', '{percent of}', '{promotions}'))
00194       tex_table.write('{} & {} & {} & {} & {} & {} & {} & {} \\\\ \\hline\n'.format('{metric}', '{fails}', '{allowed}', '{total steps}',
    '{steps}', '{per metric}', '{promotions}', '{per 1000 steps}'))
00195       for j in range(len(prom_during_metric_comb)):
00196           tex_table.write('{:s} & {:d} & {:3.0f}\\si{{\\percent}} & {:d} & {:3.2f}\\si{{\\percent}} & {} & {:3.2f}\\si{{\\percent}}  &
    {:3.2f} \\\\ \\hline\n'.format(
00197               metrics_tab[j],
00198               allowed_faild[j],
00199               100*allowed_faild[j]/sum(allowed_faild),
00200               total_steps_during_metric_comb[j],
00201               100*total_steps_during_metric_comb[j]/sum(total_steps_during_metric_comb),
00202               prom_during_metric_comb[j],
00203               100 * prom_during_metric_comb[j]/sum(prom_during_metric_comb),
00204               1000 * prom_during_metric_comb[j]/total_steps_during_metric_comb[j]))
00205       tex_table.write('{:s} & {:d} & {:3.0f}\\si{{\\percent}} & {:d} & {:3.2f}\\si{{\\percent}} & {} & {:3.2f}\\si{{\\percent}} &
    {:3.2f}\\\\ \\hline \\hline \n'.format('total', sum(allowed_faild), 100, sum(total_steps_during_metric_comb), 100,
    sum(prom_during_metric_comb), 100, 1000 * sum(prom_during_metric_comb)/sum(total_steps_during_metric_comb)))
00206       tex_table.writelines(['\\end {tabular}\n', '\\caption{{{}}}\n'.format('Summary of ({})'.format(', '.join(table_names))), '\\end
    {table}\n'])
00207
00208
00209
00210       tex_table.write('\n\n\n')
00211       norm_coef = [min(allowed_faild)/elem   for elem in allowed_faild]
00212       allowed_faild = [elem * norm_coef[k] for k, elem in enumerate(allowed_faild)]
00213
00214       tex_table.writelines(['\\begin{table}[h]\n', '\centering\n', '\sisetup{table-align-text-post=false}\n',
    '\\begin{tabular}{@{}|l|S[table-format=2.0]\
00215
    |S[table-format=3.0]\
00216
    |S[table-format=6]\
00217
    |S[table-format=3.3]\
00218
    |S[table-format=3.2]\
00219
    |S[table-format=3.3]\
00220
    |S[table-format=1.2]\
00221
    |@{}}\n'])
00222
00223       for i in range(len(total_promotions)):
00224           total_steps_during_metric[i] = [elem * norm_coef[k] for k, elem in enumerate(total_steps_during_metric[i])]
00225           prom_during_metric[i] = [elem * norm_coef[k] for k, elem in enumerate(prom_during_metric[i])]
00226
00227           tex_table.write('\multicolumn{{8}}{{c}}{{{}}} \\\\ \n'.format(table_names[i]))
00228           tex_table.write('\\hline\n')
00229           tex_table.write('{} & {} & {} & {} & {} & {} & {} & {} \\\\ \n'.format('', '{allowed}', '{percent}', '{metric}', '{percent}',
    '{promotions}', '{percent of}', '{promotions}'))
00230           tex_table.write(
00231               '{} & {} & {} & {} & {} & {} & {} & {} \\\\ \\hline\n'.format('{metric}', '{fails}', '{allowed}', '{total steps}', '{steps}',
    '{per metric}', '{promotions}', '{per 1000 steps}'))
00232           for j in range(len(prom_during_metric[i])):
00233               tex_table.write('{:s} & {:3.0f} & {:2.0f}\\si{{\\percent}} & {:3.0f} & {:3.2f}\\si{{\\percent}} & {} &
    {:3.2f}\\si{{\\percent}} & {:3.2f}  \\\\ \\hline\n'.format(
00234                   metrics_tab[j],
00235                   allowed_faild[j],
00236                   100*allowed_faild[j]/sum(allowed_faild),
00237                   total_steps_during_metric[i][j],
00238                   100*total_steps_during_metric[i][j]/sum(total_steps_during_metric[i]),
00239                   prom_during_metric[i][j],
00240                   100 * prom_during_metric[i][j]/sum(prom_during_metric[i]),
00241                   1000 * prom_during_metric[i][j]/total_steps_during_metric[i][j]))
00242           tex_table.write('{:s} & {:3.0f} & {:2.0f}\\si{{\\percent}} & {:2.0f} & {}\\si{{\\percent}} & {} & {}\\si{{\\percent}} &
    {:3.2f}\\\\ \\hline \\hline \n'.format('total', sum(allowed_faild), 100, sum(total_steps_during_metric[i]), 100, sum(prom_during_metric[i]),
    100, 1000 * sum(prom_during_metric[i])/sum(total_steps_during_metric[i])))
00243       tex_table.writelines(['\\end {tabular}\n', '\\caption{{{}}}\n'.format('Normalized ' + '{}'.format(', '.join(table_names))), '\\end
    {table}\n'])
```

```
    total_steps_during_metric_comb = [sum(x) for x in zip(*total_steps_during_metric)]
    prom_during_metric_comb = [sum(x) for x in zip(*prom_during_metric)]

    tex_table.write('\n\n\n')

    tex_table.writelines(['\\begin{table}[h]\n', '\centering\n', '\sisetup{table-align-text-post=false}\n', '\\begin{tabular}{@{}|l|S[table-format=2.0]\
|S[table-format=3.0]\
|S[table-format=6]\
|S[table-format=3.3]\
|S[table-format=3.2]\
|S[table-format=3.3]\
|S[table-format=1.2]\
                                                                          |@{}}\n', '\\hline\n'])
    tex_table.write('{} & {} & {} & {} & {} & {} & {} & {} \\\\ \n'.format('', '{allowed}', '{percent}', '{metric}', '{percent}', '{promotions}', '{percent of}', '{promotions}'))
    tex_table.write('{} & {} & {} & {} & {} & {} & {} & {} \\\\ \\hline\n'.format('{metric}', '{fails}', '{allowed}', '{total steps}', '{steps}', '{per metric}', '{promotions}', '{per 1000 steps}'))
    for j in range(len(prom_during_metric_comb)):
        tex_table.write('{:s} & {:3.0f} & {:2.0f}\\si{{\\percent}} & {:3.0f} & {:3.2f}\\si{{\\percent}} & {} & {:3.2f}\\si{{\\percent}} & {:3.2f}  \\\\ \\hline\n'.format(
            metrics_tab[j],
            allowed_faild[j],
            100*allowed_faild[j]/sum(allowed_faild),
            total_steps_during_metric_comb[j],
            100*total_steps_during_metric_comb[j]/sum(total_steps_during_metric_comb),
            prom_during_metric_comb[j],
            100 * prom_during_metric_comb[j]/sum(prom_during_metric_comb),
            1000 * prom_during_metric_comb[j]/total_steps_during_metric_comb[j]))
    tex_table.write('{:s} & {:3.0f} & {:2.0f}\\si{{\\percent}} & {:2.0f} & {}\\si{{\\percent}} & {} & {}\\si{{\\percent}} & {:3.2f}\\\\ \\hline \\hline \n'.format('total', sum(allowed_faild), 100, sum(total_steps_during_metric_comb), 100, sum(prom_during_metric_comb), 100, 1000 * sum(prom_during_metric_comb)/sum(total_steps_during_metric_comb)))
    tex_table.writelines(['\\end {tabular}\n', '\\caption{{{}}}\n'.format('Normalized ' + 'summary of ({})'.format(', '.join(table_names))), '\\end {table}\n'])




    #     tex_table.writelin('\\hline')
    #     qry = "select count(1) from log where operation='prom_O' "
    #     result_arr = cur.execute(qry) cur_arr
    #     total_prom = [res.fetchone() for res in result_arr]
    #     for partial_metr in ["RMSD", "ANGL", "AND_H", "AND", "XOR"]:
    #         qry = "select count(1) from log where operation='prom_O' and cur_metr='{}'".format(partial_metr)
    #         result_arr = [cur.execute(qry) for cur in cur_arr]
    #         fetched_one_arr = [res.fetchone() for res in result_arr]
    #         tex_table.writelin('\\hline')
    #     tex_table.writelin('\\hline')
    #
    # tex_table.writelines(['\\caption\{{}\}'.format('some caption here'), '\\end {tabular}', '\\end {table}'])


if __name__ == '__main__':
    main()
```

4.21 GMDA_main.py File Reference

Namespaces

- GMDA_main

Functions

- list GMDA_main.queue_rebuild (list process_queue, list open_queue_to_rebuild, dict node_info, float cur_mult, str new_metr_name, bool sep_proc=True)

  Resorts the queue according to the new metric.
- int GMDA_main.get_atom_num (str ndx_file)

  Computes number of atoms in the particular index file.
- tuple GMDA_main.parse_hostnames (int seednum, str hostfile='hostfile')

  Spreads the load among the hosts found in the hostfile.

- tuple GMDA_main.compute_on_local_machine (list cpu_map, list seed_list, str cur_name, str past_dir, str work_dir, dict seed_dirs, str topol_file_init, str ndx_file_init, list prev_runs_files, str old_name_digest)

  This version is optimised for usage on one machine with tMPI (see GROMACS docs).
- tuple GMDA_main.compute_with_mpi (list seed_list, str cur_name, str past_dir, str work_dir, dict seed_dirs, str topol_file_init, str ndx_file_init, list prev_runs_files, str old_name_digest, int tot_seeds, list hostnames, list ncores, bool sched=False, int ntomp=1)

  This version is optimised for usage on more than one machine with tMPI and/or MPI.
- bool GMDA_main.check_in_queue (list queue, str elem_hash)

  Checks whether elements with provided hash exists in the queue.
- list GMDA_main.second_chance (list open_queue, list visited_queue, str best_so_far_name, str cur_metric, dict main_dict, int node_max_↩att, str cur_metric_name, str best_so_far, float tol_error, float greed_mult)

  Typically executed during the seed change.
- list GMDA_main.check_dupl (str name_to_check, list visited_queue)

  This function is just a detector of duplicates.
- NoReturn GMDA_main.GMDA_main (list prev_runs_files, str past_dir, mp.JoinableQueue print_queue, mp.JoinableQueue db_input_queue, mp.↩JoinableQueue copy_queue, mp.JoinableQueue rm_queue, int tot_seeds=4)

  This is the main loop.

Variables

- int GMDA_main.MAX_ITEMS_TO_HANDLE = 50000

4.22 GMDA_main.py

```
00001 #!/usr/bin/env python3
00002
00003 """
00004 This file contains main computational loop and functions highly related to it
00005 .. module:: GMDA_main
00006     :platform: linux
00007
00008 .. moduleauthor:: Ivan Syzonenko <is2k@mtmail.mtsu.edu>
00009 """
00010 __license__ = "MIT"
00011 __docformat__ = 'reStructuredText'
00012
00013 import heapq
00014 import time
00015 import os
00016 import multiprocessing as mp
00017 import numpy as np
00018 from shutil import copy2 as cp2
00019 from pathlib import Path
00020 import zlib
00021 import gc
00022
00023 from typing import NoReturn
00024
00025 from db_proc import insert_into_log, insert_into_main_stor, insert_into_visited, copy_old_db
00026 from helper_funcs import trjcat_many, make_a_step, create_core_mapping, get_seed_dirs, check_precomputed_noize, \
00027                          get_new_seeds, get_digest, make_a_step2, rm_seed_dirs, make_a_step3, main_state_recover, supp_state_recover, \
00028                          main_state_backup, supp_state_backup
00029 from parse_topology_for_hydrogens import parse_top_for_h
00030 from gmx_wrappers import gmx_trjcat, gmx_trjconv
00031 from metric_funcs import get_knn_dist_mdsctk, get_bb_to_angle_mdsctk, get_angle_to_sincos_mdsctk, \
00032                          get_native_contacts, gen_file_for_amb_noize, save_an_file, compute_init_metric, compute_metric, \
00033                          select_metrics_by_snr, get_contat_profile_mdsctk
00034 # from pympler import muppy, summary
00035 # from memory_profiler import profile
00036 # import sys
00037 MAX_ITEMS_TO_HANDLE = 50000
00038 # extra_past = './'  # define extra past dir - this is temporary handle.
00039
00040 # def proc_local_minim(open_queue, best_so_far_name: str, tol_error, ndx_file: str, name_2_digest_map: dict, goal_top: str, local_minim_names:
      list):
00041 #     """
00042 #     Deprecated approach to block falling into local minima basin
00043 #     :param open_queue: sorted queue that contains nodes about to be processed. This is actually only a partial queue (only top elements)
00044 #     :param best_so_far_name: name of the trajectory closest to the goal (according to the current metric)
00045 #     :param tol_error: minimal metric vibration of the NMR structure
00046 #     :param ndx_file: .ndx - index of the protein atoms of the current conformation
00047 #     :param name_2_digest_map: dictionary that maps trajectory name to it's precomputed digest
00048 #     :param goal_top: .top - topology of the NMR conformation
00049 #     :param local_minim_names: list of nodes close to the local minima
00050 #     :return:
00051 #     """
```

```
00052 #     # split name into subnames
00053 #     # compute distance
00054 #     # import math
00055 #     range_lim = 6
00056 #     strict = True
00057 #     if strict:
00058 #         basin_err = tol_error * 4
00059 #         stem_err = lambda i: tol_error - 2 * tol_error * i / 5
00060 #     else:
00061 #         basin_err = tol_error * 2
00062 #         stem_err = lambda i: tol_error - tol_error * i / 5
00063 #
00064 #     prev_points = best_so_far_name.split('_')
00065 #     past_dir = './past'
00066 #     # len_prev_points = len(prev_points)
00067 #     # step = len_prev_points//18
00068 #     all_prev_names = ['_'.join(prev_points[:i]) for i in range(1, len(prev_points))]
00069 #     hashed_names = [os.path.join(past_dir, name_2_digest_map[point] + '.xtc') for point in all_prev_names]
00070 #     len_hashed_names = len(hashed_names)
00071 #     closest_to_minim = hashed_names[len_hashed_names:len_hashed_names // 2:-1]
00072 #     gmx_trjcat(f=closest_to_minim, o='local_min.xtc', n=ndx_file, cat=True, vel=False, sort=False, overwrite=True)
00073 #     # range_lim = min(6, len_prev_points)
00074 #
00075 #     hashed_names = [name_2_digest_map[name[4]] for name in open_queue]
00076 #
00077 #     trjcat_many(hashed_names, past_dir, './combinded_traj_openq.xtc')
00078 #
00079 #     rmsd = get_knn_dist_mdsctk('./combinded_traj_openq.xtc', 'local_min.xtc', goal_top)
00080 #
00081 #     rmsd_structured = list()
00082 #     for i in range(len(closest_to_minim)):
00083 #         rmsd_structured.append(rmsd[i * len(hashed_names):(i + 1) * len(hashed_names)])
00084 #
00085 #     # next part of code implements gradual pruning:
00086 #     # the closer point to the end of perfect path - the closer we are to the local minim center
00087 #     # so we need to remove all near points.
00088 #     # some extra code here to handle case when we have shorter paths and make sure that
00089 #     # the most pruning will receive only center
00090 #     step = len(rmsd_structured)//range_lim if len(rmsd_structured) > range_lim else 1
00091 #     how_many = [0]
00092 #     sum = 0
00093 #     for i in range(1, range_lim):
00094 #         sum += step
00095 #         how_many.append(sum)
00096 #         if sum == len(rmsd_structured):
00097 #             break
00098 #     how_many[-1] += len(rmsd_structured) - step * (len(how_many) - 1)
00099 #     set_of_points_to_remove = set()
00100 #
00101 #     for i in range(len(how_many)-1):
00102 #         subarr = rmsd_structured[how_many[i]:how_many[i+1]]
00103 #         for line_of_points in subarr:
00104 #             for point_pos, point in enumerate(line_of_points):
00105 #                 if point < stem_err(i):
00106 #                     set_of_points_to_remove.add(point_pos)
00107 #
00108 #     print('Main stem, trimming {} points'.format(len(set_of_points_to_remove)))
00109 #
00110 #     # at this point we cleaned main stem of perfect path
00111 #     # now its time to clean local minimum basin
00112 #
00113 #     hashed_names = [name_2_digest_map[name] for name in local_minim_names]
00114 #     trjcat_many(hashed_names, past_dir, './combinded_traj_basin.xtc')
00115 #
00116 #     if os.path.exists('./local_minim_bas.xtc'):
00117 #         gmx_trjcat(f=['./combinded_traj_basin.xtc', 'local_minim_bas.xtc'],
00118 #                    o='./combinded_traj_basin_comb.xtc',
00119 #                    n='./prot_dir/prot.ndx', cat=True, vel=False, sort=False, overwrite=True)
00120 #         os.remove('./combinded_traj_basin.xtc')
00121 #         os.rename('./combinded_traj_basin_comb.xtc', './combinded_traj_basin.xtc')
00122 #
00123 #     gmx_trjcat(f=['./combinded_traj_basin.xtc', 'local_min.xtc'],
00124 #                o='./local_minim_bas.xtc',
00125 #                n='./prot_dir/prot.ndx', cat=True, vel=False, sort=False, overwrite=True)
00126 #
00127 #     rmsd = get_knn_dist_mdsctk('./combinded_traj_openq.xtc', './combinded_traj_basin.xtc', goal_top)
00128 #
00129 #     rmsd_structured = list()
00130 #     for i in range(len(closest_to_minim)):
00131 #         rmsd_structured.append(rmsd[i * len(hashed_names):(i + 1) * len(hashed_names)])
00132 #
```

```
00133 #     for line_of_points in rmsd_structured:
00134 #         for point_pos, point in enumerate(line_of_points):
00135 #             if point < basin_err:
00136 #                 set_of_points_to_remove.add(point_pos)
00137 #
00138 #     print('Total points to trim: {} points'.format(len(set_of_points_to_remove)))
00139 #
00140 #     open_queue = [node for index, node in enumerate(open_queue) if index not in set_of_points_to_remove]
00141 #     # heapq.heappush(open_queue, elem)
00142 #     heapq.heapify(open_queue)
00143 #     return open_queue
00144
00145
00146 # def check_local_minimum(temp_xtc_file: str, goal_top: str, tol_error: float):
00147 #     """
00148 #     Checks whether tested frames are close to the local minima basin
00149 #     :param temp_xtc_file: frames to check
00150 #     :param goal_top: .top - topology of the NMR conformation
00151 #     :param tol_error: minimal metric vibration of the NMR structure
00152 #     :return: True if belongs, False otherwise
00153 #     """
00154 #     if os.path.exists('./local_minim_bas.xtc'):
00155 #         strict = True
00156 #         if strict:
00157 #             prune_err = tol_error*4
00158 #         else:
00159 #             prune_err = tol_error * 2
00160 #         min_dist = min(get_knn_dist_mdsctk(temp_xtc_file, 'local_minim_bas.xtc', goal_top))
00161 #         if min_dist < prune_err:
00162 #             return False
00163 #     return True
00164
00165
00166 def queue_rebuild(process_queue: list, open_queue_to_rebuild: list, node_info: dict, cur_mult: float, new_metr_name: str, sep_proc: bool =
      True) -> list:
00167     """Resorts the queue according to the new metric.
00168
00169     Args:
00170         :param list process_queue: queue to use if function is executed in a separate process
00171         :param list open_queue_to_rebuild: sorted queue that contains nodes about to be processed. This is actually only a partial queue (only
      top elements)
00172         :param dict node_info:
00173         :param float cur_mult: current greedy factor
00174         :param str new_metr_name: defines how to sort the new queue
00175         :param bool sep_proc: whether the function runs in a separate process
00176
00177     Returns:
00178         :return: if separate process - then new queue and metric name are pushed into the queue, otherwise returned
00179         :rtype: list
00180     """
00181     gc.collect()
00182     new_queue = list()
00183     to_goal, total = '{}_to_goal'.format(new_metr_name), '{}_dist_total'.format(new_metr_name)
00184     try:
00185         for elem in open_queue_to_rebuild[1:]:
00186             heapq.heappush(new_queue, (cur_mult*node_info[elem[2]][total] + node_info[elem[2]][to_goal], 0, elem[2]))
00187     except Exception:
00188         print(len(node_info))
00189         print(len(open_queue_to_rebuild))
00190         print(new_metr_name)
00191         print(cur_mult)
00192         print(sep_proc)
00193     del open_queue_to_rebuild
00194     gc.collect()
00195     if sep_proc:
00196         process_queue.put((new_queue, new_metr_name))
00197     else:
00198         return new_queue
00199
00200
00201 def get_atom_num(ndx_file: str) -> int:
00202     """Computes number of atoms in the particular index file.
00203
00204     Args:
00205         :param str ndx_file: .ndx - index of the protein atoms of the current conformation.
00206
00207     Returns:
00208         :return: number of atoms in the .ndx file.
00209         :rtype: int
00210     """
00211     with open(ndx_file, 'r') as index_file:
```

```
00212         index_file.readline()  # first line is the comment - skip it
00213         indices = index_file.read().strip()
00214     elems = indices.split()
00215     atom_num = len(elems)
00216     return atom_num
00217 
00218 
00219 def parse_hostnames(seednum: int, hostfile: str = 'hostfile') -> tuple:
00220     """Spreads the load among the hosts found in the hostfile. Needed for MPI
00221 
00222     Args:
00223         :param seednum: total number of seeds used in the current run
00224         :param hostfile: filename of the hostfile
00225 
00226     Returns:
00227         :return: hosts split partitioned according to the number of seeds and total number of cores for each job
00228     """
00229     with open(hostfile, 'r') as f:
00230         hosts = f.readlines()
00231     del hostfile
00232     hostnames = [elem.strip().split(' ')[0] for elem in hosts]
00233     ncores = [int(elem.strip().split(' ')[1].split('=')[1]) for elem in hosts]
00234     ev_num = len(hosts) // seednum
00235     if ev_num == 0:
00236         raise Exception('Special case is not implemented')
00237     else:
00238         chopped = [tuple(hostnames[i:i+ev_num]) for i in range(0, len(hostnames), ev_num)]
00239         ncores_sum = [sum(ncores[i:i+ev_num]) for i in range(0, len(ncores), ev_num)]
00240     return chopped, ncores_sum
00241 
00242 
00243 def compute_on_local_machine(cpu_map: list, seed_list: list, cur_name: str, past_dir: str, work_dir: str, seed_dirs: dict,
00244                              topol_file_init: str, ndx_file_init: str, prev_runs_files: list, old_name_digest: str) -> tuple:
00245     """This version is optimised for usage on one machine with tMPI (see GROMACS docs).
00246 
00247     Performs check whether requested simulation was completed in the past.
00248     If so (and all requested files exist), we skip the computation,
00249     otherwise we start the sequence of events that prepare and run the simulation in the separate process.
00250     I was playing with better core distribution, but it did not work well, since GROMACS may complain when you assign odd number of cores, or
       when 14 cores does not work, but 12 and 16 are fine.
00251     What I know fo sure that powers of 2 work the best until 128 cores, but we do not have so many cores on one machine.
00252     Two machines are worse than one (yes, 64+64 is slower than 64, same with 32+32) - maybe Infiniband can help, but we do not have one.
00253     Additionally, I commented prev_runs - it just uses more RAM without giving any significant speedup.
00254 
00255     Args:
00256         :param list cpu_map: number of cores for particular task (seed)
00257         :param list seed_list: list of current seeds
00258         :param str cur_name: name of the current node (prior path constructed from seed names s_0_1_4)
00259         :param str past_dir: path to the directory with prior computations
00260         :param str work_dir: path to the directory where seed dirs reside
00261         :param dict seed_dirs: dict which contains physical path to the directory where simulation with particular seed is performed
00262         :param str topol_file_init: .top - topology of the initial (unfolded) conformation
00263         :param str ndx_file_init: .ndx - index of the protein atoms of the unfolded conformation
00264         :param list prev_runs_files: information about all previously generated files in ./past directory
00265         :param str old_name_digest: digest of the current name
00266 
00267     Returns:
00268         :return: array of PIDs to join them later and allow some more parallel computation, hash names, simulation names.
00269         :rtype: tuple
00270 
00271     Returns: PIDs and new filenames. PIDs - to join processes later.
00272     """
00273     files_for_trjcat = list()
00274     recent_filenames = list()
00275     pid_arr = list()
00276     # global extra_past
00277     # recent_n2d = dict()
00278     # recent_d2n = dict()
00279     for i, exec_group in enumerate(cpu_map):
00280         saved_cores = 0
00281         for cur_group_sched in exec_group:
00282             cores, seed_2_process = cur_group_sched
00283             seed_2_process = seed_list[seed_2_process]
00284             new_name = '{}_{}'.format(cur_name, seed_2_process)
00285             seed_digest_filename = get_digest(new_name)
00286             # recent_n2d[new_name] = seed_digest_filename
00287             # recent_d2n[seed_digest_filename] = new_name
00288             xtc_filename = '{}.xtc'.format(seed_digest_filename)
00289             gro_filename = '{}.gro'.format(seed_digest_filename)
00290 
00291             files_for_trjcat.append(os.path.join(past_dir, xtc_filename))
```

```
00292             # if xtc_filename in prev_runs_files and gro_filename in prev_runs_files:
00293             # # if os.path.exists(os.path.join('./past', xtc_filename)) and os.path.exists(os.path.join('./past', gro_filename)):
00294             #     saved_cores += cores  # not fair, but short TODO: write better logic for cores remapping
00295             #     recent_filenames.append(xtc_filename)
00296             #     recent_filenames.append(gro_filename)
00297             #     continue
00298             # else:
00299             if not (os.path.exists(os.path.join(past_dir, xtc_filename)) and os.path.exists(os.path.join(past_dir, gro_filename))): #\
00300                 # and not (os.path.exists(os.path.join(extra_past, xtc_filename)) and os.path.exists(os.path.join(extra_past, gro_filename))):
00301                 md_process = None
00302                 md_process = mp.Process(target=make_a_step,
00303                                         args=(work_dir, seed_2_process, seed_dirs, topol_file_init, ndx_file_init,
00304                                               seed_digest_filename, old_name_digest, past_dir, cores + saved_cores))
00305                 md_process.start()
00306                 # print('Process started :{} pid:{} alive:{} ecode:{} with next param: s:{}, pd:{}, cor:{}'.format(md_process.name,
00307                 # md_process.pid, md_process.is_alive(), md_process.exitcode, seed_2_process, past_dir, cores+saved_cores))
00308                 pid_arr.append(md_process)
00309                 # make_a_step(work_dir, seed_2_process, seed_dirs, seed_list, topol_file, ndx_file, name_2_digest_map,
00310                 # cur_job_name, past_dir, cores+saved_cores)
00311                 saved_cores = 0
00312                 # print('md_process{} '.format(seed_2_process), end="")
00313                 # recent_filenames.append(xtc_filename)
00314                 # recent_filenames.append(gro_filename)
00315         if i is not len(cpu_map) - 1:  # if it is not the last portion of threads then wait for completion
00316             [proc.join() for proc in pid_arr]
00317 
00318     # combine prev_step and goal to compute two dist in one pass
00319     # rm_queue.join()  # make sure that queue is empty (all files were deleted)
00320 
00321     # Test code for multiprocessing check. There was a problem with python3.4 and old sqlite (too many parallel
00322     # connections when reusing past results).
00323     # [proc.join(timeout=90) for proc in pid_arr]
00324     # if len(pid_arr):
00325     #     print('Proc arr is not empty:', end=' ')
00326     #     while True:
00327     #         proc_stil_running = 0
00328     #         for cur_group_sched in pid_arr:
00329     #             print('waiting for name:{} pid:{} alive:{} ecode:{}'.format(cur_group_sched.name,
00330     #             cur_group_sched.pid, cur_group_sched.is_alive(), cur_group_sched.exitcode))
00331     #             cur_group_sched.join(timeout=40)
00332     #             if cur_group_sched.exitcode is not None:
00333     #                 proc_stil_running += 1
00334     #         if proc_stil_running == len(pid_arr):
00335     #             print('Done.')
00336     #             break
00337 
00338     # if len(pid_arr):
00339     #     print('j{} '.format(len(pid_arr)), end="")
00340     return pid_arr, files_for_trjcat, recent_filenames, None, None  # recent_n2d, recent_d2n
00341 
00342 
00343 def compute_with_mpi(seed_list: list, cur_name: str, past_dir: str, work_dir: str, seed_dirs: dict, topol_file_init: str,
00344                      ndx_file_init: str, prev_runs_files: list, old_name_digest: str, tot_seeds: int, hostnames: list,
00345                      ncores: list, sched: bool = False, ntomp: int = 1) -> tuple:
00346     """This version is optimised for usage on more than one machine with tMPI and/or MPI.
00347 
00348     If you use scheduler and know exactly how many cores each machine has - supply correct hostfile and use tMPI on each machine with OMP.
00349     If you use scheduler without option to choose specific machine - use version without scheduler or local version (depends on your cluster
       implementation).
00350     Performs check whether requested simulation was completed in the past.
00351     If so (and all requested files exist), we skip the computation,
00352     otherwise we start the sequence of events that prepare and run the simulation in the separate process.
00353     I was playing with better core distribution, but it did not work well, since GROMACS may complain when you assign odd number of cores, or
       when 14 cores does not work, but 12 and 16 are fine.
00354     What I know fo sure that powers of 2 work the best until 128 cores, but we do not have so many cores on one machine.
00355     Two machines are worse than one (yes, 64+64 is slower than 64, same with 32+32) - maybe InfiniBand can help, but we do not have one.
00356     Additionally, I commented prev_runs - it just uses more RAM without giving any significant speedup.
00357 
00358     Args:
00359         :param list seed_list: list of current seeds
00360         :param str cur_name: name of the current node (prior path constructed from seed names s_0_1_4)
00361         :param str past_dir: path to the directory with prior computations
00362         :param strwork_dir: path to the directory where seed dirs reside
00363         :param dict seed_dirs: dict which contains physical path to the directory where simulation with particular seed is performed
00364         :param str topol_file_init:  .top - topology of the initial (unfolded) conformation
00365         :param str ndx_file_init: .ndx - index of the protein atoms of the initial (unfolded) conformation
00366         :param list prev_runs_files: information about all previously generated files in ./past directory
00367         :param str old_name_digest: digest of the current name
00368         :param int tot_seeds: total number of seeds, controversial optimisation.
00369         :param list hostnames: correct names/IPs of the hosts
00370         :param int ncores: number of cores on each host
```

```
00371         :param bool sched: secelts proper make_a_step version
00372         :param int ntomp: how many OMP threads use during the MD simulation (2-4 is the optimal value on 32-64 core hosts)
00373 
00374     Returns:
00375         :return: array of PIDs to join them later and allow some more parallel computation, hash names, simulation names.
00376         :rtype: tuple
00377 
00378     PIDs and new filenames. PIDs - to join processes later.
00379     """
00380     # if os.path.exists(os.path.join(os.getcwd(), 'local.comp')):
00381     #     hostnames = [('Perseus', )]*tot_seeds
00382     gc.collect()
00383     files_for_trjcat = list()
00384     recent_filenames = list()
00385     pid_arr = list()
00386     # recent_n2d = dict()
00387     # recent_d2n = dict()
00388     for i in range(tot_seeds):
00389         seed_2_process = seed_list[i]
00390         new_name = '{}_{}'.format(cur_name, seed_2_process)
00391         seed_digest_filename = get_digest(new_name)
00392         # recent_n2d[new_name] = seed_digest_filename
00393         # recent_d2n[seed_digest_filename] = new_name
00394         xtc_filename = '{}.xtc'.format(seed_digest_filename)
00395         gro_filename = '{}.gro'.format(seed_digest_filename)
00396 
00397         # if os.path.exists(os.path.join(extra_past, xtc_filename)) and os.path.exists(os.path.join(extra_past, gro_filename)):
00398         #     files_for_trjcat.append(os.path.join(extra_past, xtc_filename))
00399         # else:
00400         files_for_trjcat.append(os.path.join(past_dir, xtc_filename))
00401 
00402         # if xtc_filename not in prev_runs_files or gro_filename not in prev_runs_files:
00403         if not (os.path.exists(os.path.join(past_dir, xtc_filename)) and os.path.exists(os.path.join(past_dir, gro_filename))): # \
00404                 # and not (os.path.exists(os.path.join(extra_past, xtc_filename)) and os.path.exists(os.path.join(extra_past, gro_filename))):
00405             # make_a_step2(work_dir, seed_2_process, seed_dirs, topol_file_init, ndx_file_init, seed_digest_filename, old_name_digest,
00406             # past_dir, hostnames[i], ncores[i])
00407             if sched:
00408                 md_process = mp.Process(target=make_a_step3,
00409                                         args=(work_dir, seed_2_process, seed_dirs, topol_file_init, ndx_file_init,
00410                                               seed_digest_filename, old_name_digest, past_dir, int(ncores/tot_seeds), ntomp))
00411             else:
00412                 md_process = mp.Process(target=make_a_step2,
00413                                         args=(work_dir, seed_2_process, seed_dirs, topol_file_init, ndx_file_init,
00414                                               seed_digest_filename, old_name_digest, past_dir, hostnames[i], ncores[i]))
00415             md_process.start()
00416             pid_arr.append(md_process)
00417         recent_filenames.append(xtc_filename)
00418         recent_filenames.append(gro_filename)
00419 
00420     return pid_arr, files_for_trjcat, recent_filenames, None, None  # recent_n2d, recent_d2n
00421 
00422 
00423 def check_in_queue(queue: list, elem_hash: str) -> bool:
00424     """Checks whether elements with provided hash exists in the queue
00425 
00426     Args:
00427         :param list queue: specific queue to check
00428         :param str elem_hash: name to find in the queue
00429 
00430     Returns:
00431         :return: True if element found, False otherwise
00432         :rtype: bool
00433     """
00434     for elem in queue:
00435         if elem[2] == elem_hash:
00436             return True
00437     return False
00438 
00439 
00440 def second_chance(open_queue: list, visited_queue: list, best_so_far_name: str, cur_metric: str, main_dict: dict,
00441                   node_max_att: int, cur_metric_name: str, best_so_far: str, tol_error: float, greed_mult: float) -> list:
00442     """Typically executed during the seed change.
00443 
00444     We want to give the second chance to a promising trajectories with different seeds. Typically, we allow up to 4 attempts.
00445     However, the best trajectories are always readded to the queue.
00446 
00447     Args:
00448         :param list open_queue: sorted queue that contains nodes about to be processed. This is actually only a partial queue (only top
       elements)
00449         :param list visited_queue: sorted queue that contains nodes  processed prior. This is actually only a partial queue (only top elements)
00450         :param str best_so_far_name: node with the closest distance to the goal according to
```

```
00451         the guiding metric - we want to keep it for a long time, with hope that it will jump over the energy barrier
00452         :param str cur_metric: index of the current metric
00453         :param dict main_dict: map with all the information (prior and goal distances for all metrics, names, hashnames, attempts, etc)
00454         :param int node_max_att: defines how many attempts each node can have
00455         :param str cur_metric_name: name of the current metric
00456         :param str best_so_far: name of the node with the closest metric distance to the goal
00457         :param float tol_error: minimal metric vibration of the NMR structure
00458         :param float greed_mult: greedy multiplier, used to assign correct metric value (ballance between optimality and greedyness)
00459 
00460     Returns:
00461         :return: short list of promising nodes, they will be merged with the open queue later
00462         :rtype: list
00463     """
00464 
00465     res_arr = list()
00466     recover_best = True
00467     for elem in open_queue:
00468         if elem[2] == best_so_far_name[cur_metric]:
00469             recover_best = False
00470             break
00471 
00472     for elem in visited_queue:  # elem structure: tot_dist, att, cur_name
00473         # we give node_max_att attempts for a node to make progress with different seed
00474         if (elem[1] < node_max_att and main_dict[elem[2]]['{}_to_goal'.format(cur_metric_name)] - best_so_far[cur_metric] <
       tol_error[cur_metric_name]):  # \
00475             # and elem[2] != best_so_far_name[cur_metric]:
00476             # or main_dict[elem[2]]['{}_to_goal'.format(cur_metric_name)] != best_so_far[cur_metric]:
00477             if elem[2] == best_so_far_name[cur_metric]:
00478                 if recover_best:
00479                     res_arr.append(elem)
00480                     recover_best = False
00481                     break
00482             else:
00483                 if elem[1] > 1 and check_in_queue(open_queue, elem[2]):
00484                     print('Not adding regular node (already in the queue)')
00485                 else:
00486                     res_arr.append(elem)
00487                     print('Readding "{}" with attempt counter: {} and dist: {}'.format(elem[2], elem[1], elem[0]))
00488 
00489     elem = main_dict[best_so_far_name[cur_metric]]
00490     if recover_best:
00491         res_arr.append((elem['{}_dist_total'.format(cur_metric_name)] * greed_mult + elem['{}_to_goal'.format(cur_metric_name)],
00492                         0, best_so_far_name[cur_metric]))
00493         print('Recovering best')
00494     else:
00495         print('Not recovering best (already in the open queue)')
00496     del elem
00497 
00498     return res_arr
00499 
00500 
00501 def check_dupl(name_to_check: str, visited_queue: list) -> list:
00502     """
00503     This function is just a detector of duplicates.
00504 
00505     Main source of dupplicates is when the algorithme gives the second chance to the same seed, but does not use it.
00506     This function checks whether specific name was used recently
00507 
00508     Args:
00509         :param name_to_check: name that is about to be sampled
00510         :param visited_queue: all previously used names
00511 
00512     Returns:
00513         :return: True if name was used recently, otherwise False
00514     """
00515     arr = [name[2] for name in visited_queue]
00516     if name_to_check in arr:
00517         print("Duplicate found in {} last elements, index: {}\nelem:{}".format(len(arr), arr.index(name_to_check), name_to_check))
00518         return True
00519     return False
00520 
00521 
00522 def GMDA_main(prev_runs_files: list, past_dir: str, print_queue: mp.JoinableQueue,
00523               db_input_queue: mp.JoinableQueue, copy_queue: mp.JoinableQueue, rm_queue: mp.JoinableQueue, tot_seeds: int = 4) -> NoReturn:
00524     """This is the main loop.
00525 
00526     Note that it has many garbage collector calls - it can slightly reduce the performance, but also reduces total memory usage.
00527     Feel free to comment them - they do not affect the algorithm
00528 
00529     Args:
00530         :param list prev_runs_files you may see this as the list of files found before the execution.
```

```
00531         We do not use it anymore to reduce the memory footprint.
00532         Instead we check existence of the file separately.
00533         :param str past_dir: location of all generated .gro, .xtc, metric values. Sequence of past seeds results in the unique name.
00534         :type past_dir: str
00535         :param mp.JoinableQueue print_queue: separate thread for printing operations, connected to the main process by Queue.
00536         It helps significantly during the restart without the previously saved state:
00537         you can query DB faster without waiting for printing operations to complete.
00538         :param mp.JoinableQueue db_input_queue:
00539         :param mp.JoinableQueue copy_queue: connection to the separate process that handled async copy. Should be rewriten with asyncio
00540         :param mp.JoinableQueue rm_queue: connection to the separate process that handled async rm. Should be rewriten with asyncio
00541         :param int tot_seeds: number of parallel seeds to be executed - very powerful knob
00542 
00543     Returns:
00544         :return: Nothing, once stop condition is reached, looping stops and returns to the parent to join/clean other threads
00545     """
00546     # prev_runs_files = None  # temp action - trying to save memory
00547     print('Main process rebuild_queue_process: ', os.getpid())
00548     gc.collect()
00549     prot_dir = os.path.join(os.getcwd(), 'prot_dir')
00550     if not os.path.exists(prot_dir):
00551         os.makedirs(prot_dir)
00552     print('Prot dir: ', prot_dir)
00553     # These files has to be in prot_dir
00554     init = os.path.join(prot_dir, 'init.gro')  # initial state, will be copied into work dir, used for MD
00555     goal = os.path.join(prot_dir, 'goal.gro')  # final state, will not be used, but needed for derivation of other files
00556 
00557     topol_file_init = os.path.join(prot_dir, 'topol_unfolded.top')  # needed for MD
00558     topol_file_goal = os.path.join(prot_dir, 'topol_folded.top')  # needed for MD
00559 
00560     ndx_file_init = os.path.join(prot_dir, 'prot_unfolded.ndx')  # needed for extraction of protein data
00561     ndx_file_goal = os.path.join(prot_dir, 'prot_folded.ndx')  # needed for extraction of protein data
00562 
00563     init_bb_ndx = os.path.join(prot_dir, 'bb_unfolded.ndx')
00564     goal_bb_ndx = os.path.join(prot_dir, 'bb_folded.ndx')
00565 
00566     # These files will be generated
00567     init_xtc = os.path.join(prot_dir, 'init.xtc')  # small version, used for rmsd
00568     goal_xtc = os.path.join(prot_dir, 'goal.xtc')  # small version, used for rmsd
00569     goal_prot_only = os.path.join(prot_dir, 'goal_prot.gro')  # needed for knn_rms
00570     init_prot_only = os.path.join(prot_dir, 'init_prot.gro')  # needed for contacts
00571     # goal_bb_gro = os.path.join(prot_dir, 'goal_bb.gro')
00572     goal_bb_xtc = os.path.join(prot_dir, 'goal_bb.xtc')
00573     goal_angle_file = os.path.join(prot_dir, 'goal_angle.dat')
00574     goal_sincos_file = os.path.join(prot_dir, 'goal_sincos.dat')
00575 
00576     # cp2(os.path.join(prot_dir, 'nmr.gro'), goal)
00577     # cp2(os.path.join(prot_dir, 'md_heated.gro'), goal)
00578 
00579     # h_ndx_file = os.path.join(prot_dir, 'prot_h.ndx')
00580 
00581     # create prot_only init and goal
00582     gmx_trjconv(f=init, o=init_xtc, n=ndx_file_init)
00583     gmx_trjconv(f=goal, o=goal_xtc, n=ndx_file_goal)
00584     gmx_trjconv(f=goal, o=goal_prot_only, n=ndx_file_goal, s=goal)
00585     gmx_trjconv(f=init, o=init_prot_only, n=ndx_file_init, s=init)
00586     gmx_trjconv(f=goal, o=goal_bb_xtc, n=goal_bb_ndx, s=goal)
00587 
00588     get_bb_to_angle_mdsctk(x=goal_bb_xtc, o=goal_angle_file)
00589     get_angle_to_sincos_mdsctk(i=goal_angle_file, o=goal_sincos_file)
00590 
00591     atom_num = get_atom_num(ndx_file_init)
00592     atom_num_bb = get_atom_num(goal_bb_ndx)
00593     angl_num = 2 * int(atom_num_bb / 3) - 2  # each bb amino acid has 3 atoms, thus 3 angles, we skip 1 since it is almost always 0.
00594     # In order to make plain you need three points, this is why you loos 2 elements. Last two do not have extra atoms to form a plain
00595 
00596     with open(goal_sincos_file, 'rb') as file:
00597         initial_1d_array = np.frombuffer(file.read(), dtype=np.float64, count=-1)
00598     goal_angles = np.reshape(initial_1d_array, (-1, angl_num*2))[0]
00599     del file, initial_1d_array
00600 
00601     cont_dist = 3.0
00602     goal_ind = get_contat_profile_mdsctk(goal_prot_only, goal_xtc, ndx_file_goal, cont_dist)[1:]  # first is total num of contacts
00603     goal_contacts = np.zeros(atom_num * atom_num, dtype=np.bool)
00604     goal_contacts[goal_ind] = True
00605     del goal_ind
00606 
00607     h_pos_goal = parse_top_for_h(topol_file_goal)
00608     h_filter_goal = np.zeros(atom_num * atom_num, dtype=np.bool)
00609     for pos in h_pos_goal:
00610         h_filter_goal[(pos - 1) * atom_num:pos * atom_num] = True
00611     del pos
```

```
00612     goal_cont_h = np.logical_and(goal_contacts, h_filter_goal)
00613
00614     h_pos_init = parse_top_for_h(topol_file_init)
00615     h_filter_init = np.zeros(atom_num * atom_num, dtype=np.bool)
00616     for pos in h_pos_init:
00617         h_filter_init[(pos - 1) * atom_num:pos * atom_num] = True
00618     del pos
00619
00620     # usually h_filter_init is the same as h_filter_goal since they share same force field
00621     if np.sum(np.logical_xor(h_filter_init, h_filter_goal)) > 0:
00622         print('Warning, H positions in init and goal are different')
00623     del h_pos_goal, h_pos_init
00624
00625     cpu_pool = mp.Pool(mp.cpu_count())
00626
00627     goal_contacts_and_sum = np.sum(goal_contacts)
00628     goal_contacts_xor_sum = get_native_contacts(goal_prot_only, [goal_xtc], ndx_file_goal, goal_contacts,
00629                                                 atom_num, cont_dist, np.logical_xor, pool=cpu_pool)[0]
00630     if goal_contacts_xor_sum != 0:
00631         raise Exception('goal.gro XOR goal.xtc is not 0 - they are different')
00632     else:
00633         del goal_contacts_xor_sum
00634     goal_contacts_and_h_sum = get_native_contacts(goal_prot_only, [goal_xtc], ndx_file_goal, goal_cont_h,
00635                                                   atom_num, cont_dist, np.logical_and, pool=cpu_pool)[0]
00636     # nat_contacts = np.sum(logic_fun(goal_contacts, init_contacts))
00637
00638     if not os.path.exists(init_xtc) or not os.path.exists(goal_xtc) or \
00639             not os.path.exists(topol_file_init) or not os.path.exists(ndx_file_init):
00640         print('Copy initial and final state in to prot_dir')
00641         exit("Copy initial and final state in to prot_dir")
00642
00643     work_dir = os.path.join(os.getcwd(), 'work_dir')  # either /dev/shm or os.getcwd()
00644
00645     # counter = 0
00646     # work_dir = os.path.join('/dev/shm', 'work_dir_{}'.format(counter))  # either /dev/shm or os.getcwd()
00647     # while os.path.exists(work_dir):
00648     #     counter += 1
00649     #     work_dir = os.path.join('/dev/shm', 'work_dir_{}'.format(counter))  # either /dev/shm or os.getcwd()
00650     # del counter
00651
00652     if not os.path.exists(work_dir):
00653         os.makedirs(work_dir)
00654     print('Work dir: ', work_dir)
00655
00656     if not os.path.exists(past_dir):
00657         os.makedirs(past_dir)
00658
00659     print('Past dir: ', past_dir)
00660
00661     simulation_temp = 300
00662
00663     print('Information about the protein:\nIt contains {} atoms and {} hydrogen contacts'
00664           '\n{} phipsi angles is going to be used as for angle distance'
00665           '\nthere are {} protein-protein contacts with distance {}A\nand {} protein-protein-h contacts with distance {}A.'
00666           '\nSimulation temp is set to {}K'
00667           ".format(atom_num, np.sum(goal_cont_h), angl_num, goal_contacts_and_sum, cont_dist,
00668                    goal_contacts_and_h_sum, cont_dist, simulation_temp))
00669
00670     seed_start = 0
00671     seed_list = list(range(seed_start, tot_seeds+seed_start))
00672     del seed_start
00673     seed_dirs = get_seed_dirs(work_dir, seed_list, simulation_temp)
00674     # rm_seed_dirs(seed_dirs)
00675
00676     if os.path.exists(os.path.join(os.getcwd(), 'local.comp')):
00677         use_mpi = False
00678     else:
00679         use_mpi = True
00680
00681     scheduler = False
00682     if scheduler:
00683         use_mpi = True
00684         core_map = 16
00685         nomp = 2
00686         hostnames = False
00687     else:
00688         nomp = False
00689         if use_mpi:
00690             hostnames, core_map = parse_hostnames(tot_seeds)
00691         else:
00692             cpu_map = create_core_mapping(nseeds=tot_seeds)
```

```
00693             hostnames = False
00694
00695
00696     metric_names =      ['RMSD', 'ANGL', 'AND_H', 'AND', 'XOR']
00697     metric_allowed_sc = [  20,     10,      5,      5,    10  ]
00698     alowed_metrics =    ['RMSD', 'ANGL', 'AND_H', 'AND', 'XOR']
00699     cur_metric = 0
00700     cur_metric_name = alowed_metrics[cur_metric]
00701     guiding_metric = 0  # main metric to tack global progress
00702
00703     num_metrics = len(metric_names)
00704
00705     an_file = 'ambient.noise'
00706     err_mult = 0.8
00707     tol_error = check_precomputed_noize(an_file, metric_names)
00708     if tol_error is None:
00709         goal_nz = os.path.join(prot_dir, 'folded_for_noise.gro')
00710         if hostnames:
00711             noize_file = gen_file_for_amb_noize(work_dir, seed_list, seed_dirs, ndx_file_goal,
00712                                                 topol_file_goal, goal_nz, hostnames, core_map)
00713         else:
00714             # noize_file = gen_file_for_amb_noize(work_dir, goal_nz, seed_list, seed_dirs, ndx_file_goal, topol_file_goal, goal_nz)
00715             noize_file = gen_file_for_amb_noize(work_dir, seed_list, seed_dirs, ndx_file_goal, topol_file_goal, goal_nz)
00716         # 0 - rmsd, 1 - angles, 2 - h_contacts, 3 - full_contacts_xor, 4 - full_contacts_and
00717     if tol_error is None or len(tol_error) < num_metrics:
00718         goal_prot_only_nz = os.path.join(prot_dir, 'goal_prot_nz.gro')
00719         gmx_trjconv(f=goal_nz, o=goal_prot_only_nz, n=ndx_file_goal, s=goal_nz)
00720         goal_angle_file_nz = os.path.join(prot_dir, 'goal_angle_nz.dat')
00721         goal_sincos_file_nz = os.path.join(prot_dir, 'goal_sincos_nz.dat')
00722         goal_bb_xtc_nz = os.path.join(prot_dir, 'goal_bb_nz.xtc')
00723         gmx_trjconv(f=goal_nz, o=goal_bb_xtc_nz, n=goal_bb_ndx, s=goal_nz)
00724         goal_xtc_nz = os.path.join(prot_dir, 'goal_nz.xtc')
00725         gmx_trjconv(f=goal_nz, o=goal_xtc_nz, n=ndx_file_goal)
00726         get_bb_to_angle_mdsctk(x=goal_bb_xtc_nz, o=goal_angle_file_nz)
00727         get_angle_to_sincos_mdsctk(i=goal_angle_file_nz, o=goal_sincos_file_nz)
00728         with open(goal_sincos_file_nz, 'rb') as file:
00729             initial_1d_array = np.frombuffer(file.read(), dtype=np.float64, count=-1)
00730         goal_angles_nz = np.reshape(initial_1d_array, (-1, angl_num * 2))[0]
00731         del file, initial_1d_array
00732         goal_ind_nz = get_contat_profile_mdsctk(goal_prot_only, goal_xtc, ndx_file_goal, cont_dist)[1:]  # first is total num of contacts
00733         goal_contacts_nz = np.zeros(atom_num * atom_num, dtype=np.bool)
00734         goal_contacts_nz[goal_ind_nz] = True
00735         del goal_ind_nz
00736
00737         h_pos_goal_nz = parse_top_for_h(topol_file_goal)
00738         h_filter_goal_nz = np.zeros(atom_num * atom_num, dtype=np.bool)
00739         for pos in h_pos_goal_nz:
00740             h_filter_goal_nz[(pos - 1) * atom_num:pos * atom_num] = True
00741         del h_pos_goal_nz, pos
00742         goal_cont_h_nz = np.logical_and(goal_contacts_nz, h_filter_goal_nz)
00743
00744         goal_contacts_and_h_sum_nz = get_native_contacts(goal_prot_only_nz, [goal_xtc_nz], ndx_file_goal, goal_cont_h_nz,
00745                                                          atom_num, cont_dist, np.logical_and, pool=cpu_pool)[0]
00746         goal_contacts_and_sum_nz = np.sum(goal_contacts_nz)
00747         err_node_info = compute_init_metric(past_dir, tot_seeds, noize_file, goal_xtc_nz, goal_prot_only_nz, angl_num, goal_bb_ndx,
00748                                             goal_angles_nz, goal_prot_only_nz, ndx_file_goal, goal_cont_h_nz, atom_num, cont_dist,
00749                                             h_filter_goal_nz, goal_contacts_nz, goal_contacts_and_h_sum_nz, goal_contacts_and_sum_nz)
00750         tol_error = dict()
00751         for metr_name in metric_names:
00752             tol_error[metr_name] = min([node['{}_to_goal'.format(metr_name)] for node in err_node_info]) * err_mult
00753         save_an_file(an_file, tol_error, metric_names)
00754         del err_node_info, metr_name
00755     del an_file
00756
00757     print('Done measuring ambient noise for folded state at {}K.\n'
00758           'Min result for {} seeds was multiplied by {}.\n'
00759           'RMSD noise was {:0.5f}A\n'
00760           'PhiPsi angle noise was {:0.5f}\n'
00761           'Contact distance noise with AND logical function for H contacts was {:.3f}\n'
00762           'Contact distance noise with AND logical function was {:.3f}\n'
00763           'Contact distance noise with XOR logical function was {:.3f}\n'
00764           ".format(simulation_temp, tot_seeds, err_mult, tol_error['RMSD'], tol_error['ANGL'], tol_error['AND_H'],
00765                    tol_error['AND'], tol_error['XOR']))
00766     del err_mult
00767     node_info = compute_init_metric(past_dir, 1, init_xtc, goal_xtc, goal_prot_only, angl_num, init_bb_ndx, goal_angles, init_prot_only,
00768                                     ndx_file_init, goal_cont_h, atom_num, cont_dist, h_filter_init, goal_contacts,
00769                                     goal_contacts_and_h_sum, goal_contacts_and_sum)
00770
00771     print('Done measuring distance from initial state at {}K.\n'
00772           'RMSD dist: {:0.5f}A\n'
00773           'PhiPsi angle difference: {:0.5f}\n'
```

```
00774           'H contact disagreement (AND_H): {} of {}\n'
00775           'All contact disagreement (AND): {} of {}\n'
00776           'All contact disagreement (XOR): {}\n'.format(simulation_temp,
00777                                                         node_info['RMSD_to_goal'],
00778                                                         node_info['ANGL_to_goal'],
00779                                                         node_info['AND_H_to_goal'], goal_contacts_and_h_sum,
00780                                                         node_info['AND_to_goal'], goal_contacts_and_sum,
00781                                                         node_info['XOR_to_goal']))
00782     print('Unfolded to noise ratio:\n'
00783           'RMSD : {:.5f}\n'
00784           'PhiPsi angles: {:.5f}\n'
00785           'H contact (AND_H) disagreement: {:.5f}\n'
00786           'All contact (AND) disagreement: {:.5f}\n'
00787           'All contact disagreement (XOR): {:.5f}\n'.format(node_info['RMSD_to_goal'] / tol_error['RMSD'],
00788                                                             node_info['ANGL_to_goal'] / tol_error['ANGL'],
00789                                                             node_info['AND_H_to_goal']/tol_error['AND_H'],
00790                                                             node_info['AND_to_goal'] / tol_error['AND'],
00791                                                             node_info['XOR_to_goal'] / tol_error['XOR']))
00792
00793     # part of code used to study relation between contact distance and noise
00794     # f.write(
00795     #     '{} \n'.format(' '.join(str(elem) for elem in [cont_dist, node_info['AND_H_to_goal'], goal_contacts_and_h_sum,
00796     #     node_info['AND_H_to_goal'] / goal_contacts_and_h_sum, node_info['AND_to_goal'],
00797     #                                                    goal_contacts_and_sum,
00798     #                                                    node_info['AND_to_goal'] / goal_contacts_and_sum, node_info['XOR_to_goal'],
00799     #                                                    node_info['AND_H_to_goal'] / tol_error['AND_H'],
00800     #                                                    node_info['AND_to_goal'] / tol_error['AND'],
00801     #                                                    node_info['XOR_to_goal'] / tol_error['XOR']])))
00802     # print('done writing the file')
00803     # exit(22)
00804     # name_2_digest_map = dict()
00805     # digest_2_name_map = dict()
00806     # name_2_digest_map['s'] = get_digest('s')
00807     cur_hash_name = get_digest('s')
00808     # digest_2_name_map[name_2_digest_map['s']] = 's'
00809
00810     main_dict = dict()
00811     main_dict[cur_hash_name] = node_info
00812
00813     open_queue = list()
00814     heapq.heappush(open_queue, (node_info['RMSD_to_goal'], 0, cur_hash_name))  # metric_val, attempts, name
00815
00816     cp2(init_xtc[:-4] + '.gro', os.path.join(past_dir, cur_hash_name + '.gro'))
00817     cp2(init_xtc[:-4] + '.xtc', os.path.join(past_dir, cur_hash_name + '.xtc'))
00818     # copy_queue.put_nowait((init_xtc[:-4] + '.gro', os.path.join(past_dir, name_2_digest_map['s'] + '.gro')))
00819     # copy_queue.put_nowait((init_xtc[:-4] + '.xtc', os.path.join(past_dir, name_2_digest_map['s'] + '.xtc')))
00820     # copy_queue.put_nowait(None)
00821
00822     visited_queue = list()
00823     skipped_counter = 0
00824
00825     combined_pg = os.path.join(work_dir, "out.xtc")
00826     temp_xtc_file = os.path.join(work_dir, "temp.xtc")
00827     # temp_xtc_file_bb = os.path.join(work_dir, "temp_bb.xtc")
00828
00829     loop_start = time.perf_counter()
00830
00831     # info_form_str = 'n:{}\db_input_thread:{:.4f}\tg:{:.4f}\ts:{}\tq:{}\tv:{}\tl:{:.2f}s\tc:{:.2f}s'
00832     info_form_str = 'o_q:{:<5}  v_q:{:<3}  s:{:<3}  grm:{:6.3f}  gan:{:6.3f}  gah:{:<4}  gad:{:<4}  gxo:{:<4}  ' \
00833                     't:{:5.2f}s  gbr:{:.4f}  gba:{:.4f}  gc:{:<2}  ns:{:3.1f}  sc:{}'
00834     #  node_info['rmds_total'], node_info['rmds_to_goal'], skipped_counter, len(open_queue), len(visited_queue),
00835     # loop_end - loop_start, best_so_far, global_best_so_far, greed_count, greed_mult, seed_change_counter,
00836     # node_info['nat_cont_to_goal']))
00837     # info_form_str.format(len(open_queue), len(visited_queue), skipped_counter, node_info['RMSD_to_goal'],
00838     # node_info['ANGL_to_goal'], node_info['AND_H_to_goal'],
00839     #                      node_info['AND_to_goal']), node_info['XOR_to_goal'], loop_end - loop_start, best_so_far[1],
00840     #                      best_so_far[0], greed_count, greed_mult, seed_change_counter)
00841     under_form_str = '{}_{}'
00842
00843     greed_mult = 1.0
00844     greed_count = 0
00845
00846     # con, dbname = get_db_con(tot_seeds)
00847     # insert_into_main_stor(con, node_info, greed_count, name_2_digest_map['s'], 's')
00848     db_input_queue.put_nowait((insert_into_main_stor, (node_info, greed_count, cur_hash_name, 's')))
00849
00850     node_max_att = 4
00851
00852     seed_change_counter = 0
00853     # change_metrics_limit = 3  # how many seed changes(20 iter per change) with no problems we have to have to change cur metricss
00854
```

```
00855     # search LMA in the code
00856     # seed_change_limit = 1000
00857     # local_minimum_counter = 0
00858     # local_minim_names = list()
00859
00860     # nmr_structure_switch = 2  # 0 for nmr, 1 for relaxed, 2 for heated
00861
00862     best_so_far = [node_info['{}_to_goal'.format(metr)] for metr in metric_names]
00863     print(best_so_far)
00864     best_so_far_name = [cur_hash_name] * num_metrics
00865     # global_best_so_far = best_so_far
00866
00867     Path(combined_pg).touch()
00868     Path(temp_xtc_file).touch()
00869     if os.path.exists('./local_min.xtc'):
00870         os.remove(('./local_min.xtc'))
00871
00872     compute_all_at_once = True
00873     counter_since_seed_changed = 0
00874
00875     recover = False  # STOP! before changing this toggle read bellow:
00876     # 1. Make backup of your pickles
00877     # 2. Remember number of the last good db - this name should always be the last one
00878     # There was no proper testing of this functionality and backups may overwrite last good state
00879     # Backups rely on time and number of steps, but if you have too fast/slow I/O - everything may go wrong. Thus do the pickle backup.
00880     if recover:  # this can (and should) be done in parallel or instead of most var initialization (much earlier)
00881         visited_queue, open_queue, main_dict = main_state_recover()
00882         prev_state = supp_state_recover()
00883         tol_error, seed_list, seed_dirs, seed_change_counter, skipped_counter, \
00884         cur_metric_name, cur_metric, counter_since_seed_changed, guiding_metric, greed_mult, \
00885         best_so_far_name, best_so_far, greed_count = prev_state
00886         del prev_state
00887         copy_old_db(list(main_dict.keys()), visited_queue[-3:].copy()[::-1], open_queue[0][2], greed_count-1)
00888
00889     # try:
00890     # aa = 0
00891     iter_from_bak = 0
00892     time_for_backup = False
00893     bak_time_check = time.perf_counter()
00894     while len(open_queue) > 0:  # and aa < 137:
00895         gc.collect()
00896         # if not aa % 10:
00897         #     # Prints out a summary of the large objects
00898         #     summary.print_(summary.summarize(muppy.get_objects()))
00899         # aa +=1
00900         new_elem = heapq.heappop(open_queue)  # tot_dist, att, name
00901         tot_dist, att, cur_hash_name = new_elem
00902         del new_elem
00903         if counter_since_seed_changed:  # you may disable this check, it was here to track nodes with the same name.
00904             if check_dupl(cur_hash_name, visited_queue[-counter_since_seed_changed:]):
00905                 continue
00906         # however, if you see nodes with the same name - check real name and if it is different - change hashing function
00907         # much
00908         counter_since_seed_changed += 1
00909
00910         node_info = main_dict[cur_hash_name]
00911         cur_name = zlib.decompress(node_info['native_name']).decode()
00912         # cur_file = os.path.join(past_dir, node_info['digest_name'])
00913
00914         visited_queue.append((tot_dist, att+1, cur_hash_name))  # TODO: trim it when size > 500 by 300, update tot_trim
00915         del tot_dist, att
00916
00917         db_input_queue.put_nowait((insert_into_visited, (cur_hash_name, greed_count)))
00918         db_input_queue.put_nowait((insert_into_log, ('result', cur_hash_name, 'WQ', 'VIZ', best_so_far, greed_count, greed_mult,
00919                                                      node_info['{}_dist_total'.format(cur_metric_name)],
00920                                                      node_info['{}_to_goal'.format(cur_metric_name)], cur_metric_name)))
00921         # insert_into_visited(con, cur_name, greed_count)
00922         # insert_into_log(con, 'result', cur_name, 'WQ', 'VIZ', best_so_far, greed_count, greed_mult, node_info['{}_dist_total'.
00923         #                 format(cur_metric_name)], node_info['{}_to_goal'.format(cur_metric_name)])
00924         loop_end = time.perf_counter()
00925
00926         print_queue.put_nowait((info_form_str,
00927                                 ((len(open_queue), len(visited_queue), skipped_counter, node_info['RMSD_to_goal'],
00928                                   node_info['ANGL_to_goal'], node_info['AND_H_to_goal'], node_info['AND_to_goal'],
00929                                   node_info['XOR_to_goal'], loop_end - loop_start, best_so_far[0], best_so_far[1],
00930                                   greed_count, greed_mult, seed_change_counter))))
00931         # print(info_form_str.format(len(open_queue), len(visited_queue), skipped_counter, node_info['RMSD_to_goal'],
00932         #                            node_info['ANGL_to_goal'], node_info['AND_H_to_goal'], node_info['AND_to_goal'],
00933         #                            node_info['XOR_to_goal'], loop_end - loop_start, best_so_far[0], best_so_far[1],
00934         #                            greed_count, greed_mult, seed_change_counter))
00935
```

```
00936         # if node_info['ANGL_to_goal'] < best_so_far[1]:
00937         #     print('BSF:')
00938         #     print(best_so_far)
00939         #     print('Cur node info ANGL'.format(node_info['ANGL_to_goal']))
00940         #     print('Cur node info name'.format(cur_name))
00941         #     raise Exception('Error in best so far')
00942 
00943         loop_start = time.perf_counter()
00944         if not use_mpi:
00945             pid_arr, files_for_trjcat, recent_filenames, recent_n2d, recent_d2n = compute_on_local_machine(cpu_map, seed_list, cur_name,
00946                                                                                                            past_dir, work_dir, seed_dirs,
00947                                                                                                            topol_file_init, ndx_file_init,
00948                                                                                                            prev_runs_files,
00949                                                                                                            cur_hash_name)
00950         else:
00951             pid_arr, files_for_trjcat, recent_filenames, recent_n2d, recent_d2n = compute_with_mpi(seed_list, cur_name, past_dir, work_dir,
00952                                                                                                    seed_dirs, topol_file_init,
00953                                                                                                    ndx_file_init, prev_runs_files,
00954                                                                                                    cur_hash_name, tot_seeds, hostnames,
00955                                                                                                    core_map, scheduler, nomp)
00956 
00957         # update map
00958         # name_2_digest_map.update(recent_n2d)
00959         # digest_2_name_map.update(recent_d2n)
00960         # update prev files
00961         # prev_runs_files.extend(recent_filenames)
00962         if prev_runs_files:
00963             if len(prev_runs_files) <= tot_seeds*2:  # gro+xtc - two types
00964                 prev_runs_files = None
00965             else:
00966                 for file in recent_filenames:
00967                     try:
00968                         prev_runs_files.remove(file)
00969                     except Exception:
00970                         pass  # this is not an error - this behaviour is expected when you started following other route.
00971                         # print("Was not able to remove {}, list size: {}".format(file, len(prev_runs_files)))
00972                 del file
00973         del recent_filenames, recent_n2d, recent_d2n
00974 
00975         os.remove(combined_pg)
00976         gmx_trjcat(f=['{}.xtc'.format(os.path.join(past_dir, cur_hash_name)), goal_xtc],
00977                    o=combined_pg, n=ndx_file_init, cat=True, vel=False, sort=False, overwrite=True)
00978 
00979         [proc.join() for proc in pid_arr]
00980         del pid_arr
00981 
00982         if compute_all_at_once or cur_metric < 2:
00983             os.remove(temp_xtc_file)
00984             gmx_trjcat(f=files_for_trjcat, o=temp_xtc_file, n=ndx_file_init, cat=True, vel=False, sort=False, overwrite=True)
00985 
00986         new_nodes_names = [under_form_str.format(cur_name, seed_name) for seed_name in seed_list]
00987         # for i, node in enumerate(new_nodes):
00988         #     new_nodes[i]['digest_name'] = get_digest(new_nodes_names[i])
00989         #     # new_nodes[i]['native_name'] = new_nodes_names[i]
00990         #     new_nodes[i]['native_name'] = zlib.compress(new_nodes_names[i].encode(), 9)
00991         # del node, i
00992         new_nodes, metric_to_goal, metric_form_prev, metric_to_tot = compute_metric(past_dir, new_nodes_names, tot_seeds, combined_pg,
00993                                                                                     temp_xtc_file, goal_prot_only, node_info, angl_num,
00994                                                                                     init_bb_ndx, goal_angles, init_prot_only,
00995                                                                                     files_for_trjcat, ndx_file_init, goal_cont_h,
00996                                                                                     atom_num, cont_dist, h_filter_init, goal_contacts,
00997                                                                                     cur_metric, goal_contacts_and_h_sum,
00998                                                                                     goal_contacts_and_sum, prev_runs_files is not None,
00999                                                                                     cpu_pool=cpu_pool,
01000                                                                                     compute_all_at_once=compute_all_at_once)
01001         del files_for_trjcat
01002 
01003         new_filtered = list()
01004         for i in range(tot_seeds):
01005             # if seed_change_counter:
01006             #     local_minim_names.append(seed_name)
01007 
01008             # MAIN INSERT  new_nodes, metric_form_prev, metric_to_goal, metric_to_tot
01009             # we have two conditions to get intro the queue:
01010             # 1st - get better than the best result (obvious)
01011             # 2nd - we have to make big enough step from the previous point
01012             # AND this step should bring us closer to the goal 1/2 of just a noise
01013             if (metric_form_prev[i] > tol_error[cur_metric_name]
01014                 and metric_to_goal[i] - node_info['{}_to_goal'.format(cur_metric_name)] < tol_error[cur_metric_name] / 2) \
01015                     or metric_to_goal[i] <= best_so_far[cur_metric]:
01016                 # LMA - this approach is currently frozen since it did not show any benefits with RMSD,
```

```
01017                 # but was never adapted to multiple metrics
01018                 # if check_local_minimum(temp_xtc_file, goal_prot_only, tol_error):
01019                 # else:
01020                 #     print('point was on path to local minimum')
01021 
01022                 heapq.heappush(open_queue, (greed_mult * metric_to_tot[i] + metric_to_goal[i], 0, new_nodes[i]['digest_name']))
01023                 new_filtered.append((greed_mult * metric_to_tot[i] + metric_to_goal[i], 0, new_nodes[i]['digest_name']))
01024                 # insert_into_main_stor(con, new_nodes[i], greed_count,
01025                 # name_2_digest_map[new_nodes_names[i]], new_nodes_names[i])
01026                 db_input_queue.put_nowait((insert_into_main_stor,
01027                                            (new_nodes[i], greed_count, new_nodes[i]['digest_name'], new_nodes_names[i])))
01028                 main_dict[new_nodes[i]['digest_name']] = new_nodes[i]
01029             else:
01030                 skipped_counter += 1
01031                 # insert_into_log(con, 'skip', cur_name, '', 'SKIP', best_so_far, greed_count,
01032                 # greed_mult, metric_form_prev[i], metric_form_prev[i])
01033                 db_input_queue.put_nowait((insert_into_log, ('skip', cur_hash_name, '', 'SKIP', best_so_far, greed_count,
01034                                                              greed_mult, metric_form_prev[i], metric_to_goal[i], cur_metric_name)))
01035         db_input_queue.put_nowait((insert_into_log, ('current', cur_hash_name, '', 'WQ', best_so_far, greed_count,
01036                                                      greed_mult, metric_form_prev, metric_to_goal, cur_metric_name)))
01037         del metric_to_tot, metric_form_prev, i, new_nodes_names
01038 
01039         if compute_all_at_once:
01040             for metr in metric_names:
01041                 if metr != cur_metric_name:
01042                     min_val = min([node['{}_to_goal'.format(metr)] for node in new_nodes])
01043                     if best_so_far[metric_names.index(metr)] > min_val:
01044                         # print('bsf["{}"]={:.4f}, min={:.4f}'.
01045                         # format(metr, best_so_far[metric_names.index(metr)], min_val), end=' ')
01046                         best_so_far[metric_names.index(metr)] = min_val
01047                     del min_val
01048                 # else:
01049                     # print('skipping "{}"'.format(metr), end=' ')
01050             del metr
01051         # print()
01052         if best_so_far[guiding_metric] >
       new_nodes[metric_to_goal.index(min(metric_to_goal))]['{}_to_goal'.format(metric_names[guiding_metric])]:
01053             seed_change_counter = 0
01054 
01055         if best_so_far[cur_metric] > min(metric_to_goal):
01056             best_so_far_new = min(metric_to_goal)
01057             best_so_far[cur_metric] = best_so_far_new
01058             best_so_far_name[cur_metric] = new_nodes[metric_to_goal.index(best_so_far_new)]['digest_name']
01059             db_input_queue.put_nowait((insert_into_log,
01060                                        ('prom_0', best_so_far_name[cur_metric], '', '', best_so_far, greed_count, greed_mult,
01061                                         new_nodes[metric_to_goal.index(best_so_far_new)]['{}_from_prev'.format(cur_metric_name)],
01062                                         new_nodes[metric_to_goal.index(best_so_far_new)]['{}_to_goal'.format(cur_metric_name)],
01063                                         cur_metric_name)))
01064             if guiding_metric == cur_metric or best_so_far[guiding_metric] >=
       new_nodes[metric_to_goal.index(best_so_far_new)]['{}_to_goal'.format(metric_names[guiding_metric])]:
01065                 for i in range(num_metrics):
01066                     if i != cur_metric:
01067                         best_so_far_name[i] = best_so_far_name[cur_metric]
01068                         best_so_far[i] = new_nodes[metric_to_goal.index(best_so_far_new)]['{}_to_goal'.format(metric_names[i])]
01069                 del i
01070             seed_change_counter = 0
01071 
01072             # local_minim_names = list()  # search for LMA
01073             # if global_best_so_far[cur_metric] > best_so_far_new:
01074             #     global_best_so_far[cur_metric] = best_so_far_new
01075 
01076             # This code is for multiple stage folding. Code has to be adapted for several metrics.
01077             # if len(visited_queue) > 1 and global_best_so_far < visited_queue[1][2]/5 and nmr_structure_switch == 1:
01078             #     print('Changing goal to nmr structure')
01079             #     cp2(os.path.join(prot_dir, 'nmr.gro'), goal)
01080             #     gmx_trjconv(f=goal, o=goal_xtc, n=ndx_file)
01081             #     gmx_trjconv(f=goal, o=goal_prot_only, n=ndx_file, s=goal)
01082             #     open_queue = recompute_rmsd_for_openq(open_queue, goal_xtc, name_2_digest_map, past_dir,
01083             #     goal_prot_only, greed_mult)
01084             #     best_so_far = open_queue[-1][2]
01085             #     nmr_structure_switch = 0
01086             # elif len(visited_queue) > 1 and global_best_so_far < visited_queue[1][2]/3 and nmr_structure_switch == 2:
01087             #     print('Changing goal to relaxed structure')
01088             #     cp2(os.path.join(prot_dir, 'relaxed.gro'), goal)
01089             #     gmx_trjconv(f=goal, o=goal_xtc, n=ndx_file)
01090             #     gmx_trjconv(f=goal, o=goal_prot_only, n=ndx_file, s=goal)
01091             #     open_queue = recompute_rmsd_for_openq(open_queue, goal_xtc, name_2_digest_map, past_dir,
01092             #     goal_prot_only, greed_mult)
01093             #     best_so_far = open_queue[-1][2]
01094             #     nmr_structure_switch = 1
01095 
```

```
01096             # This is part of local minimum approach (LMA) search for LMA in this code
01097             # if os.path.exists('./local_minim_bas.xtc'):
01098             #     os.remove('./local_minim_bas.xtc')
01099             del best_so_far_new
01100             if greed_mult < 1.0:  # perfect place to optimize queue rebuild
01101                 greed_count = max(0, 10 * (greed_count // 10) - 8)
01102                 if 100 < greed_count < 110:
01103                     greed_count = 101
01104                 else:
01105                     greed_mult = min(1.001 - min(1.0, (greed_count // 10) / 10), 1.0)
01106                     open_queue = queue_rebuild(None, open_queue, main_dict, greed_mult, cur_metric_name, sep_proc=False)
01107             else:
01108                 greed_count = 0
01109         else:
01110             greed_count += 1
01111 
01112             if greed_count in range(10, 101, 10):
01113                 # open_queue = rebuild_queue.get(timeout=1800)[0]  # 30min
01114                 open_queue = rebuild_queue.get()[0]  # 30min
01115                 if new_filtered:
01116                     for elem in new_filtered:
01117                         heapq.heappush(open_queue, elem)
01118                 # cur_metric = metric_names.index(cur_metric_name)
01119                 del rebuild_queue
01120                 # if not isinstance(rebuild_queue_process, mp.Process):
01121                 #     a=8
01122                 rebuild_queue_process.join()
01123 
01124             elif greed_count == 121:
01125                 seeds_next = get_new_seeds(seed_list)
01126                 seed_change_counter += 1
01127                 seed_dirs_next = get_seed_dirs(work_dir, seeds_next, simulation_temp)
01128                 # previously I passed here "seed_dirs", but decided to save RAM
01129                 if seed_change_counter > metric_allowed_sc[cur_metric]:
01130                     new_metr_name = select_metrics_by_snr(new_nodes, node_info, metric_names, tol_error,
01131                                                           compute_all_at_once, alowed_metrics, cur_metric_name)
01132                     rebuild_queue = mp.Queue()
01133                     # open_queue = queue_rebuild(None, open_queue, main_dict, greed_mult, new_metr_name, sep_proc=False)
01134                     rebuild_queue_process = mp.Process(target=queue_rebuild,
01135                                                        args=(rebuild_queue, open_queue, main_dict, greed_mult, new_metr_name))
01136                     # if not isinstance(rebuild_queue_process, mp.Process):
01137                     #     a = 8
01138                     rebuild_queue_process.start()
01139                     del new_metr_name
01140                 # TODO: local minimum has to be rethought and rewritten.
01141                 #  At this point (before multiple metrics) experiments show that is does not give any benefits
01142                 # if seed_change_counter == seed_change_limit:
01143                 #     seed_change_counter = 0
01144                 #     greed_count = 112
01145                 #     open_queue = proc_local_minim(open_queue, best_so_far_name[cur_metric], tol_error, ndx_file_init,
01146                 #     name_2_digest_map, goal_prot_only, local_minim_names)
01147                 #     local_minim_names = list()
01148                 #     best_so_far[cur_metric] = (init_distance[cur_metric] + best_so_far[cur_metric])/2
01149                 #     local_minimum_counter += 1
01150                 #     continue
01151         del metric_to_goal
01152 
01153         if greed_count in range(9, 100, 10):
01154             rebuild_queue = mp.Queue()
01155             greed_mult = min(1.001 - (greed_count+1) / 100, 1.0)
01156             rebuild_queue_process = mp.Process(target=queue_rebuild, args=(rebuild_queue, open_queue, main_dict,
01157                                                                            greed_mult, cur_metric_name))
01158             rebuild_queue_process.start()
01159         elif greed_count == 122:
01160             greed_count = 102
01161             if seed_change_counter > metric_allowed_sc[cur_metric]:
01162                 print('Switching metric from {} to '.format(cur_metric_name), end='')
01163                 open_queue, cur_metric_name = rebuild_queue.get()  # 30min
01164                 # open_queue, cur_metric_name = rebuild_queue.get(timeout=1800)  # 30min
01165                 print(cur_metric_name)
01166                 cur_metric = metric_names.index(cur_metric_name)
01167                 del rebuild_queue
01168                 rebuild_queue_process.join()
01169                 extra_elem_q = queue_rebuild(None, new_filtered, main_dict, greed_mult, cur_metric_name, sep_proc=False)
01170                 for elem in extra_elem_q:
01171                     heapq.heappush(open_queue, elem)
01172                 del extra_elem_q, elem
01173                 seed_change_counter = 0
01174                 # greed_count = 102
01175 
01176             if seeds_next:
```

```
01177                 seed_list = seeds_next
01178                 rm_seed_dirs(seed_dirs)
01179                 seed_dirs = seed_dirs_next
01180                 res_arr = second_chance(open_queue[0:min(len(open_queue)-1, max(40, 4*counter_since_seed_changed))],
01181                                         visited_queue[min(-1, -counter_since_seed_changed):],
01182                                         best_so_far_name, cur_metric, main_dict, node_max_att,
01183                                         cur_metric_name, best_so_far, tol_error, greed_mult)
01184                 counter_since_seed_changed = 0
01185                 for elem in res_arr:
01186                     heapq.heappush(open_queue, elem)
01187                     # print(elem)
01188                     db_input_queue.put_nowait((insert_into_log,
01189                                                ('result', cur_hash_name, 'VIZ', 'WQ', best_so_far, greed_count, greed_mult,
01190                                                 main_dict[elem[2]]['{}_from_prev'.format(cur_metric_name)],
01191                                                 main_dict[elem[2]]['{}_to_goal'.format(cur_metric_name)], cur_metric_name)))
01192             else:
01193                 print('\nOUT OF SEEDS\n')
01194                 greed_count = 102  # will be changed soon
01195             del seeds_next, seed_dirs_next
01196         del cur_hash_name, cur_name, new_nodes, node_info
01197         new_filtered.clear()
01198         iter_from_bak += 1
01199         if loop_start - bak_time_check > 60*60 and not time_for_backup:  # every hour
01200             if iter_from_bak < 1000:  # expected value 240 - means that we are computing (on 32 cores), but not reading from ./past, typical
       read speed 10 000 iterations/hour (for non SSD)
01201                 time_for_backup = True
01202             else:
01203                 iter_from_bak = 0
01204                 bak_time_check = loop_start
01205
01206         if time_for_backup and (greed_count in range(104, 109) or greed_count in range(113, 117) or greed_count in range(93, 97)):
01207             main_state_backup((visited_queue, open_queue, main_dict))
01208             supp_state_backup((tol_error, seed_list, seed_dirs, seed_change_counter, skipped_counter, cur_metric_name,
01209                                cur_metric, counter_since_seed_changed, guiding_metric, greed_mult,
01210                                best_so_far_name, best_so_far, greed_count))
01211             time_for_backup = False
01212             bak_time_check = time.perf_counter()
01213             iter_from_bak = 0
01214
01215
01216     # except (KeyboardInterrupt, Exception) as e:
01217     #     print('Got exception: ', e)
01218     #     exc_type, exc_obj, exc_tb = sys.exc_info()
01219     #     fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
01220     #     print(exc_type, fname, exc_tb.tb_lineno)
01221     #     # print('Dumping work_queue')
01222     #     # dump_the_queue('work_queue.txt', open_queue, visited_queue, init_rmsd, tol_error, skipped_counter)
01223     #     # print('Dumping visited_queue')
01224     #     # dump_the_queue('visited_queue.txt', visited_queue, visited_queue, init_rmsd, tol_error, skipped_counter)
01225     #     # print('Done dumping ')
01226     #     # exit(-1)
01227     #
01228     #     # if keyboard.is_pressed('md_process'):
01229     #     #     print('Dumping ')
01230     #     #     dump_the_queue('work_queue.txt', open_queue, visited_queue, init_rmsd, tol_error, skipped_counter)
01231     #     #     print('Dumping ')
01232     #     #     dump_the_queue('visited_queue.txt', visited_queue, visited_queue, init_rmsd, tol_error, skipped_counter)
01233     #     #     print('Done dumping ')
01234     #
01235     #     # ne = open_queue[0]
01236     #     # trav = ne[1]
01237     #     # to_goal = ne[2]
01238     #     # sds = ne[3]
01239     #     # tot_points = len(sds.split("_")) - 1
01240     #     # from_prev_dist, prev_goal_dist = current_job[1], current_job[2]
01241     #     # trav_from_prev = trav - from_prev_dist
01242     #     # coef_1 = 1 - to_goal / init_rmsd
01243     #     # coef_1_a = coef_1 / tot_points if tot_points != 0 else 9999
01244     #     # deriv = (prev_goal_dist - to_goal) / trav_from_prev  # this cannot be zero
01245     #     # full_line = '{:.5f} {:.5f} {:.5f} {:.5f} {:.5f} {:.5f} {}\n'.format(trav,
01246     #     #                                                                    to_goal,
01247     #     #                                                                    trav_from_prev,
01248     #     #                                                                    coef_1,
01249     #     #                                                                    coef_1_a,
01250     #     #                                                                    deriv,
01251     #     #                                                                    sds)
01252     #     # file.write(full_line)
01253     #
01254     #     # check_end = time.perf_counter()
01255     #
01256     # print('We are finally done with search.')
```

```
01257     # print('Current queue size: ', len(open_queue))
01258     # print('Current visited_queue queue: ', len(visited_queue))
01259     # # dump_the_queue('work_queue.txt', open_queue, visited_queue, init_rmsd, tol_error, skipped_counter)
01260     # # dump_the_queue('visited_queue.txt', visited_queue, visited_queue, init_rmsd, tol_error, skipped_counter)
```

4.23 gmx_wrappers.py File Reference

Namespaces

- gmx_wrappers

Functions

- str gmx_wrappers.convert_gro_to_xtc (str gro_file, str ndx_file)

  Converts .gro into .xtc format.
- NoReturn gmx_wrappers.gmx_trjconv (str f, str o, str n=None, str s=None, int b=None, int e=None, int dump=None, str fit=None, str vel=None, str pbc=None)
- NoReturn gmx_wrappers.gmx_trjcat (str f, str o, str n, bool cat=True, bool vel=False, bool sort=False, bool overwrite=True)

  'gmx trjcat' - GROMACS tool - concatenates several input trajectory files in sorted order
- NoReturn gmx_wrappers.gmx_eneconv (str f, str o)

  'gmx eneconv' - GROMACS tool - Concatenates several energy files in sorted order
- NoReturn gmx_wrappers.gmx_energy (str f, str o, bool w=None, str w_prog=None, bool fee=True, float fetemp=300)

  'gmx trjconv' - GROMACS tool - extracts energy components from an energy file
- NoReturn gmx_wrappers.gmx_mdrun (str work_dir, int seed, str new_name, int ncores=multiprocessing.cpu_count(), str thread_type='nt')

  gmx localhost version.
- NoReturn gmx_wrappers.gmx_mdrun_mpi (str work_dir, int seed, str new_name, list hostnames, list ncores=None, str thread_type='ntomp')

  gmx MPI version
- NoReturn gmx_wrappers.gmx_mdrun_mpi_with_sched (str work_dir, int seed, str new_name, list ncores=None, int ntomp=1)

  gmx MPI version with scheduler
- NoReturn gmx_wrappers.gmx_grompp (str work_dir, int seed, str top_file, str prev_name)

  gmx grompp (the gromacs preprocessor) reads a molecular topology file, checks the validity of the file, expands the topology from a molecular description to an atomic description.

Variables

- gmx_wrappers.my_env = os.environ.copy()

4.24 gmx_wrappers.py

```
00001 """
00002 This file contains GROMACS wrappers.
00003     :platform: linux
00004
00005 .. moduleauthor:: Ivan Syzonenko <is2k@mtmail.mtsu.edu>
00006 """
00007 __license__ = "MIT"
00008 __docformat__ = 'reStructuredText'
00009
00010 import subprocess
00011 import multiprocessing
00012 import os
00013 from typing import NoReturn, Mapping, Sequence, List, Set
00014
00015 my_env = os.environ.copy()
00016 my_env["GMX_MAXBACKUP"] = "-1"
00017 my_env["GMX_NO_QUOTES"] = ""
00018 os.environ.update(my_env)
00019
00020
00021 def convert_gro_to_xtc(gro_file: str, ndx_file: str) -> str:
00022     """Converts .gro into .xtc format. Just a wrapper around trjconv.
00023
00024     Args:
00025         :param str gro_file: input filename
00026         :param str ndx_file: index file, shows which atoms to store in .xtc
00027
00028     Returns:
00029         :return: .xtc filename
00030     """
00031     out_filename = gro_file[0:-3] + 'xtc'
00032     gmx_trjconv(f=gro_file, o=out_filename, n=ndx_file)
00033     return out_filename
```

```
00034
00035
00036 def gmx_trjconv(f: str, o: str, n: str = None, s: str = None, b: int = None, e: int = None,
00037                 dump: int = None, fit: str = None, vel: str = None, pbc: str = None) -> NoReturn:
00038     """'gmx trjconv' - GROMACS tool - converts trajectory files in many ways
00039
00040     Converts between various formats. In our case from .gro to .xtc or
00041      from .gro to .gro with specific index file to filter protein only or it's specific parts.
00042
00043     Args:
00044         :param str f: Input trajectory: xtc trr cpt gro g96 pdb tng
00045         :param str o: Output trajectory: xtc trr gro g96 pdb tng
00046         :param str n: Index file
00047         :param str s: Structure+mass(db): tpr gro g96 pdb brk ent
00048         :param int b: Time of first frame to read from trajectory (default unit ps)
00049         :param int e: Time of last frame to read from trajectory (default unit ps)
00050         :param int dump: Dump frame nearest specified time (ps)
00051         :param str fit: Fit molecule to ref structure in the structure
00052         file: none, rot+trans, rotxy+transxy, translation, transxy, progressive
00053         :param str vel: Read and write velocities if possible
00054         :param str pbc: PBC treatment (see help text for full description):
00055         none, mol, res, atom, nojump, cluster, whole
00056
00057     Returns:
00058     Generates one output file passed with -o parameter.
00059     """
00060     if not (f and o):
00061         raise Exception('Missing in/out arguments.')
00062     command_trjconv = 'gmx trjconv -f {:s} -o {:s} '.format(f, o)
00063     if n:
00064         command_trjconv += '-n {} '.format(n)
00065     if s:
00066         command_trjconv += '-s {} '.format(s)
00067     if b:
00068         command_trjconv += '-b {} '.format(b)
00069     if e:
00070         command_trjconv += '-e {} '.format(e)
00071     if dump:
00072         command_trjconv += '-dump {} '.format(dump)
00073     # if vel:
00074     #     command_trjconv += '-vel '
00075     # else:
00076     #     command_trjconv += '-novel '
00077     if fit:
00078         if fit not in ['none', 'rot+trans', 'rotxy+transxy', 'translation', 'transxy', 'progressive']:
00079             raise Exception('Wrong fit parameter in gmx_trjconv.')
00080         command_trjconv += '-fit {} '.format(fit)
00081     if pbc:
00082         if pbc not in ['none', 'mol', 'res', 'atom', 'nojump', 'cluster', 'whole']:
00083             raise Exception('Wrong pbc parameter in gmx_trjconv.')
00084         command_trjconv += '-pbc {} '.format(pbc)
00085
00086     # command_trjconv = os.path.expandvars(command_trjconv)
00087     # print(command_trjconv)
00088     proc_obj = subprocess.Popen(command_trjconv, stdout=-1, shell=True, cwd='.', stderr=-1, env=my_env)
00089     output, error = proc_obj.communicate()
00090     error = error.decode("utf-8")
00091     if 'error' in error.lower():
00092         print(error)
00093     # print(output.decode("utf-8"))
00094     # print(error)
00095
00096
00097 def gmx_trjcat(f: str, o: str, n: str, cat: bool = True, vel: bool = False, sort: bool = False, overwrite: bool = True) -> NoReturn:
00098     """'gmx trjcat' - GROMACS tool - concatenates several input trajectory files in sorted order
00099
00100     Outputs one .xtc file that contains all frames (99% frames are NOT sorted, since trajectories have the same time)
00101
00102     Args:
00103         :param str f: Input trajectory: xtc trr cpt gro g96 pdb tng
00104         :param str o: Output trajectory: xtc trr gro g96 pdb tng
00105         :param str n: Index file
00106         :param bool cat: Do not discard double time frames
00107         :param bool vel: Read and write velocities if possible
00108         :param bool sort: Sort trajectory files (not frames)
00109         :param bool overwrite: Overwrite overlapping frames during appending
00110
00111     Returns:
00112     Generates one output file passed with -o parameter.
00113     """
00114     command_trjcat = 'gmx trjcat -keeplast '
```

```
00115     if not (f and o):
00116         raise Exception('Missing in/out arguments.')
00117     command_trjcat += '-o {:s} '.format(o)
00118     if isinstance(f, list):
00119         command_trjcat += '-f ' + ' '.join(f) + ' '
00120     else:
00121         command_trjcat += '-f {:s} '.format(f)
00122     if n:
00123         command_trjcat += '-n {} '.format(n)
00124     if cat:
00125         command_trjcat += '-cat '
00126     else:
00127         command_trjcat += '-nocat '
00128     # if vel:
00129     #     command_trjcat += '-vel '
00130     # else:
00131     #     command_trjcat += '-novel '
00132     if sort:
00133         command_trjcat += '-sort '
00134     else:
00135         command_trjcat += '-nosort '
00136     if overwrite:
00137         command_trjcat += '-overwrite '
00138 
00139     command_trjcat = os.path.expandvars(command_trjcat)
00140     proc_obj = subprocess.Popen(command_trjcat, stdout=-1, shell=True, cwd='.', stderr=-1, env=my_env)
00141     output, error = proc_obj.communicate()
00142     error = error.decode("utf-8")
00143     if 'error' in error.lower():
00144         print(error)
00145 
00146 
00147 def gmx_eneconv(f: str, o: str) -> NoReturn:
00148     """'gmx eneconv' - GROMACS tool - Concatenates several energy files in sorted order
00149 
00150     Stores converted energy files. Not used by main algorithm, but during the postprocessing.
00151 
00152     Args:
00153         :param str f: Input trajectory: xtc trr cpt gro g96 pdb tng
00154         :param str o: Output trajectory: xtc trr gro g96 pdb tng
00155 
00156     Returns:
00157     Generates one output energy file passed with -o parameter.
00158     """
00159     command_eneconv = 'gmx eneconv '
00160     if not (f and o):
00161         raise Exception('Missing in/out arguments.')
00162     command_eneconv += '-o {:s} '.format(o)
00163     if isinstance(f, list):
00164         command_eneconv += '-f ' + ' '.join(f) + ' -nosort -settime '
00165         #command_eneconv += '-f ' + ' '.join(f) + ' -settime '
00166         # command_eneconv = 'echo -e "{}" | '.format('\n'.join([str(i) for i in range(0, len(f) * 20, 20)])) + command_eneconv
00167         command_eneconv = 'echo -e "{}" | '.format('\n'.join(['c']*(len(f)+1))) + command_eneconv
00168     else:
00169         command_eneconv += '-f {:s} '.format(f)
00170 
00171     command_eneconv = os.path.expandvars(command_eneconv)
00172     proc_obj = subprocess.Popen(command_eneconv, stdout=-1, shell=True, cwd='.', stderr=-1, env=my_env)
00173     output, error = proc_obj.communicate()
00174     error = error.decode("utf-8")
00175     if 'error' in error.lower():
00176         print(error)
00177 
00178 
00179 def gmx_energy(f: str, o: str, w: bool = None, w_prog: str = None, fee: bool = True, fetemp: float = 300) -> NoReturn:
00180     """'gmx trjconv' - GROMACS tool - extracts energy components from an energy file
00181 
00182     Args:
00183         :param str f: .edr Energy file
00184         :param str o: energy.xvg - xvgr/xmgr file
00185         :param str w: View output .xvg, .xpm, .eps and .pdb files
00186         :param str w_prog: viewing programm
00187         :param bool fee: Do a free energy estimate
00188         :param float fetemp: Reference temperature for free energy calculation
00189 
00190     Returns:
00191     Generates one output .xvg file passed with -o parameter.
00192     """
00193     command_energy = 'gmx energy '
00194     command_energy += ' -f ' + f
00195     command_energy += ' -o ' + o
```

```
00196     if w:
00197         command_energy += '-w {} {} '.format(w, w_prog)
00198     if fee:
00199         command_energy += ' -fee '
00200     if fetemp:
00201         command_energy += ' -fetemp {}'.format(fetemp)
00202     command_energy = 'echo -e "10" | ' + command_energy
00203     command_energy = os.path.expandvars(command_energy)
00204     proc_obj = subprocess.Popen(command_energy, stdout=-1, shell=True, cwd='.', stderr=-1, env=my_env)
00205     output, error = proc_obj.communicate()
00206     error = error.decode("utf-8")
00207     if 'error' in error.lower():
00208         print(error)
00209 
00210 
00211 def gmx_mdrun(work_dir: str, seed: int, new_name: str, ncores: int = multiprocessing.cpu_count(), thread_type: str = 'nt') -> NoReturn:
00212     """gmx mdrun - localhost version.
00213 
00214     Args:
00215         :param str work_dir: path to work directory, where all seed directories reside
00216         :param int seed: seed value used in the MD simulation
00217         :param str new_name: output name for a final state
00218         :param int ncores: number of cores to use in the current simulation
00219         :param str thread_type: thread type: MPI ? OMP ? TMPI ?
00220 
00221     Returns:
00222     Starts a shell in a separate process and runs mdrun there.
00223     """
00224     if thread_type not in ['nt', 'ntomp']:  # 'ntmpi' is prohibited when gromacs compiled without mpi support
00225         raise Exception('Wrong thread type passed in gmx_mdrun')
00226     ncores = ncores if ncores > 0 else 1
00227 
00228     command_run_md = "gmx mdrun -deffnm md -{} {} -c {} -reprod".format(thread_type, ncores, new_name)
00229     # command_run_md = "gmx mdrun -deffnm md -{} {} -c {} -pin on -reprod".format(thread_type, ncores, new_name)
00230     proc_obj = subprocess.Popen(command_run_md, stdout=-1, shell=True, cwd='{}/{}/'.format(work_dir, seed), stderr=-1, env=my_env)
00231     output, error = proc_obj.communicate()
00232     error = error.decode("utf-8")
00233     output = output.decode("utf-8")
00234     # with open(str(os.getpid())+'_err.log', 'a') as log_out:
00235     #     log_out.write(error)
00236     # with open(str(os.getpid())+'_out.log', 'a') as log_out:
00237     #     log_out.write(output.decode("utf-8"))
00238 
00239     if 'error' in error.lower():
00240         print(error)
00241 
00242 
00243 def gmx_mdrun_mpi(work_dir: str, seed: int, new_name: str, hostnames: list, ncores: list = None, thread_type: str = 'ntomp') -> NoReturn:
00244     """gmx mdrun - MPI version
00245 
00246     Args:
00247         :param str work_dir: path to work directory, where all seed directories reside
00248         :param int seed: seed value used in the MD simulation
00249         :param str new_name: output name for a final state
00250         :param list hostnames: must be a list
00251         :param list ncores: number of cores to use in the current simulation
00252         :param str thread_type: type of the thread, OMP ? MPI ?
00253 
00254     Returns:
00255     Starts a shell in a separate process and runs mdrun there.
00256     This version uses MPI to run on a separate host
00257     """
00258     if thread_type not in ['ntmpi', 'ntomp']:  # 'nt' is prohibited when gromacs compiled with mpi support
00259         raise Exception('Wrong thread type passed in gmx_mdrun')
00260     one_host_only_mpi = True
00261     if one_host_only_mpi:
00262             command_run_md = "mpirun -host {} -np 1 mdrun -deffnm md -c {} -nt 32 -ntomp 2 -pin on -reprod \
00263                              ".format(','.join(hostnames), new_name, int(ncores))
00264     else:
00265         if ncores:
00266             command_run_md = "mpirun -host {0} -np {1} mdrun -deffnm md -c {2} -ntomp 2 -nt {1} -pin on -reprod \
00267                              ".format(','.join(hostnames), min(1, int(ncores)), new_name)
00268             # command_run_md = "mpirun -host {} -np {} mdrun_mpi -deffnm md -c {} -ntomp 2 -pin on -reprod \
00269             #                  ".format(','.join(hostnames), min(1, int(ncores)//2), new_name)
00270         else:
00271             command_run_md = "mpirun -hosts {} gmx mdrun -deffnm md -c {} -ntomp 2 -pin on -reprod".format(','.join(hostnames), new_name)
00272     proc_obj = subprocess.Popen(command_run_md, stdout=-1, shell=True, cwd='{}/{}/'.format(work_dir, seed), stderr=-1, env=my_env)
00273     output, error = proc_obj.communicate()
00274     error = error.decode("utf-8")
00275     output = output.decode("utf-8")
00276     # with open(str(os.getpid())+'_err.log', 'a') as log_out:
```

```
00277     #     log_out.write(error)
00278     # with open(str(os.getpid())+'_out.log', 'a') as log_out:
00279     #     log_out.write(output.decode("utf-8"))
00280 
00281     if 'error' in error.lower():
00282         print(error)
00283 
00284 
00285 def gmx_mdrun_mpi_with_sched(work_dir: str, seed: int, new_name: str, ncores: list = None, ntomp: int = 1) -> NoReturn:
00286     """gmx mdrun - MPI version with scheduler
00287 
00288     Args:
00289         :param str work_dir: path to work directory, where all seed directories reside
00290         :param int seed: seed value used in the MD simulation
00291         :param str new_name: output name for a final state
00292         :param list ncores: number of cores to use in the current simulation
00293         :param int ntomp: number of OMP threads
00294 
00295     Returns:
00296     Starts a shell in a separate process and runs mdrun there.
00297     This version uses MPI but does not specify the host, it should be done through the scheduler.
00298     Do not use this version if you know the exact host names - then you have more control and potentially less overhead.
00299     """
00300     if ncores % ntomp != 0 or (ntomp > ncores):
00301         raise Exception('Not possible to divide OMP threads evenly among the specified number of cores.\nCores: {}\tOMP threads:
      {}\n'.format(ncores, ntomp))
00302 
00303     if ntomp == ncores:
00304         command_run_md = "mpirun -np {0} mdrun -deffnm md -c {1} -pin on -reprod".format(ncores, new_name)
00305     else:
00306         command_run_md = "mpirun -np {0} mdrun -deffnm md -c {1} -ntomp {2} -pin on -reprod".format(ncores, new_name, ntomp)
00307 
00308     proc_obj = subprocess.Popen(command_run_md, stdout=-1, shell=True, cwd='{}/{}/'.format(work_dir, seed), stderr=-1, env=my_env)
00309     output, error = proc_obj.communicate()
00310     error = error.decode("utf-8")
00311     output = output.decode("utf-8")
00312     # with open(str(os.getpid())+'_err.log', 'a') as log_out:
00313     #     log_out.write(error)
00314     # with open(str(os.getpid())+'_out.log', 'a') as log_out:
00315     #     log_out.write(output.decode("utf-8"))
00316 
00317     if 'error' in error.lower():
00318         print(error)
00319 
00320 
00321 def gmx_grompp(work_dir: str, seed: int, top_file: str, prev_name: str) -> NoReturn:
00322     """gmx grompp (the gromacs preprocessor) reads a molecular topology file, checks the validity of the file,
00323      expands the topology from a molecular description to an atomic description.
00324 
00325     Args::
00326 
00327         :param str work_dir: path to work directory, where all seed directories reside
00328         :param int seed: seed value used in the MD simulation
00329         :param str top_file: .top - topology of the conformation
00330         :param str prev_name: previous simulation digest. Used as starting point.
00331 
00332     Returns
00333 
00334     Creates .tpr - binary config file.
00335     """
00336     command_prep_run = "gmx grompp -f md.mdp -c {}.gro -p {} -o md.tpr".format(prev_name, top_file)
00337     proc_obj = subprocess.Popen(command_prep_run, stdout=-1, shell=True, cwd=os.path.join(work_dir, str(seed)), stderr=-1, env=my_env)
00338     output, error = proc_obj.communicate()
00339     error = error.decode("utf-8")
00340     # with open(str(os.getpid())+'_err.log', 'a') as log_out:
00341     #     log_out.write(error)
00342     # with open(str(os.getpid())+'_out.log', 'a') as log_out:
00343     #     log_out.write(output.decode("utf-8"))
00344 
00345     if 'error' in error.lower():
00346         print(error)
```

4.25 helper_funcs.py File Reference

Namespaces

- helper_funcs

Functions

- str helper_funcs.get_digest (str in_str)

Computes digest of the input string.

- list helper_funcs.create_core_mapping (int ncores=mp.cpu_count(), int nseeds=1)

  Tries to map cores evenly among tasks.

- list helper_funcs.get_previous_runs_info (str check_dir)

  Scans direcotory for prior results and outputs the list of filenames.

- def helper_funcs.check_precomputed_noize (str an_file, list metr_order)

  Checks whether file with precomputed ambient noise exists.

- NoReturn helper_funcs.make_a_step (str work_dir, int cur_seed, dict seed_dirs, str top_file, str ndx_file, str seed_digest_filename, str old_name_digest, str past_dir, int ncores=1)

  Version for the case when you use one machine, for example, local computer or one remote server.

- NoReturn helper_funcs.make_a_step2 (str work_dir, int cur_seed, dict seed_dirs, str top_file, str ndx_file, str seed_digest_filename, str old_name_digest, str past_dir, str hostname, int ncores)

  Version for the case when you use cluster and have hostnames.

- NoReturn helper_funcs.make_a_step3 (str work_dir, int cur_seed, dict seed_dirs, str top_file, str ndx_file, str seed_digest_filename, str old_name_digest, str past_dir, int ncores, int ntomp=1)

  Version for the case when you use scheduler and have many cores, but no hostnames.

- dict helper_funcs.get_seed_dirs (str work_dir, list list_with_cur_seeds, int simulation_temp, dict sd=None)

  Create directories with unique names for simulation with specified seeds and puts .mdp, config files for the MD simulation.

- NoReturn helper_funcs.rm_seed_dirs (dict seed_dirs)

  Removes seed directory and all it's content.

- list helper_funcs.get_new_seeds (list old_seeds, int seed_num=4)

  Returns next seed sequence.

- NoReturn helper_funcs.trjcat_many (list hashed_names, str past_dir, str out_name)

  Concatenates many trajectories into one file.

- NoReturn helper_funcs.general_bak (str fname, tuple state)

  Stores variables in the picke with the specific name.

- tuple helper_funcs.general_rec (str fname)

  Reads pickle content from the file.

- NoReturn helper_funcs.main_state_backup (tuple state)

  Just a wrapper around the general_bak.

- NoReturn helper_funcs.supp_state_backup (tuple state)

  Just a wrapper around the general_bak.

- tuple helper_funcs.main_state_recover ()

  Just a wrapper around the general_rec.

- tuple helper_funcs.supp_state_recover ()

  Just a wrapper around the general_rec.

4.26 helper_funcs.py

```
00001 """This file contains various wrappers and functions that ease the code digestion and programming in general.
00002
00003     :platform: linux
00004
00005 .. moduleauthor:: Ivan Syzonenko <is2k@mtmail.mtsu.edu>
00006 """
00007 __license__ = "MIT"
00008 __docformat__ = 'reStructuredText'
00009
00010 import os
00011 import multiprocessing as mp
00012 import hashlib
00013 from shutil import copy2 as cp2
00014 import heapq
00015 import shutil
00016 import pickle
00017
00018 from typing import NoReturn
00019
00020 from gen_mdp import get_mdp
00021 from gmx_wrappers import gmx_grompp, gmx_trjconv, gmx_trjcat, gmx_mdrun, gmx_mdrun_mpi, gmx_mdrun_mpi_with_sched
00022
00023
00024 def get_digest(in_str: str) -> str:
00025     """Computes digest of the input string.
```

```
00026
00027     Args:
00028         :param str in_str: typically list of seeds concatenated with _. like s_0_1_5
00029
00030     Returns:
00031     :return: blake2 hash of the in_str. We use short version,
00032      but you can use full version - slightly slower, but less chances of name collision.
00033     :rtype: str
00034     """
00035     # return hashlib.md5(in_str.encode()).hexdigest()
00036     # if you have python older than 3.6 - use md5 or update python
00037     return hashlib.blake2s(in_str.encode()).hexdigest()
00038
00039
00040 def create_core_mapping(ncores: int = mp.cpu_count(), nseeds: int = 1) -> list:
00041     """Tries to map cores evenly among tasks.
00042
00043     Args:
00044         :param int ncores: number of cores available
00045         :param int nseeds: number of seeds used in current run
00046
00047     Returns:
00048         :return: list of tuples, each tuple consist of (cores number, task identifier)
00049         :rtype: list
00050     """
00051     ncores = ncores if ncores > 0 else 1
00052     nseeds = nseeds if nseeds > 0 else 1
00053     print('I will use {} cores for {} seeds'.format(ncores, nseeds))
00054
00055     even = ncores // nseeds
00056     remainder = ncores % nseeds
00057
00058     sched_arr = list()
00059     if even:
00060         cur_sched = [(even+1, i) if i < remainder else (even, i) for i in range(nseeds)]
00061         sched_arr.append(cur_sched)
00062     else:
00063         seeds_range_iter = iter(range(nseeds))
00064         tot_batches = nseeds//ncores
00065         remainder = nseeds-tot_batches*ncores
00066         tot_batches = tot_batches if not remainder else tot_batches+1  # if we can't divide tasks evenly, we need one more batch
00067         for i in range(tot_batches):
00068             if i < tot_batches-1:
00069                 cur_sched = [(1, 0)]*ncores
00070             else:
00071                 cur_sched = [(1, 0) if i < remainder else (0, 0) for i in range(ncores)]
00072                 free_cores = ncores - sum(i for i, j in cur_sched)
00073                 if free_cores:
00074                     cur_sched = [(j[0]+1, 0) if i < free_cores else (j[0], 0) for i, j in enumerate(cur_sched)]
00075             sched_arr.append(cur_sched)
00076         for i, cur_sched in enumerate(sched_arr):
00077             for j, cornum_seed in enumerate(cur_sched):
00078                 if cornum_seed[0]:
00079                     cur_seed = next(seeds_range_iter)
00080                     sched_arr[i][j] = (cornum_seed[0], cur_seed)
00081                     print('Seed {} will be run on {} cores.'.format(cur_seed, cornum_seed[0]))
00082
00083     return sched_arr
00084
00085
00086 def get_previous_runs_info(check_dir: str) -> list:
00087     """Scans direcotory for prior results and outputs the list of filenames.
00088
00089     Args:
00090         :param str check_dir:  directory to scan for prior trajectories
00091
00092     Returns:
00093         :return: list of filenames .xtc or .gro
00094         :rtype: list
00095     """
00096     # filenames_found = os.walk(check_dir).__next__()[2]
00097     filenames_found = [f.split("/")[-1] for f in os.listdir(check_dir)]
00098     # filenames_found = [f.path.split("/")[-1] for f in os.scandir(check_dir)]
00099     filenames_found_important = [f for f in filenames_found if f.split('.')[1] in ['xtc', 'gro']]
00100     del filenames_found
00101     print('Found files: {} with .gro and .xtc'.format(len(filenames_found_important)))
00102     return filenames_found_important
00103
00104
00105 def check_precomputed_noize(an_file: str, metr_order: list):
00106     """Checks whether file with precomputed ambient noise exists.
```

```
00107
00108     Tries to read correct number of metrics, in case of error throws and exception
00109     Otherwise returns dict{metric_name: noise_value}
00110
00111     Args:
00112         :param str an_file: ambient noise filename to check
00113         :param list metr_order: order of metric names (should be correct sequence)
00114
00115     Returns:
00116         :return: dict{metric_name: noise_value}
00117         :rtype: dict or None
00118     """
00119     # TODO: rewrite function to save noise and metric name, so you do not read the wrong sequence (add a check)
00120     if an_file in os.walk(".").__next__()[2]:
00121         print(an_file, ' was found. Reading... ')
00122         with open(an_file, 'r') as f:
00123             noize_arr = f.readlines()
00124         try:
00125             res_arr = [float(res.strip()) for res in noize_arr]
00126             err_node = dict()
00127             for i in range(len(res_arr)):
00128                 err_node[metr_order[i]] = res_arr[i]
00129         except Exception as e:
00130             print(e)
00131             return None
00132         return err_node
00133     return None
00134
00135
00136 def make_a_step(work_dir: str, cur_seed: int, seed_dirs: dict, top_file: str, ndx_file: str, seed_digest_filename: str,
00137                 old_name_digest: str, past_dir: str, ncores: int = 1) -> NoReturn:
00138     """Version for the case when you use one machine, for example, local computer or one remote server.
00139
00140     Generates the actual MD simulation by first - setting the simulation with grommp,
00141      then using several mdruns, and finally conctatenating the result into the one file.
00142
00143     Args:
00144         :param str work_dir: path to the directory where seed dirs reside
00145         :param int cur_seed: current seed value used for MD production
00146         :param dict seed_dirs: dict which contains physical path to
00147          the directory where simulation with particular seed is performed
00148         :param str top_file: .top - topology of the current conformation
00149         :param str ndx_file: .ndx - index of the protein atoms of the current conformation
00150         :param str seed_digest_filename: digest for a current MD simulation, used to store files in the past
00151         :param str old_name_digest: digest for a prior MD simulation
00152         :param str past_dir: path to the directory with prior computations
00153         :param int ncores: number of cores to use for this task
00154     """
00155     # global extra_past
00156     old_name = os.path.join(past_dir, old_name_digest)
00157     if not os.path.exists(old_name+'.gro'):
00158         # old_name = os.path.join(extra_past, old_name_digest)
00159         # if not os.path.exists(old_name + '.gro'):
00160         raise Exception("make_a_step: did not find {} in {} ".format(old_name_digest, past_dir))
00161     gmx_grompp(work_dir, cur_seed, top_file, old_name)
00162     new_name = os.path.join(past_dir, seed_digest_filename)
00163     gmx_mdrun(work_dir, cur_seed, new_name + '.gro', ncores)
00164     gmx_trjconv(f=os.path.join(seed_dirs[cur_seed], 'md.xtc'), o='{}.xtc'.format(new_name),
00165                 n=ndx_file, s=os.path.join(seed_dirs[cur_seed], 'md.tpr'), pbc='mol', b=1)
00166     try:
00167         cp2(os.path.join(seed_dirs[cur_seed], 'md.edr'), '{}.edr'.format(new_name))
00168     except:
00169         print('Error when tried to copy energy file. Maybe you do not produce them ? Then comment this line.')
00170     os.remove(os.path.join(seed_dirs[cur_seed], 'md.xtc'))
00171
00172
00173 def make_a_step2(work_dir: str, cur_seed: int, seed_dirs: dict, top_file: str, ndx_file: str, seed_digest_filename: str,
00174                  old_name_digest: str, past_dir: str, hostname: str, ncores: int) -> NoReturn:
00175     """Version for the case when you use cluster and have hostnames.
00176
00177     Generates the actual MD simulation by first - setting the simulation with grommp,
00178      then using several mdruns, and finally conctatenating the result into the one file.
00179
00180     Args:
00181         :param str work_dir: path to the directory where seed dirs reside
00182         :param int cur_seed: current seed value used for MD production
00183         :param dict seed_dirs: dict which contains physical path to the directory
00184          where simulation with particular seed is performed
00185         :param str top_file: .top - topology of the current conformation
00186         :param str ndx_file: .ndx - index of the protein atoms of the current conformation
00187         :param str seed_digest_filename: digest for a current MD simulation, used to store files in the past
```

```
00188         :param str old_name_digest: digest for a prior MD simulation
00189         :param str past_dir: path to the directory with prior computations
00190         :param str hostname: hostname to use for MD simulation
00191         :param intncores: number of cores to use for this task
00192     """
00193     # global extra_past
00194     old_name = os.path.join(past_dir, old_name_digest)
00195     if not os.path.exists(old_name + '.gro'):
00196         # old_name = os.path.join(extra_past, old_name_digest)
00197         # if not os.path.exists(old_name + '.gro'):
00198         raise Exception("make_a_step2: did not find {} in {}".format(old_name_digest, past_dir))
00199     gmx_grompp(work_dir, cur_seed, top_file, old_name)
00200     new_name = os.path.join(past_dir, seed_digest_filename)
00201     gmx_mdrun_mpi(work_dir, cur_seed, new_name + '.gro', hostname, ncores)
00202     gmx_trjconv(f=os.path.join(seed_dirs[cur_seed], 'md.xtc'), o='{}.xtc'.format(new_name),
00203                 n=ndx_file, s=os.path.join(seed_dirs[cur_seed], 'md.tpr'), pbc='mol', b=1)
00204     try:
00205         cp2(os.path.join(seed_dirs[cur_seed], 'md.edr'), '{}.edr'.format(new_name))
00206     except:
00207         print('Error when tried to copy energy file. Maybe you do not produce them ? Then comment this line.')
00208     os.remove(os.path.join(seed_dirs[cur_seed], 'md.xtc'))
00209
00210
00211 def make_a_step3(work_dir: str, cur_seed: int, seed_dirs: dict, top_file: str, ndx_file: str, seed_digest_filename: str,
00212                  old_name_digest: str, past_dir: str, ncores: int, ntomp: int = 1) -> NoReturn:
00213     """Version for the case when you use scheduler and have many cores, but no hostnames.
00214
00215     Generates the actual MD simulation by first - setting the simulation with grommp,
00216      then using several mdruns, and finally conctatenating the result into the one file.
00217
00218     Args:
00219         :param str work_dir:  path to the directory where seed dirs reside
00220         :param int cur_seed: current seed value used for MD production
00221         :param dict seed_dirs: dict which contains physical path to the directory where simulation with particular seed is performed
00222         :param str top_file: .top - topology of the current conformation
00223         :param str ndx_file: .ndx - index of the protein atoms of the current conformation
00224         :param str seed_digest_filename: digest for a current MD simulation, used to store files in the past
00225         :param str old_name_digest: digest for a prior MD simulation
00226         :param str past_dir: path to the directory with prior computations
00227         :param int ncores: number of cores to use for this task
00228         :param int ntomp: number of OMP threads to use during the simulation
00229     """
00230     # global extra_past
00231     old_name = os.path.join(past_dir, old_name_digest)
00232     if not os.path.exists(old_name +'.gro'):
00233         # old_name = os.path.join(extra_past, old_name_digest)
00234         # if not os.path.exists(old_name + '.gro'):
00235         raise Exception("make_a_step3: did not find {} in {}".format(old_name_digest, past_dir))
00236     gmx_grompp(work_dir, cur_seed, top_file, old_name)
00237     new_name = os.path.join(past_dir, seed_digest_filename)
00238     # gmx_mdrun_mpi(work_dir, cur_seed, new_name + '.gro', hostname, ncores)
00239     gmx_mdrun_mpi_with_sched(work_dir, cur_seed, new_name + '.gro', ncores, ntomp)
00240     gmx_trjconv(f=os.path.join(seed_dirs[cur_seed], 'md.xtc'), o='{}.xtc'.format(new_name),
00241                 n=ndx_file, s=os.path.join(seed_dirs[cur_seed], 'md.tpr'), pbc='mol', b=1)
00242     try:
00243         cp2(os.path.join(seed_dirs[cur_seed], 'md.edr'), '{}.edr'.format(new_name))
00244     except:
00245         print('Error when tried to copy energy file. Maybe you do not produce them ? Then comment this line.')
00246     os.remove(os.path.join(seed_dirs[cur_seed], 'md.xtc'))
00247
00248
00249 def get_seed_dirs(work_dir: str, list_with_cur_seeds: list, simulation_temp: int, sd: dict = None) -> dict:
00250     """Create directories with unique names for simulation with specified seeds and puts .mdp, config files for the MD simulation.
00251
00252     Args:
00253         :param str work_dir: path to work directory, where all seed directories reside
00254         :param list list_with_cur_seeds: list of seed currently used
00255         :param int simulation_temp: simulation temperature used to generate proper .mdp file
00256         :param dict sd: Not used anymore, but left for sime time as deprecated. sd - previous seed deers
00257
00258     Returns:
00259         :return: dictionary with seed dir paths
00260         :rtype dict
00261     """
00262     if not sd:
00263         sd = dict()
00264     for seed in list_with_cur_seeds:
00265         seed_dir = os.path.join(work_dir, str(seed))
00266         sd[seed] = seed_dir
00267         if not os.path.exists(seed_dir):
00268             os.makedirs(seed_dir)
```

```
00269         with open(os.path.join(sd[seed], 'md.mdp'), 'w') as f:
00270             f.write(get_mdp(seed, simulation_temp))
00271     return sd
00272 
00273 
00274 def rm_seed_dirs(seed_dirs: dict) -> NoReturn:
00275     """Removes seed directory and all it's content
00276 
00277     Args:
00278         :param dict seed_dirs: dict which contains physical path to the directory where simulation with particular seed is performed
00279 
00280     Removes old working directories to save disc space.
00281     """
00282     for seed_dir in seed_dirs.values():
00283         if os.path.exists(seed_dir):
00284             shutil.rmtree(seed_dir, ignore_errors=True)
00285 
00286 
00287 def get_new_seeds(old_seeds: list, seed_num: int = 4) -> list:
00288     """Returns next seed sequence.
00289 
00290     Args:
00291         :param list old_seeds: list of previous seeds
00292         :param int seed_num: number of unique seeds in the current run
00293 
00294     Returns:
00295         :return: list of new seeds
00296         :rtype list
00297     """
00298     max_seeds = 64000  # change this if you want more exploration
00299     if min(old_seeds) + seed_num > max_seeds:
00300         return None
00301     return [seed + seed_num for seed in old_seeds]
00302 
00303 
00304 def trjcat_many(hashed_names: list, past_dir: str, out_name: str) -> NoReturn:
00305     """Concatenates many trajectories into one file.
00306 
00307     Args:
00308         :param list hashed_names: .xtc filenames to concatenate
00309         :param str past_dir: path to the directory with prior computations
00310         :param str out_name: single output filename
00311 
00312     Returns:
00313     Generates one file with many frames.
00314     """
00315     wave = 100
00316     tot_chunks = int((len(hashed_names) + 1) / wave)
00317     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00318     gmx_trjcat(f=[os.path.join(past_dir, hashed_name) + '.xtc' for hashed_name in hashed_names[:wave]],
00319                o='./combinded_traj.xtc', n='./prot_dir/prot.ndx', cat=True, vel=False, sort=False, overwrite=True)
00320     for i in range(wave, len(hashed_names), wave):
00321         os.rename('./combinded_traj.xtc', './combinded_traj_prev.xtc')
00322         gmx_trjcat(f=[" ./combinded_traj_prev.xtc "] + [os.path.join(past_dir, hashed_name) + '.xtc' for hashed_name in
    hashed_names[i:i+wave]],
00323                    o='./combinded_traj.xtc',
00324                    n='./prot_dir/prot.ndx', cat=True, vel=False, sort=False, overwrite=True)
00325         if int(i / wave) % 10 == 0:
00326             print('{}/{} ({:.1f}%)'.format(int(i / wave), tot_chunks, 100 * int(i / wave) / tot_chunks))
00327     if os.path.exists('./combinded_traj_prev.xtc'):
00328         os.remove('./combinded_traj_prev.xtc')
00329     os.rename('./combinded_traj.xtc', out_name)
00330 
00331 
00332 def general_bak(fname: str, state: tuple) -> NoReturn:
00333     """Stores variables in the picke with the specific name
00334 
00335     Args:
00336         :param str fname: filename for the pickle
00337         :param tuple state: variables to store
00338 
00339     Returns:
00340     Generates a file with pickled data.
00341     """
00342     if os.path.exists(os.path.join(os.getcwd(), fname)):
00343         try:
00344             os.rename(os.path.join(os.getcwd(), fname), os.path.join(os.getcwd(), fname + '_prev'))
00345         except Exception as e:
00346             # print(e)
00347             os.remove(os.path.join(os.getcwd(), fname))
00348             os.rename(os.path.join(os.getcwd(), fname), os.path.join(os.getcwd(), fname + '_prev'))
```

```
00349
00350     with open(fname, 'wb') as f:
00351         pickle.dump(state, f)
00352
00353
00354 def general_rec(fname: str) -> tuple:
00355     """Reads pickle content from the file.
00356
00357     Args:
00358         :param str fname: pickle filename
00359
00360     Returns:
00361         :return: state from the pickle
00362         :rtype: tuple
00363     """
00364     with open(fname, 'rb') as f:
00365         state = pickle.load(f)
00366     return state
00367
00368
00369 def main_state_backup(state: tuple) -> NoReturn:
00370     """Just a wrapper around the general_bak
00371
00372     Args:
00373         :param tuple state: (visited_queue, open_queue, main_dict)
00374     """
00375     general_bak('small.pickle', state)
00376
00377
00378 def supp_state_backup(state: tuple) -> NoReturn:
00379     """Just a wrapper around the general_bak
00380
00381     Args:
00382         :param tuple state: (tol_error, seed_list, seed_dirs, seed_change_counter, skipped_counter, cur_metric_name,
00383                              cur_metric, counter_since_seed_changed, guiding_metric, greed_mult,
00384                             best_so_far_name, best_so_far, greed_count)
00385     """
00386     general_bak('big.pickle', state)
00387
00388
00389 def main_state_recover() -> tuple:
00390     """Just a wrapper around the general_rec
00391
00392     Returns:
00393         :return: state from the pickle
00394     """
00395     return general_rec('small.pickle')
00396
00397
00398 def supp_state_recover() -> tuple:
00399     """Just a wrapper around the general_rec
00400
00401     Returns:
00402         :return: state from the pickle
00403     """
00404     return general_rec('big.pickle')
```

4.27 main.py File Reference

Namespaces

- main

Functions

- def main.main ()

  This function is basically a launcher.

4.28 main.py

```
00001 #!/usr/bin/env python3.6
00002
00003 """
00004 This file contains various wrappers and functions that ease the code digestion and programming in general.
00005     .. module:: main
00006     :platform: linux
00007
00008 .. moduleauthor:: Ivan Syzonenko <is2k@mtmail.mtsu.edu>
00009 """
```

```
00010 __license__ = "MIT"
00011 __docformat__ = 'reStructuredText'
00012
00013 import multiprocessing
00014 import os
00015 from GMDA_main import GMDA_main
00016 from threaded_funcs import threaded_db_input, threaded_print  # ,threaded_copy, threaded_rm
00017 # from helper_funcs import get_previous_runs_info
00018
00019
00020 def main():
00021     """This function is basically a launcher
00022
00023     Parallel threads did not result in a much better performance and was masked for better times.
00024     However, if you decide to implement C++ parallel I/O - it should help.
00025     """
00026     # Compilation steps:
00027     # compile latest gcc
00028     # compile gromacs with shared libs and static libs, without mpi; install
00029     # compile mdsctk
00030     # OPTIONAL: compile gromacs with mpi/openmp if needed.
00031     tot_seeds = 4
00032     # get_db_con(tot_seeds=4)
00033
00034     past_dir = os.path.join(os.getcwd(), 'past/')
00035     #
00036     # PRINT_LOCK = Lock()
00037     # COPY_LOCK = Lock()
00038     # RM_LOCK = Lock()
00039
00040     # print_queue = queue.Queue()
00041     # printing_thread = Thread(target=threaded_print, args=(print_queue,))
00042     # printing_thread.start()
00043
00044     # db_input_queue = queue.Queue()
00045     # db_input_thread = Thread(target=threaded_db_input, args=(db_input_queue, tot_seeds,))
00046     # db_input_thread.start()
00047     # # db_input_queue.put(None)
00048     #
00049     # copy_queue = queue.Queue()
00050     # copy_thread = Thread(target=threaded_copy, args=(copy_queue,))
00051     # copy_thread.start()
00052     #
00053     # rm_queue = queue.Queue()
00054     # rm_thread = Thread(target=threaded_rm, args=(rm_queue, RM_LOCK,))
00055     # rm_thread.start()
00056
00057     # prev_runs_files = get_previous_runs_info(past_dir)
00058     prev_runs_files = None
00059
00060     print_queue = multiprocessing.JoinableQueue(102400)
00061     printing_thread = multiprocessing.Process(target=threaded_print, args=(print_queue,))
00062     printing_thread.start()
00063
00064     db_input_queue = multiprocessing.JoinableQueue(102400)
00065     db_input_thread = multiprocessing.Process(target=threaded_db_input, args=(db_input_queue, tot_seeds,))
00066     db_input_thread.start()
00067
00068     # no need in the next queues. Maybe helpful if working with /dev/shm
00069     copy_queue = None
00070     # copy_queue = multiprocessing.Queue()
00071     # copy_thread = multiprocessing.Process(target=threaded_copy, args=(copy_queue,))
00072     # copy_thread.start()
00073
00074     rm_queue = None
00075     # rm_queue = multiprocessing.JoinableQueue(3)
00076     # rm_thread = multiprocessing.Process(target=threaded_rm, args=(rm_queue,))
00077     # rm_thread.start()
00078
00079     GMDA_main(prev_runs_files, past_dir, print_queue, db_input_queue, copy_queue, rm_queue, tot_seeds)
00080
00081     printing_thread.join()
00082     db_input_thread.join()
00083     print_queue.put_nowait(None)
00084     db_input_queue.put_nowait(None)
00085     rm_queue.put_nowait(None)
00086     # print_queue.join()
00087     # db_input_queue.join()
00088     # rm_queue.join()
00089
00090
```

```
00091 if __name__ == "__main__":
00092     main()
```

4.29 make_best_trajectory_new.py File Reference

Namespaces

- make_best_trajectory_new

Functions

- def make_best_trajectory_new.main ()
- def make_best_trajectory_new.build_best_traj (str metr_name, str db_to_connect)

  Finds the lowest value of the metric and builds the trajectory that leads to this point.
- def make_best_trajectory_new.main_energy ()

4.30 make_best_trajectory_new.py

```
00001 #!/usr/bin/env python3
00002
00003 import sqlite3 as lite
00004 import os
00005 import sys
00006 from gmx_wrappers import gmx_trjcat
00007 import sqlite3 as lite
00008 import os
00009 # import matplotlib.pyplot as plt
00010 # import scipy
00011 # from scipy.optimize import curve_fit
00012 # import numpy as np
00013 # from matplotlib.ticker import NullFormatter  # useful for 'logit' scale
00014 # from matplotlib import gridspec
00015 # from PIL import Image
00016 # from matplotlib import figure
00017 # from matplotlib.figure import figaspect
00018 from gmx_wrappers import gmx_eneconv, gmx_energy
00019 from shutil import copy2
00020 import multiprocessing as mp
00021
00022
00023 def main():
00024     db_to_connect = 'results_opls_trp_300_fixed'
00025     # if len(sys.argv) < 2:
00026     #     raise Exception('Not enough arguments')
00027     # db_to_connect = sys.argv[1]
00028     # try:
00029     #     os.mkdir('best_past')
00030     # except:
00031     #     pass
00032     for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']:
00033         build_best_traj(metr, db_to_connect)
00034     # pool = mp.Pool(len(['rmsd', 'angl', 'andh', 'and', 'xor']))  # we are IO bound in graphs, no need to use exact number of CPUs
       mp.cpu_count()
00035     # results1 = pool.starmap_async(build_best_traj, [(metr, db_to_connect) for metr in ['rmsd', 'angl', 'andh', 'and', 'xor']])
00036     # results1.get()
00037     # pool.close()
00038
00039
00040
00041 def build_best_traj(metr_name: str, db_to_connect: str):
00042     """Finds the lowest value of the metric and builds the trajectory that leads to this point.
00043
00044     Once best value is found, we search for a name, parse it (name consist of prev seeds separated by _).
00045     Once we have all the preceeding seeds, we can extract their frames and join them.
00046
00047 Parameters
00048 ----------
00049         :param str metr_name:
00050         :param str db_to_connect:
00051
00052 Returns
00053 -------
00054     Generates one .xtc trajectory with frames that result in the best conformation according to the specific metric.
00055     """
00056
00057     # db_to_connect = 'results_opls_trp_300_2_fixed'
00058
00059     past_dir = './past'
00060     if not os.path.exists(db_to_connect + '.sqlite3'):
```

```
00061         raise Exception('DB not found')
00062
00063     con = lite.connect(db_to_connect + '.sqlite3', check_same_thread=False, isolation_level=None)
00064     cur = con.cursor()
00065
00066     qry = "select a.name, a.hashed_name, a.{0}_goal_dist from main_storage a \
00067            where a.{0}_goal_dist= ( select min(b.{0}_goal_dist) from main_storage b)".format(metr_name)
00068     result = cur.execute(qry)
00069     all_res = result.fetchone()
00070     print('The closest frame to goal has {} {} and name:\n{}'.format(metr_name, all_res[2], all_res[1]))
00071     name = all_res[0]
00072     spname = name.split('_')
00073     all_prev_names = ['\'{}\"'.format('_'.join(spname[:i])) for i in range(1, len(spname)+1)]
00074     long_line = ", ".join(all_prev_names)
00075
00076     qry = "select name, hashed_name from main_storage where name in ({})".format(long_line)
00077     result = cur.execute(qry)
00078     all_res = result.fetchall()
00079     con.close()
00080
00081     names, hashed_names = zip(*all_res)
00082
00083     # for file in [os.path.join(past_dir, hashed_name) for hashed_name in hashed_names]:
00084     #     copy2('{}.xtc'.format(file), './best_past/')
00085     #     try:
00086     #         copy2('{}.edr'.format(file), './best_past/')
00087     #     except:
00088     #         print('Failed to copy {}; Normal for the first frame.'.format(file))
00089
00090     wave = 100
00091     tot_chunks = int((len(hashed_names) + 1) / wave)
00092     print('Computing best trajectory for {}'.format(metr_name))
00093     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00094     if os.path.exists('./{}_combined_traj.xtc'.format(metr_name)):
00095         os.remove('./{}_combined_traj.xtc'.format(metr_name))
00096     if os.path.exists('./{}_combined_traj_prev.xtc'.format(metr_name)):
00097         os.remove('./{}_combined_traj_prev.xtc'.format(metr_name))
00098
00099     gmx_trjcat(f=[os.path.join(past_dir, hashed_name) + '.xtc' for hashed_name in hashed_names[:wave]],
00100                o='./{}_combined_traj.xtc'.format(metr_name), n='./prot_dir/prot_unfolded.ndx', cat=True, vel=False, sort=False, overwrite=True)
00101     for i in range(wave, len(hashed_names), wave):
00102         os.rename('./{}_combined_traj.xtc'.format(metr_name), './{}_combined_traj_prev.xtc'.format(metr_name))
00103         gmx_trjcat(f=[" ./{}_combined_traj_prev.xtc ".format(metr_name)] + [os.path.join(past_dir, hashed_name) + '.xtc' for hashed_name in
      hashed_names[i:i+wave]],
00104                    o='./{}_combined_traj.xtc'.format(metr_name), n='./prot_dir/prot_unfolded.ndx', cat=True, vel=False, sort=False,
      overwrite=True)
00105         if int(i / wave) % 10 == 0:
00106             print('{}/{} ({:.1f}%)'.format(int(i / wave), tot_chunks, 100 * int(i / wave) / tot_chunks))
00107
00108     if os.path.exists('./{}_combined_traj.xtc'.format(metr_name)):
00109         os.rename('./{}_combined_traj.xtc'.format(metr_name), './{}_{}_traj_best.xtc'.format(metr_name, db_to_connect))
00110     if os.path.exists('./{}_combined_traj_prev.xtc'.format(metr_name)):
00111         os.remove('./{}_combined_traj_prev.xtc'.format(metr_name))
00112     print('Done with best for {}: {}'.format(metr_name, db_to_connect))
00113
00114
00115     # ###### ENERGIES
00116     if os.path.exists('./{}_combined_energy.edr'.format(metr_name)):
00117         os.remove('./{}_combined_energy.edr'.format(metr_name))
00118     if os.path.exists('./{}_combined_energy_prev.edr'.format(metr_name)):
00119         os.remove('./{}_combined_energy_prev.edr'.format(metr_name))
00120     hashed_names = hashed_names[1:]
00121     tot_chunks = int((len(hashed_names) + 1) / wave)
00122     print('Computing energy for best trajectory for {}'.format(metr_name))
00123     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00124     gmx_eneconv(f=[os.path.join("./past", hashed_name) + '.edr' for hashed_name in hashed_names[:wave]],
      o='./{}_combined_energy.edr'.format(metr_name))
00125     for i in range(wave, len(hashed_names), wave):
00126         os.rename('./{}_combined_energy.edr'.format(metr_name), './{}_combined_energy_prev.edr'.format(metr_name))
00127         gmx_eneconv(f=["./{}_combined_energy_prev.edr".format(metr_name)] + [os.path.join("./past", hashed_name + '.edr') for hashed_name in
      hashed_names[i:i + wave if i + wave < len(hashed_names) else -1]],
00128                     o='./{}_combined_energy.edr'.format(metr_name))
00129         if int(i / wave) % 10 == 0:
00130             print('{}/{} ({:.1f}%)'.format(int(i / wave), tot_chunks, 100 * int(i / wave) / tot_chunks))
00131
00132     os.rename('./{}_combined_energy.edr'.format(metr_name), './{}_combined_energy_best.edr'.format(metr_name))
00133
00134
00135 if __name__ == '__main__':
00136     main()
00137
```

```
00138
00139 def main_energy():
00140     """
00141
00142 Returns
00143 -------
00144     Generates one .edr trajectory with energy of the frames that result in the best conformation according to the specific metric.
00145     """
00146     past_dir = './past'
00147     db_to_connect = 'results_12'
00148     polynomial = False
00149     font = {'family': 'serif',
00150             'color': 'darkred',
00151             'weight': 'normal',
00152             'size': 16,
00153             }
00154     if not os.path.exists(db_to_connect + '.sqlite3'):
00155         raise Exception('DB not found')
00156
00157     con = lite.connect(db_to_connect + '.sqlite3', check_same_thread=False, isolation_level=None)
00158     cur = con.cursor()
00159
00160     qry = "select a.name, a.hashed_name from main_storage a  where a.goal_dist= ( select min(b.goal_dist) from main_storage b)"
00161     result = cur.execute(qry)
00162     all_res = result.fetchone()
00163     name = all_res[0]
00164     spname = name.split('_')
00165     all_prev_names = ['\'{}\"".format('_'.join(spname[:i])) for i in range(1, len(spname))]
00166     long_line = ", ".join(all_prev_names)
00167
00168     qry = "select name, hashed_name from main_storage where name in ({})".format(long_line)
00169     result = cur.execute(qry)
00170     _ = result.fetchone()
00171     all_res = result.fetchall()
00172     names, hashed_names = zip(*all_res)
00173     wave = 100
00174     tot_chunks = int((len(hashed_names) + 1) / wave)
00175     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00176     gmx_eneconv(f=[os.path.join("./past", hashed_name) + '.edr' for hashed_name in hashed_names[:wave]], o='./combined_energy.edr')
00177     for i in range(wave, len(hashed_names) + 1 - wave, wave):
00178         os.rename('./combined_energy.edr', './combined_energy_prev.edr')
00179         gmx_eneconv(f=["./combined_energy_prev.edr"] + [os.path.join("./past", hashed_name + '.edr') for hashed_name in hashed_names[i:i + wave
      if i + wave < len(hashed_names) else -1]],
00180                     o='./combined_energy.edr')
00181         if int(i / wave) % 10 == 0:
00182             print('{}/{} ({:.1f}%)'.format(int(i / wave), tot_chunks, 100 * int(i / wave) / tot_chunks))
00183
00184     os.rename('./combined_energy.edr', './combined_energy_best.edr')
00185     print('Done with best')
00186
00187
00188
00189     qry = "select a.name, a.hashed_name from main_storage a "
00190     result = cur.execute(qry)
00191     _ = result.fetchone()
00192     all_res = result.fetchall()
00193     names, hashed_names = zip(*all_res)
00194
00195     # gmx_eneconv(f=[os.path.join(past_dir, hash_name+'.edr') for hash_name in hashed_names], o='./combined_energy.edr')
00196
00197     wave = 100
00198     tot_chunks = int((len(hashed_names)+1)/wave)
00199     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00200     gmx_eneconv(f=[os.path.join("./past", hashed_name)+'.edr' for hashed_name in hashed_names[:wave]], o='./combined_energy.edr')
00201     for i in range(wave, len(hashed_names)+1-wave, wave):
00202         os.rename('./combined_energy.edr', './combined_energy_prev.edr')
00203         gmx_eneconv(f=["./combined_energy_prev.edr"] +[os.path.join("./past", hashed_name + '.edr') for hashed_name in hashed_names[i:i+wave if
      i+wave < len(hashed_names) else -1]], o='./combined_energy.edr')
00204         if int(i/wave) % 10 == 0:
00205             print('{}/{} ({:.1f}%)'.format(int(i/wave), tot_chunks, 100*int(i/wave)/tot_chunks))
00206
00207     os.rename('./combined_energy.edr', './combined_energy_all_main.edr')
00208     print('Done with all main')
00209
00210
00211     qry = "select a.name, a.hashed_name from main_storage a join log b on a.id=b.id where b.dst='VIZ' order by b.timestamp"
00212     result = cur.execute(qry)
00213     _ = result.fetchone()
00214     all_res = result.fetchall()
00215     names, hashed_names = zip(*all_res)
00216
```

```
00217     wave = 100
00218     tot_chunks = int((len(hashed_names)+1)/wave)
00219     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00220     gmx_eneconv(f=[os.path.join("./past", hashed_name)+'.edr' for hashed_name in hashed_names[:wave]], o='./combined_energy.edr')
00221     for i in range(wave, len(hashed_names)+1-wave, wave):
00222         os.rename('./combined_energy.edr', './combined_energy_prev.edr')
00223         gmx_eneconv(f=["./combined_energy_prev.edr"] +[os.path.join("./past", hashed_name + '.edr') for hashed_name in hashed_names[i:i+wave if
      i+wave < len(hashed_names) else -1]], o='./combined_energy.edr')
00224         if int(i/wave) % 10 == 0:
00225             print('{}/{} ({:.1f}%)'.format(int(i/wave), tot_chunks, 100*int(i/wave)/tot_chunks))
00226
00227     os.rename('./combined_energy.edr', './combined_energy_all_viz.edr')
00228     print('Done with viz')
00229
00230
00231     # gmx_energy('./combined_energy.edr', './combined_energy.xvg', fee=True, fetemp=300)
00232
00233
```

4.31 metric_funcs.py File Reference

Namespaces

- metric_funcs

Functions

- list metric_funcs.get_knn_dist_mdsctk (str ref_file, str fitfile, str topology)

  'knn_rms' - MDSCTK tool - computes RMSD between two (or more) structures
- np.ndarray metric_funcs.get_contat_profile_mdsctk (str ref_file, str fitfile, str index, float dist=2.7)

  'contact_profile' - MDSCTK tool - computes number of contacts between two (or more) structures
- NoReturn metric_funcs.get_bb_to_angle_mdsctk (str x='noise_bb.xtc', str o='noise_angle.dat')

  'bb_xtc_to_phipsi' - MDSCTK tool - takes backbone structure and computes dihedral angles between atoms
- NoReturn metric_funcs.get_angle_to_sincos_mdsctk (str i='noise_angle.dat', str o='noise_sincos.dat')

  'angles_to_sincos' - MDSCTK tool - converts dihedrals into sin/cos values
- str metric_funcs.gen_file_for_amb_noize (str work_dir, int seeds, dict seed_dirs, str ndx_file, str top_file, str goal_file='folded_↩
  for_noise.gro', list hostnames=None, list cpu_map=None)

  Performs simulation of the NMR (not unfolded) conformation to measure ambient vibrations.
- np.ndarray metric_funcs.compute_phipsi_angles (int angl_num, str target_filename, str ndx, str stor_name=None)

  Top level function that outputs sin/cos of the dihedral angles of the provided conformation.
- np.ndarray metric_funcs.ang_dist (list target_ang, list goal_ang)

  Computes difference between two angle lists.
- NoReturn metric_funcs.save_an_file (str an_file_name, dict tol_error, list metr_order)

  Writes noise values into the specified file for future use during the restarts.
- tuple metric_funcs.get_native_contacts (str goal_prot_only, list files_to_check, str ndx_file, np.ndarray cont_corr, int atom_num, float
  dist=2.7, np.ufunc logic_fun=np.logical_xor, list h_filter=None, mp.Pool pool=None, bool just_contacts=False)

  Computes number of contacts between the goal_prot_only and files_to_check.
- NoReturn metric_funcs.and_h (mp.Queue q, np.int goal_contacts_and_h_sum, list goal_cont_h, list contacts_h, list prev_contacts_h, np.int
  and_h_dist_tot)

  Separate AND_H computation, used to be executed in parallel,.
- NoReturn metric_funcs.and_p (mp.Queue q, np.int goal_contacts_and_sum, list goal_contacts, list contacts, list prev_contacts, np.int
  prev_tot_dist)

  Separate AND computation, used to be executed in parallel,.
- NoReturn metric_funcs.rmsd (mp.Queue q, str combined_pg, str temp_xtc_file, str goal_prot_only, np.float64 prev_tot_dist)

  Separate RMSD computation, used to be executed in parallel,.
- NoReturn metric_funcs.angl (mp.Queue q, int angl_num, str temp_xtc_file, str init_bb_ndx, list pangl, list goal_angles, np.float64 prev↩
  _tot_dist)

  Separate ANGL computation, used to be executed in parallel,.
- list metric_funcs.compute_metric (str past_dir, list new_nodes_names, int tot_seeds, str combined_pg, str temp_xtc_file, str goal_↩
  prot_only, dict node_info, int angl_num, str init_bb_ndx, list goal_angles, str init_prot_only, list files_for_trjcat, str ndx_file_↩
  init, list goal_cont_h, int atom_num, float cont_dist, list h_filter_init, list goal_contacts, int cur_metric, np.int goal_contacts_and_h_↩
  sum, np.int goal_contacts_and_sum, bool chance_to_reuse=False, mp.Pool cpu_pool=None, bool compute_all_at_once=True)

  Computes metric distances from the previous node and to the goal (NMR) conformation.
- list metric_funcs.compute_init_metric (str past_dir, int tot_seeds, str init_xtc, str goal_xtc, str goal_prot_only, int angl_num, str
  init_bb_ndx, np.ndarray goal_angles, str init_prot_only, str ndx_file_init, np.ndarray goal_cont_h, int atom_num, float cont_dist, np.↩
  ndarray h_filter_init, np.ndarray goal_contacts, np.int 64 goal_contacts_and_h_sum, np.int 64 goal_contacts_and_sum)

Special case of the "compute_metric".

- str metric_funcs.select_metrics_by_snr (list cur_nodes, dict prev_node, list metric_names, dict tol_error, bool compute_all_at_once, list alowed_metrics, str cur_metr)

  SNR approach to a metric selection.

4.32 metric_funcs.py

```
00001 """This file contains functions to compute various metric distances.
00002
00003 .. module:: GMDA_main
00004     :platform: linux
00005
00006 .. moduleauthor:: Ivan Syzonenko <is2k@mtmail.mtsu.edu>
00007 """
00008 __license__ = "MIT"
00009 __docformat__ = 'reStructuredText'
00010
00011
00012 import numpy as np
00013 import os
00014 import subprocess
00015 import multiprocessing as mp
00016 from scipy.sparse import csc_matrix, save_npz, load_npz
00017 import zlib
00018 from typing import NoReturn
00019 # from shutil import copy2 as cp2
00020
00021 from helper_funcs import get_digest
00022 from gmx_wrappers import gmx_grompp, gmx_mdrun, gmx_trjcat, gmx_trjconv, gmx_mdrun_mpi
00023 # from gen_mdp import get_mdp
00024
00025
00026 def get_knn_dist_mdsctk(ref_file: str, fitfile: str, topology: str) -> list:
00027     """'knn_rms' - MDSCTK tool - computes RMSD between two (or more) structures
00028
00029     Args:
00030         :param str ref_file: reference file - .xtc or .gro filename
00031         :param str fitfile: .xtc or .gro filename - structure will be centered according to the fitfile and used in distance computation
00032         :param str topology: .top topology file of the simulation box
00033
00034     Returns:
00035         :return: list of RMSD distances from all frames to the goal
00036         :rtype: list
00037     """
00038     if os.path.exists(os.path.join(os.getcwd(), 'local.comp')):
00039         mdsctk_bash = 'source /opt/mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00040     else:
00041         mdsctk_bash = 'source ./mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00042
00043     command = '{} knn_rms -s {} -p {} -r {} -f {}'.format(mdsctk_bash, 0, topology, ref_file, fitfile)
00044     proc_obj = subprocess.Popen(os.path.expandvars(command), stdout=subprocess.PIPE, shell=True, stderr=None)
00045     try:
00046         output, error = proc_obj.communicate()
00047     except Exception as e:
00048         print(e)
00049         return None
00050     if error:
00051         error = error.decode("utf-8")
00052         if 'error' in error.lower():
00053             print(error)
00054     if output:
00055         output = output.decode("utf-8")
00056         if 'error' in output.lower():
00057             print(output)
00058     dist_arr = np.fromfile('distances.dat', dtype=np.double)
00059     os.remove('distances.dat')
00060     os.remove('indices.dat')
00061
00062     return dist_arr.tolist()
00063
00064
00065 def get_contat_profile_mdsctk(ref_file: str, fitfile: str, index: str, dist: float = 2.7) -> np.ndarray:
00066     """'contact_profile' - MDSCTK tool - computes number of contacts between two (or more) structures
00067
00068     Args:
00069         :param str ref_file: reference file - .xtc or .gro filename
00070         :param str fitfile: .xtc or .gro filename - structure will be centered according
00071          to the fitfile and used in distance computation
00072         :param str index: .ndx file to compute distance among particular atoms
00073         :param floatdist: in Angstroms - how close should two atoms be, so treat them as a contact
```

```
00074
00075     Returns:
00076         :return: ndarray, first value - number of indices with contacts, next N indices are atoms with contact
00077         :rtype np.ndarray
00078     """
00079     if os.path.exists(os.path.join(os.getcwd(), 'local.comp')):
00080         mdsctk_bash = 'source /opt/mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00081     else:
00082         mdsctk_bash = 'source ./mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00083
00084     slash_pos = fitfile.rfind('/')
00085     if slash_pos >= 0:
00086         unique_name = '{}/{}.svi'.format(fitfile[:slash_pos], fitfile.split('/')[-1].split('.')[0])
00087     else:
00088         unique_name = '{}.svi'.format(fitfile.split('/')[-1].split('.')[0])
00089     command = '{} contact_profile -p {} -x {} -n {} -e {} -i {} -d /dev/null 2>/dev/null 1>/dev/null'.format(
00090         mdsctk_bash, ref_file, fitfile, index, dist, unique_name)
00091     proc_obj = subprocess.Popen(os.path.expandvars(command), stdout=None, shell=True, stderr=None)
00092     try:
00093         output, error = proc_obj.communicate()
00094     except Exception as e:
00095         print(command)
00096         print(e)
00097         return None
00098     if error:
00099         error = error.decode("utf-8")
00100         if 'error' in error.lower():
00101             print(command)
00102             print(error)
00103     if output:
00104         output = output.decode("utf-8")
00105         if 'error' in output.lower():
00106             print(command)
00107             print(output)
00108     cont_arr = np.fromfile(unique_name, dtype=np.uint32)
00109
00110     os.remove(unique_name)
00111
00112     return cont_arr
00113
00114
00115 def get_bb_to_angle_mdsctk(x: str = 'noise_bb.xtc', o: str = 'noise_angle.dat') -> NoReturn:
00116     """'bb_xtc_to_phipsi' - MDSCTK tool - takes backbone structure and computes dihedral angles between atoms
00117
00118     Args:
00119         :param str x: backbone input trajectory
00120         :param str o: filename of the binary C array
00121
00122     Returns:
00123     Generates a file with dihedral angles.
00124     """
00125     if os.path.exists(os.path.join(os.getcwd(), 'local.comp')):
00126         mdsctk_bash = 'source /opt/mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00127     else:
00128         mdsctk_bash = 'source ./mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00129     #  bb_xtc_to_phipsi -x traj_bb_315.xtc -o angles_bb_315.dat
00130     command = '{} bb_xtc_to_phipsi -x {} -o {} 2>/dev/null 1>/dev/null'.format(
00131         mdsctk_bash, x, o)
00132     proc_obj = subprocess.Popen(
00133         os.path.expandvars(command), stdout=None, shell=True, stderr=None)
00134     # proc_obj = subprocess.Popen(os.path.expandvars(command), stdout=subprocess.PIPE, shell=True, stderr=None)
00135     try:
00136         output, error = proc_obj.communicate()
00137     except Exception as e:
00138         print(command)
00139         # print(e)
00140         raise Exception(e)
00141     if error:
00142         error = error.decode("utf-8")
00143         if 'error' in error.lower():
00144             print(command)
00145             print(error)
00146     if output:
00147         output = output.decode("utf-8")
00148         if 'error' in output.lower():
00149             print(command)
00150             print(output)
00151
00152
00153 def get_angle_to_sincos_mdsctk(i: str='noise_angle.dat', o: str='noise_sincos.dat') -> NoReturn:
00154     """'angles_to_sincos' - MDSCTK tool - converts dihedrals into sin/cos values
```

```
00155
00156     Args:
00157         :param str i: filename that contains angle values in the  binary form
00158         :param str o: filename that contains sin/cos values in the  binary form
00159
00160     Returns:
00161     Generates file with sin/cos values.
00162     """
00163     if os.path.exists(os.path.join(os.getcwd(), 'local.comp')):
00164         mdsctk_bash = 'source /opt/mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00165     else:
00166         mdsctk_bash = 'source ./mdsctk/MDSCTK.bash ; '  # need this since load_envbash does not work
00167     # angles_to_sincos -i angles_bb_315.dat -o sincos_bb_315.dat
00168     command = '{} angles_to_sincos -i {} -o {} 2>/dev/null 1>/dev/null'.format(
00169         mdsctk_bash, i, o)
00170     proc_obj = subprocess.Popen(
00171         os.path.expandvars(command), stdout=None, shell=True, stderr=None)
00172     # proc_obj = subprocess.Popen(os.path.expandvars(command), stdout=subprocess.PIPE, shell=True, stderr=None)
00173     try:
00174         output, error = proc_obj.communicate()
00175     except Exception as e:
00176         print(command)
00177         # print(e)
00178         raise Exception(e)
00179     if error:
00180         error = error.decode("utf-8")
00181         if 'error' in error.lower():
00182             print(command)
00183             print(error)
00184     if output:
00185         output = output.decode("utf-8")
00186         if 'error' in output.lower():
00187             print(command)
00188             print(output)
00189
00190
00191 def gen_file_for_amb_noize(work_dir: str, seeds: int, seed_dirs: dict, ndx_file: str, top_file: str,
00192                            goal_file: str = 'folded_for_noise.gro', hostnames: list = None, cpu_map: list = None) -> str:
00193     """Performs simulation of the NMR (not unfolded) conformation to measure ambient vibrations
00194
00195     Args:
00196         :param str work_dir: path to the working directory
00197         :param int seeds: number of seed in the current run
00198         :param dict seed_dirs: paths to directories where emulation is performed with particular seed
00199         :param str ndx_file: index file to extract only specific atoms (strip water)
00200         :param str top_file: .top topology file of the simulation box
00201         :param str goal_file: goal (typically NMR) conformation
00202         :param list hostnames: for MPI, to perform parallel computation
00203         :param listcpu_map: number of cores for particular task (seed)
00204
00205     Returns:
00206         :return: filename which contains all seed simulations concatenated
00207         :rtype: str
00208
00209     Generates a file with trajectories from the goal.
00210     """
00211     # if file ambient.rmsd found, read it
00212
00213     temp_xtc_file = 'noise.xtc'
00214     #  generate and save if not found
00215     if temp_xtc_file not in os.walk(".").__next__()[2]:
00216         pid_arr = list()
00217         for i, seed in enumerate(seeds):
00218
00219             gmx_grompp(work_dir, seed, top_file,
00220                        goal_file[:-4])  # TODO: update filenames
00221
00222             if hostnames:
00223                 md_process = mp.Process(target=gmx_mdrun_mpi,
00224                                         args=(work_dir, seed, os.path.join(seed_dirs[seed], 'md.gro'), hostnames[i], cpu_map[i]))
00225                 # gmx_mdrun_mpi(work_dir, seed, seed_dirs[seed] + '/md.gro', hostnames[i], cpu_map[i])
00226             else:
00227                 md_process = mp.Process(target=gmx_mdrun, args=(work_dir, seed, os.path.join(seed_dirs[seed], 'md.gro')))
00228                 # gmx_mdrun(work_dir, seed, seed_dirs[seed] + '/md.gro')
00229             md_process.start()
00230             pid_arr.append(md_process)
00231         [proc.join() for proc in pid_arr]
00232         for i, seed in enumerate(seeds):
00233             gmx_trjconv(
00234                 f=os.path.join(seed_dirs[seed], 'md.xtc'),
00235                 o=os.path.join(seed_dirs[seed], 'md_prot.xtc'),
```

```
00236                 n=ndx_file,
00237                 b=1)  # , dump=20
00238 
00239         results_arr = list(os.path.join(os.path.join(work_dir, str(seed)), 'md_prot.xtc') for seed in seeds)
00240         gmx_trjcat(f=results_arr, o=temp_xtc_file, n=ndx_file, cat=True, vel=False, sort=False, overwrite=True)
00241 
00242     return temp_xtc_file
00243 
00244 
00245 # def get_ambient_noise_rmsd(goal_xtc, noize_file, goal_prot_only, mul=0.8):
00246 #     dist_arr = get_knn_dist_mdsctk(goal_xtc, noize_file, goal_prot_only)
00247 #     min_rmsd = min(dist_arr)*mul  # I expect that current min does not represent real min.
00248 #     print('Min rmsd for simulation is going to be : ', min_rmsd)
00249 #     return min_rmsd
00250 #
00251 #
00252 # def get_ambient_noise_angles(num_el, gro_file, noize_file, goal_bb_ndx, goal_angles, mul=0.8):
00253 #     # generate filename
00254 #     # convert_gro_to_xtc(gro_file, goal_bb_ndx)
00255 #     sincos_file = 'noise_sincos.dat'
00256 #     noize_file_bb = 'noize_bb.xtc'
00257 #     angle_file = 'noise_angle.dat'
00258 #
00259 #     gmx_trjconv(f=noize_file, o=noize_file_bb, n=goal_bb_ndx, s=gro_file)
00260 #     get_bb_to_angle_mdsctk(x=noize_file_bb, o=angle_file)
00261 #     get_angle_to_sincos_mdsctk(i=angle_file, o=sincos_file)
00262 #
00263 #     os.remove(angle_file)
00264 #
00265 #     with open(sincos_file, 'rb') as file:
00266 #         initial_1d_array = np.frombuffer(file.read(), dtype=np.float64, count=-1)
00267 #     check_arr = np.reshape(initial_1d_array, (-1, num_el*2))
00268 #     del initial_1d_array
00269 #
00270 #     res_arr = [None]*check_arr.shape[0]
00271 #     for i in range(check_arr.shape[0]):
00272 #         res_arr[i] = np.sum(abs(check_arr[i] - goal_angles))
00273 #     return float(np.min(res_arr)*mul)
00274 
00275 
00276 def compute_phipsi_angles(angl_num: int, target_filename: str, ndx: str, stor_name: str = None) -> np.ndarray:
00277     """Top level function that outputs sin/cos of the dihedral angles of the provided conformation.
00278 
00279     Args:
00280         :param int angl_num: total number of angles in the protein
00281         :param str target_filename:
00282         :param str ndx: index file to extract only specific atoms (extract the backbone)
00283         :param str stor_name:
00284 
00285     Returns:
00286         :return: array with sin/cos values of the backbone angles.
00287         :rtype: np.ndarray
00288     """
00289     xtc_filename = "{}.xtc".format(target_filename)
00290     if stor_name is None:  # then create temp file in /dev/shm
00291         bb_filename = "{}_bb.xtc".format(target_filename)
00292         ang_filename = "{}_bb.ang".format(target_filename)
00293         sin_cos_filename = "{}_bb.sc".format(target_filename)
00294         # making sure that we do not reuse old files
00295         if os.path.exists(bb_filename):
00296             os.remove(bb_filename)
00297         if os.path.exists(bb_filename):
00298             os.remove(bb_filename)
00299     else:  # then store in ./past/
00300         bb_filename = "{}_bb.xtc".format(stor_name)
00301         ang_filename = "{}_bb.ang".format(stor_name)
00302         sin_cos_filename = "{}_bb.sc".format(stor_name)
00303 
00304     gmx_trjconv(f=xtc_filename, o=bb_filename, n=ndx)
00305     get_bb_to_angle_mdsctk(x=bb_filename, o=ang_filename)
00306     get_angle_to_sincos_mdsctk(i=ang_filename, o=sin_cos_filename)
00307 
00308     with open(sin_cos_filename, 'rb') as file:
00309         initial_1d_array = np.frombuffer(file.read(), dtype=np.float64, count=-1)
00310     check_arr = np.reshape(initial_1d_array, (-1, angl_num * 2))
00311     if len(check_arr) == 1:
00312         return check_arr[0]
00313     return check_arr
00314 
00315 
00316 def ang_dist(target_ang: list, goal_ang: list) -> np.ndarray:
```

```
00317     """Computes difference between two angle lists.
00318
00319     Args:
00320         :param list target_ang: angles to test
00321         :param list goal_ang: goal angles
00322
00323     Returns:
00324         :return: one number when input is a list or list of sums in case intput is list of lists
00325         :rtype: np.ndarray
00326     """
00327     if target_ang.shape[0] == 1 or target_ang.ndim == 1:
00328         return np.abs(target_ang - goal_ang).sum()
00329     else:
00330         return [np.abs(target_ang[i] - goal_ang).sum() for i in range(target_ang.shape[0])]
00331
00332
00333 # def get_ambient_noise_contacts_xor(goal_prot_only, noize_xtc, ndx_file_cont, atom_num, logic_fun,
00334 # corr_contacts, cont_dist, prev_cont, mult=0.8):
00335 #     cont_sum, nat_contacts = get_native_contacts(goal_prot_only, [noize_xtc], ndx_file_cont,
00336 # corr_contacts, atom_num, dist=cont_dist, logic_fun=logic_fun)
00337 #     return max(1,int(min(abs(prev_cont - cont_sum))*mult))
00338
00339 # def get_ambient_noise_contacts(goal_prot_only, noize_xtc, ndx_file_cont, atom_num, logic_fun,
00340 # corr_contacts, cont_dist, prev_cont, mult=0.8):
00341 #     cont_sum, nat_contacts = get_native_contacts(goal_prot_only, [noize_xtc], ndx_file_cont,
00342 # corr_contacts, atom_num, dist=cont_dist, logic_fun=logic_fun)
00343 #     return max(1, int(min(abs(prev_cont - cont_sum)) * mult))
00344
00345
00346 def save_an_file(an_file_name: str, tol_error: dict, metr_order: list) -> NoReturn:
00347     """Writes noise values into the specified file for future use during the restarts
00348
00349     Args:
00350         :param str an_file_name: ambient noise filename
00351         :param dict tol_error: dict with ambient noise values for each metric
00352         :param list metr_order: list of metrics used in the current run
00353
00354     Returns:
00355     Generates a file with noise values.
00356     """
00357     with open(an_file_name, 'w') as f:
00358         for metr_name in metr_order:
00359             f.write('{}\n'.format(tol_error[metr_name]))
00360
00361
00362 def get_native_contacts(goal_prot_only: str, files_to_check: list, ndx_file: str, cont_corr: np.ndarray, atom_num: int,
00363                         dist: float = 2.7, logic_fun: np.ufunc = np.logical_xor, h_filter: list = None,
00364                         pool: mp.Pool = None, just_contacts: bool = False) -> tuple:  # goal_prot_only, files_for_trjcat, ndx_file
00365     """Computes number of contacts between the goal_prot_only and files_to_check.
00366
00367     If files to check is a single list of contacts, then function returns int and list
00368     Otherwise it returns list of ints and list of lists
00369
00370     Args:
00371         :param str goal_prot_only: .gro filename with stripped waters and salt
00372         :param list files_to_check: .xtc filename with frames we want to measure number of contacts with the goal
00373         :param str ndx_file: .ndx - index filename to select protein only in .xtc
00374         :param np.ndarray cont_corr: correct contacts between goal and goal (no mistakes) to compare with the files_to_check
00375         :param int atom_num: number of atoms used for memory (structure) allocation
00376         :param dist: distance that defines a contact
00377         :param np.ufunc logic_fun: defines what relation between the goal and the files_to_check we want to measure - AND, XOR
00378         :type logic_fun: Numpy logic function, typically logical_xor or logical_and
00379         :param list h_filter: boolean array with 1s in positions of H atoms, used to filter the final contacts
00380         :param mp.Pool pool: CPU pool - passed, since each instance does not deallocate the RAM
00381         :param bool just_contacts: flags to skip computation of the sum of correct contacts
00382
00383     Returns:
00384         :return: sum of the correct contacts and contacts.
00385         :rtype: tuple
00386     """
00387     # nat_cont_arr = list()
00388     # contacts = list()
00389     if len(files_to_check) == 0:
00390         return None
00391     elif len(files_to_check) > 1:  # case for many files with one frame
00392         if pool is None:
00393             # pool = mp.Pool(mp.cpu_count())  # creation pool every time creates memory leak on python3.6.6 compiled with gcc 8.2.0
00394             raise Exception('Please pass pool variable')
00395         # ind = [get_contat_profile_mdsctk(goal_prot_only, file, ndx_file, dist)[1:] for file in files_to_check]
00396         ind = [elem[1:] for elem in pool.starmap(get_contat_profile_mdsctk,
00397                                                  ((goal_prot_only, file, ndx_file, dist) for file in files_to_check))]
```

```
00398         # corr_len = [elem[:1] for elem in ind if len(elem) > 0]
00399         contacts = [None] * len(ind)
00400         for i in range(len(ind)):
00401             elem = np.zeros(atom_num * atom_num, dtype=np.bool)
00402             elem[ind[i]] = True
00403             contacts[i] = elem
00404         del ind, elem, i
00405     else:  # case for one file with any number of frames
00406         cont_arr = get_contat_profile_mdsctk(goal_prot_only, files_to_check[0], ndx_file, dist)
00407         # print('Done with cont prof')
00408         if cont_arr[0] + 1 == len(cont_arr):  # we have only one frame
00409             full_arr = np.zeros(atom_num * atom_num, dtype=np.bool)
00410             full_arr[cont_arr[1:]] = True
00411             contacts = [full_arr]
00412             del full_arr
00413         else:  # we have many frames
00414             tot_ind = 0
00415             contacts = list()
00416             while tot_ind < len(cont_arr):
00417                 tot_ind += 1
00418                 next_ind = tot_ind + cont_arr[tot_ind - 1]
00419                 full_arr = np.zeros(atom_num * atom_num, dtype=np.bool)
00420                 full_arr[cont_arr[tot_ind:next_ind]] = True
00421                 contacts.append(full_arr)
00422                 tot_ind += cont_arr[tot_ind - 1]
00423             del cont_arr, tot_ind, next_ind, full_arr
00424     if not just_contacts:
00425         if h_filter is not None:
00426             contacts = [np.logical_and(arr_elem, h_filter) for arr_elem in contacts]  # while here we can just use logic_fun,
00427             # since we use filter only with AND to compute AND_H, I took a safe path
00428         nat_cont_sum_arr = [logic_fun(arr_elem, cont_corr).sum() for arr_elem in contacts]
00429     else:
00430         nat_cont_sum_arr = [None] * len(contacts)
00431
00432     if len(nat_cont_sum_arr) == 1:
00433         return nat_cont_sum_arr[0], contacts[0]
00434     return nat_cont_sum_arr, contacts
00435
00436
00437 def and_h(q: mp.Queue, goal_contacts_and_h_sum: np.int, goal_cont_h: list, contacts_h: list, prev_contacts_h: list, and_h_dist_tot: np.int) ->
      NoReturn:
00438     """Separate AND_H computation, used to be executed in parallel,
00439
00440     NOT used anymore since does not result in any significant speed up, but left here "just in case".
00441
00442     Args:
00443         :param mp.Queue q: queue used to communicate with the parent process
00444         :param np.int goal_contacts_and_h_sum: exact number of NMR contacts
00445         :param list goal_cont_h: correct (NMR) contacts
00446         :param list contacts_h: current nodes' contacts
00447         :param list prev_contacts_h: previous node contacts
00448         :param np.int and_h_dist_tot: distance accumulated from the origin
00449
00450     Returns:
00451         :return: Returns by putting into the queue (metric to goal, metric from previous, total traveled in metric units).
00452     """
00453     goal_cont_dist_and_h = goal_contacts_and_h_sum - [np.logical_and(arr_elem, goal_cont_h).sum() for arr_elem in contacts_h]
00454     prev_cont_dist_and_h_1 = [np.logical_xor(arr_elem, prev_contacts_h).sum() for arr_elem in contacts_h]
00455     prev_cont_dist_and_h_2 = [arr_elem.sum() for arr_elem in contacts_h] + prev_contacts_h.sum()
00456     prev_cont_dist_and_h_2 = prev_cont_dist_and_h_2 / 2 - \
00457         [elem.sum() for elem in [np.logical_and(arr_elem, prev_contacts_h) for arr_elem in contacts_h]]
00458     total_cont_dist_and_h = and_h_dist_tot + prev_cont_dist_and_h_1
00459     q.put((goal_cont_dist_and_h, prev_cont_dist_and_h_2, total_cont_dist_and_h))
00460
00461
00462 def and_p(q: mp.Queue, goal_contacts_and_sum: np.int, goal_contacts: list, contacts: list, prev_contacts: list, prev_tot_dist: np.int) ->
      NoReturn:
00463     """Separate AND computation, used to be executed in parallel,
00464
00465     NOT used anymore since does not result in any significant speed up, but left here "just in case".
00466
00467     Args:
00468         :param mp.Queue q: queue used to communicate with the parent process
00469         :param np.int goal_contacts_and_sum: exact number of NMR contacts
00470         :param list goal_contacts: correct (NMR) contacts
00471         :param list contacts: current nodes' contacts
00472         :param list prev_contacts: previous node contacts
00473         :param np.int prev_tot_dist: distance accumulated from the origin
00474
00475     Returns:
00476         :return: Returns by putting into the queue (metric to goal, metric from previous, total traveled in metric units).
```

```
00477     """
00478     goal_cont_dist_and = goal_contacts_and_sum - [np.logical_and(arr_elem, goal_contacts).sum() for arr_elem in contacts]
00479     prev_cont_dist_and_1 = [np.logical_xor(arr_elem, prev_contacts).sum() for arr_elem in contacts]
00480     prev_cont_dist_and_2 = [arr_elem.sum() for arr_elem in contacts] + prev_contacts.sum()
00481     prev_cont_dist_and_2 = prev_cont_dist_and_2 / 2 - \
00482         [elem.sum() for elem in [np.logical_and(arr_elem, prev_contacts) for arr_elem in contacts]]
00483     total_cont_dist_and = prev_tot_dist + prev_cont_dist_and_1
00484     q.put((goal_cont_dist_and, prev_cont_dist_and_2, total_cont_dist_and))
00485
00486
00487 def rmsd(q: mp.Queue, combined_pg: str, temp_xtc_file: str, goal_prot_only: str, prev_tot_dist: np.float64) -> NoReturn:
00488     """Separate RMSD computation, used to be executed in parallel,
00489
00490     NOT used anymore since does not result in any significant speed up, but left here "just in case".
00491
00492     Args:
00493         :param mp.Queue q: queue used to communicate with the parent process
00494         :param str combined_pg: two frames previous and goal
00495         :param str temp_xtc_file: new frames (same as number of seeds) you want to measure distance from previous and to the goal
00496         :param str goal_prot_only: goal protein only conformation
00497         :param np.float64 rev_tot_dist: distance accumulated from the origin
00498
00499     Returns:
00500         :return: Returns by putting into the queue (metric to goal, metric from previous, total traveled in metric units).
00501     """
00502     dist_arr = get_knn_dist_mdsctk(combined_pg, temp_xtc_file, goal_prot_only)
00503     from_prev_dist = dist_arr[0::2]
00504     rmsd_to_goal = dist_arr[1::2]
00505     rmsd_total_trav = [prev_tot_dist + elem for elem in from_prev_dist]
00506     q.put((rmsd_to_goal, from_prev_dist, rmsd_total_trav))
00507
00508
00509 def angl(q: mp.Queue, angl_num: int, temp_xtc_file: str, init_bb_ndx: str, pangl: list, goal_angles: list, prev_tot_dist: np.float64) ->
      NoReturn:
00510     """Separate ANGL computation, used to be executed in parallel,
00511
00512     NOT used anymore since does not result in any significant speed up, but left here "just in case".
00513
00514     Args:
00515         :param mp.Queue q: queue used to communicate with the parent process
00516         :param int angl_num: total number of angles in the protein
00517         :param str temp_xtc_file: new frames (same as number of seeds) you want to measure distance from previous and to the goal
00518         :param str init_bb_ndx: .ndx to extract the backbone atoms
00519         :param list pangl: previous node angles
00520         :param list goal_angles: correct angles (NMR angles)
00521         :param np.float64 prev_tot_dist: distance accumulated from the origin
00522
00523     Returns:
00524         :return: Returns by putting into the queue (metric to goal, metric from previous, total traveled in metric units).
00525     """
00526     cur_angles = compute_phipsi_angles(angl_num, temp_xtc_file.split('.')[0], init_bb_ndx)
00527     angl_sum_from_prev = ang_dist(cur_angles, pangl)
00528     angl_sum_to_goal = ang_dist(cur_angles, goal_angles)
00529     angl_sum_tot = prev_tot_dist + angl_sum_from_prev
00530     q.put((angl_sum_to_goal, angl_sum_from_prev, angl_sum_tot, cur_angles))
00531
00532
00533 def compute_metric(past_dir: str, new_nodes_names: list, tot_seeds: int, combined_pg: str, temp_xtc_file: str, goal_prot_only: str, node_info:
      dict, angl_num: int,
00534                    init_bb_ndx: str, goal_angles: list, init_prot_only: str, files_for_trjcat: list, ndx_file_init: str, goal_cont_h: list,
      atom_num: int,
00535                    cont_dist: float, h_filter_init: list, goal_contacts: list, cur_metric: int, goal_contacts_and_h_sum: np.int,
      goal_contacts_and_sum: np.int,
00536                    chance_to_reuse: bool = False, cpu_pool: mp.Pool = None, compute_all_at_once: bool = True) -> list:
00537     """Computes metric distances from the previous node and to the goal (NMR) conformation.
00538
00539     Before I was computing metrics separately, but computing them all at once add very little overhead
00540      and allows to track trajectory behavior, so later I fixed only the code with all at once option.
00541
00542     Args:
00543         :param str past_dir: path to the directory with prior computation results
00544         :param list new_nodes_names: full names of newly computed nodes (not current)
00545         :param int tot_seeds: total number of seed in the current run
00546         :param str combined_pg: previous and goal frames combined into one trajectory
00547         :param str temp_xtc_file: new nodes' final frames
00548         :param str goal_prot_only: NMR (folded) conformation without water and salt (protein only)
00549         :param dict node_info: info about the current node (not just computed, but rather previous)
00550         :param int angl_num: number of dihedral angles in the protein
00551         :param str init_bb_ndx: index file with backbone atom positions for the initial conformation
00552         :param list goal_angles: angle values of the NMR structure
00553         :param str init_prot_only: initial (unfolded) conformation without water and salt (protein only)
```

```
00554         :param list files_for_trjcat: list of newly computed nodes (files, with hash as a name)
00555         :param str ndx_file_init: index file with backbone atom positions for the NMR conformation
00556         :param list goal_cont_h: contact values of the NMR structure (hydrogens only)
00557         :param int atom_num: total number of atoms in the protein (same for folded and unfolded)
00558         :param float cont_dist: distance between atoms treated as 'contact'
00559         :param list h_filter_init: positions of the hydrogen atoms in the initial (unfolded) conformation
00560         :param list goal_contacts: list of correct contacts in the NMR (folded) conformation
00561         :param int cur_metric: metric index
00562         :param np.int goal_contacts_and_h_sum: total sum of the contacts between hydrogents in the NMR (folded) conformation
00563         :param np.int goal_contacts_and_sum: total sum of the contacts in the NMR (folded) conformation
00564         :param bool chance_to_reuse:
00565         :param mp.Pool cpu_pool: CPU pool for local parallel processing
00566         :param bool compute_all_at_once: toggle whether to compute all metrics at the same time or not (yes, if no check the code)
00567 
00568     Returns:
00569         :return: new nodes with all metrics (compute_all_at_once only) and current metric distances
00570         :rtype: list
00571     """
00572     # global extra_past
00573     new_nodes = [None] * tot_seeds
00574     # prev_contacts = node_info['contacts']
00575     try:
00576         prev_contacts = load_npz(os.path.join(past_dir, '{}.cont.npz'.format(node_info['digest_name']))).toarray()
00577     except:
00578         print('Previous contact do not exists. Probably error in the previous step.\nFile: ',
00579               os.path.join(past_dir, '{}.cont.npz'.format(node_info['digest_name'])),
00580               ' was not found')
00581         exit(-10)
00582         # prev_contacts = load_npz(os.path.join(extra_past, '{}.cont.npz'.format(node_info['digest_name']))).toarray()
00583     digests = [get_digest(new_nodes_names[i]) for i in range(tot_seeds)]
00584     if compute_all_at_once:
00585         # Parallel approach does not work on small/medium proteins. Overhead of proc creation is more than time to compute.
00586         # However, when you decide to speed up execution, make only angl dist to be computed in sep process.
00587         # q = mp.Queue()
00588         # pid = multiprocessing.Process(target=angl, args=(q, angl_num, temp_xtc_file, init_bb_ndx, node_info['angles'],
00589         # goal_angles, node_info['ANGL_dist_total']))
00590         # pid.start()
00591 
00592         # *********  PREP ************
00593         reusing_old_cont = False
00594         # if chance_to_reuse:
00595         try:  # lets always check for previous files and regenerate them in case of the error - incomplete or do not exist
00596             contacts = [load_npz(os.path.join(past_dir, '{}.cont.npz'.format(digests[i]))).toarray() for i in range(tot_seeds)]
00597             reusing_old_cont = True
00598         except OSError:
00599             contacts = get_native_contacts(init_prot_only, files_for_trjcat, ndx_file_init, None,
00600                                            atom_num, cont_dist, None, pool=cpu_pool, just_contacts=True)[1]
00601         # else:
00602         #     contacts = get_native_contacts(init_prot_only, files_for_trjcat, ndx_file_init, None,
00603         #                                    atom_num, cont_dist, None, pool=cpu_pool, just_contacts=True)[1]
00604 
00605         # print(init_prot_only, files_for_trjcat, ndx_file_init, atom_num, cont_dist)
00606         #  Cont prep
00607         contacts_h = [np.logical_and(arr_elem, h_filter_init) for arr_elem in contacts]
00608         prev_contacts_h = np.logical_and(prev_contacts, h_filter_init)
00609 
00610         # ************** PAR ************
00611         # q = [mp.Queue() for i in range(4)]
00612         # bad approach
00613         # par_metr = [multiprocessing.Process(target=and_h, args=(q[0], goal_contacts_and_h_sum, goal_cont_h, contacts_h,
00614         # prev_contacts_h, node_info['AND_H_dist_total'])),
00615         #             multiprocessing.Process(target=and_p, args=(q[1], goal_contacts_and_sum, goal_contacts, contacts,
00616         # prev_contacts, node_info['AND_dist_total'])),
00617         #             multiprocessing.Process(target=rmsd, args=(q[2], combined_pg, temp_xtc_file,
00618         # goal_prot_only, node_info['RMSD_dist_total'])),
00619         #             multiprocessing.Process(target=angl, args=(q[3], angl_num, temp_xtc_file, init_bb_ndx,
00620         # node_info['angles'], goal_angles, node_info['ANGL_dist_total']))]
00621         # [pid.start() for pid in par_metr]
00622         # [pid.join() for pid in par_metr]
00623         # goal_cont_dist_and_h, prev_cont_dist_and_h_2, total_cont_dist_and_h = q[0].get()
00624         # goal_cont_dist_and, prev_cont_dist_and_2, total_cont_dist_and = q[1].get()
00625         # rmsd_to_goal, from_prev_dist, rmsd_total_trav = q[2].get()
00626         # angl_sum_to_goal, angl_sum_from_prev, angl_sum_tot, cur_angles = q[3].get()
00627         #
00628         # better approach
00629         # q = [mp.Queue() for i in range(4)]
00630         # pid = multiprocessing.Process(target=angl, args=(q[3], angl_num, temp_xtc_file, init_bb_ndx, node_info['angles'],
00631         # goal_angles, node_info['ANGL_dist_total']))
00632         # pid.start()
00633         # and_h(q[0], goal_contacts_and_h_sum, goal_cont_h, contacts_h, prev_contacts_h, node_info['AND_H_dist_total'])
00634         # and_p(q[1], goal_contacts_and_sum, goal_contacts, contacts, prev_contacts, node_info['AND_dist_total'])
```

```
00635         # rmsd(q[2], combined_pg, temp_xtc_file, goal_prot_only, node_info['RMSD_dist_total'])
00636         # pid.join()
00637         # angl_sum_to_goal, angl_sum_from_prev, angl_sum_tot, cur_angles = q[3].get()
00638 
00639         # *********  RMSD ************
00640         dist_arr = get_knn_dist_mdsctk(combined_pg, temp_xtc_file, goal_prot_only)
00641         from_prev_dist = dist_arr[0::2]
00642         rmsd_to_goal = dist_arr[1::2]
00643         rmsd_total_trav = [node_info['RMSD_dist_total'] + elem for elem in from_prev_dist]
00644 
00645         # *********  ANG ************
00646         reusing_old_angl = False
00647         # if chance_to_reuse:
00648         try:
00649             cur_angles = [np.fromfile(os.path.join(past_dir, '{}.angl'.format(digests[i])), dtype=np.float32) for i in range(tot_seeds)]
00650             cur_angles = np.asarray(cur_angles, dtype=np.float32)
00651             reusing_old_angl = True
00652         except OSError:
00653             cur_angles = compute_phipsi_angles(angl_num, temp_xtc_file.split('.')[0], init_bb_ndx)
00654         # else:
00655         #     cur_angles = compute_phipsi_angles(angl_num, temp_xtc_file.split('.')[0], init_bb_ndx)
00656 
00657         # angl_sum_from_prev = ang_dist(cur_angles, node_info['angles'])
00658         # if os.path.exists(os.path.join(past_dir, '{}.angl'.format(node_info['digest_name']))):
00659         try:
00660             angl_sum_from_prev = ang_dist(cur_angles, np.fromfile(os.path.join(past_dir, '{}.angl'.format(node_info['digest_name'])),
      dtype=np.float32))
00661         except Exception as e:
00662             print('Error during previous angle read.\nCheck ', os.path.join(past_dir, '{}.angl'.format(node_info['digest_name'])), 'Error: ',
      e)
00663             exit(-10)
00664         # else:
00665             # angl_sum_from_prev = ang_dist(cur_angles, np.fromfile(os.path.join(extra_past, '{}.angl'.format(node_info['digest_name'])),
      dtype=np.float32))
00666         angl_sum_to_goal = ang_dist(cur_angles, goal_angles)
00667         angl_sum_tot = node_info['ANGL_dist_total'] + angl_sum_from_prev
00668 
00669         # *********  AND_H ************
00670         goal_cont_dist_and_h = goal_contacts_and_h_sum - [np.logical_and(arr_elem, goal_cont_h).sum() for arr_elem in contacts_h]
00671         prev_cont_dist_and_h_1 = [np.logical_xor(arr_elem, prev_contacts_h).sum() for arr_elem in contacts_h]
00672         # prev_cont_dist_and_h_2 = [arr_elem.sum() for arr_elem in contacts_h] + prev_contacts_h.sum()
00673         # prev_cont_dist_and_h_2 = prev_cont_dist_and_h_2 / 2 - \
00674         #    [elem.sum() for elem in [np.logical_and(arr_elem, prev_contacts_h) for arr_elem in contacts_h]]
00675         total_cont_dist_and_h = node_info['AND_H_dist_total'] + prev_cont_dist_and_h_1
00676 
00677         # *********  AND ************
00678         goal_cont_dist_and = goal_contacts_and_sum - [np.logical_and(arr_elem, goal_contacts).sum() for arr_elem in contacts]
00679         prev_cont_dist_and_1 = [np.logical_xor(arr_elem, prev_contacts).sum() for arr_elem in contacts]
00680         # prev_cont_dist_and_2 = [arr_elem.sum() for arr_elem in contacts] + prev_contacts.sum()
00681         # prev_cont_dist_and_2 = prev_cont_dist_and_2 / 2 - \
00682         #                        [elem.sum() for elem in [np.logical_and(arr_elem, prev_contacts) for arr_elem in contacts]]
00683         total_cont_dist_and = node_info['AND_dist_total'] + prev_cont_dist_and_1
00684 
00685         # *********  XOR ************
00686         goal_cont_dist_sum_xor = [np.logical_xor(arr_elem, goal_contacts).sum() for arr_elem in contacts]
00687         # prev_cont_dist_sum_xor = [np.logical_xor(arr_elem, prev_contacts).sum() for arr_elem in contacts]
00688         prev_cont_dist_sum_xor = prev_cont_dist_and_1  # it is the same, no need to compute twice
00689         total_cont_dist_xor = node_info['XOR_dist_total'] + prev_cont_dist_sum_xor
00690 
00691         # # END PAR
00692         # pid.join()
00693         # angl_sum_to_goal, angl_sum_from_prev, angl_sum_tot, cur_angles = q.get()
00694 
00695         # store all metrics
00696         for i in range(tot_seeds):
00697             new_nodes[i] = dict()
00698             new_nodes[i]['digest_name'] = get_digest(new_nodes_names[i])
00699 
00700             new_nodes[i]['RMSD_to_goal'] = np.float32(rmsd_to_goal[i])
00701             new_nodes[i]['RMSD_from_prev'] = np.float32(from_prev_dist[i])
00702             new_nodes[i]['RMSD_dist_total'] = np.float32(rmsd_total_trav[i])
00703 
00704             new_nodes[i]['ANGL_to_goal'] = np.float32(angl_sum_to_goal[i])
00705             new_nodes[i]['ANGL_from_prev'] = np.float32(angl_sum_from_prev[i])
00706             new_nodes[i]['ANGL_dist_total'] = np.float32(angl_sum_tot[i])
00707 
00708             new_nodes[i]['AND_H_to_goal'] = np.int32(goal_cont_dist_and_h[i])
00709             new_nodes[i]['AND_H_from_prev'] = np.int32(prev_cont_dist_and_h_1[i])
00710             new_nodes[i]['AND_H_dist_total'] = np.int32(total_cont_dist_and_h[i])
00711 
00712             new_nodes[i]['AND_to_goal'] = np.int32(goal_cont_dist_and[i])
```

```
00713             new_nodes[i]['AND_from_prev'] = np.int32(prev_cont_dist_and_1[i])
00714             new_nodes[i]['AND_dist_total'] = np.int32(total_cont_dist_and[i])
00715 
00716             new_nodes[i]['XOR_to_goal'] = np.int32(goal_cont_dist_sum_xor[i])
00717             new_nodes[i]['XOR_from_prev'] = np.int32(prev_cont_dist_sum_xor[i])
00718             new_nodes[i]['XOR_dist_total'] = np.int32(total_cont_dist_xor[i])
00719 
00720             new_nodes[i]['native_name'] = zlib.compress(new_nodes_names[i].encode(), 9)
00721             # new_nodes[i]['contacts'] = csc_matrix(contacts[i])  # csc is the most efficient for contacts data, I tested it.
00722             # new_nodes[i]['angles']   = cur_angles[i].astype('float32')
00723 
00724             if not reusing_old_cont:
00725                 save_npz((os.path.join(past_dir, '{}.cont'.format(new_nodes[i]['digest_name']))), csc_matrix(contacts[i]), compressed=True)
00726 
00727             if not reusing_old_angl:
00728                 cur_angles[i].astype('float32').tofile(os.path.join(past_dir, '{}.angl'.format(new_nodes[i]['digest_name'])))
00729 
00730         if cur_metric == 0:
00731             return new_nodes, rmsd_to_goal, from_prev_dist, rmsd_total_trav
00732         elif cur_metric == 1:
00733             return new_nodes, angl_sum_to_goal, angl_sum_from_prev, angl_sum_tot
00734         elif cur_metric == 2:
00735             # if not isinstance(goal_cont_dist_and_h, (list,)):
00736             #     raise Exception('AND_H_to_goal: ', goal_cont_dist_and_h)
00737             return new_nodes, list(goal_cont_dist_and_h), list(prev_cont_dist_and_h_1), list(total_cont_dist_and_h)
00738         elif cur_metric == 3:
00739             # if not isinstance(goal_cont_dist_and, (list,)):
00740             #     raise Exception('AND_to_goal: ', goal_cont_dist_and)
00741             return new_nodes, list(goal_cont_dist_and), list(prev_cont_dist_and_1), list(total_cont_dist_and)
00742         elif cur_metric == 4:
00743             # if not isinstance(goal_cont_dist_sum_xor, (list,)):
00744             #     raise Exception('XOR_to_goal: ', goal_cont_dist_sum_xor)
00745             return new_nodes, list(goal_cont_dist_sum_xor), list(prev_cont_dist_sum_xor), list(total_cont_dist_xor)
00746         else:
00747             raise Exception('Unknown metric')
00748     else:  # This version is outdated. Using one metric does not produce significant speedup
00749         if cur_metric == 0:  # RMSD
00750             dist_arr = get_knn_dist_mdsctk(combined_pg, temp_xtc_file, goal_prot_only)
00751             # TODO: fix rm files and check if other files has to be removed
00752             # rm_queue.put_nowait(combined_pg)
00753             # rm_queue.put_nowait(temp_xtc_file)
00754             # since combined_pg had two points we have to divide result into two arrays
00755             from_prev_dist = dist_arr[0::2]
00756             rmsd_to_goal = dist_arr[1::2]
00757             rmsd_total_trav = [node_info['RMSD_dist_total'] + elem for elem in from_prev_dist]
00758             for i in range(tot_seeds):
00759                 new_nodes[i]['RMSD_to_goal'] = rmsd_to_goal[i]
00760                 new_nodes[i]['RMSD_from_prev'] = from_prev_dist[i]
00761                 new_nodes[i]['RMSD_dist_total'] = rmsd_total_trav[i]
00762 
00763             return new_nodes, rmsd_to_goal, from_prev_dist, rmsd_total_trav
00764 
00765         elif cur_metric == 1:  # PhyPsi
00766             cur_angles = compute_phipsi_angles(angl_num, temp_xtc_file.split('.')[0], init_bb_ndx)
00767             angl_sum_from_prev = ang_dist(cur_angles, node_info['angles'])
00768             angl_sum_to_goal = ang_dist(cur_angles, goal_angles)
00769             angl_sum_tot = node_info['ANG_dist_total'] + angl_sum_from_prev
00770             for i in range(tot_seeds):
00771                 new_nodes[i]['ANGL_to_goal'] = angl_sum_to_goal[i]
00772                 new_nodes[i]['ANGL_from_prev'] = angl_sum_from_prev[i]
00773                 new_nodes[i]['ANGL_dist_total'] = angl_sum_tot[i]
00774                 new_nodes[i]['angles'] = cur_angles[i]
00775 
00776             return new_nodes, angl_sum_to_goal, angl_sum_from_prev, angl_sum_tot
00777 
00778         elif cur_metric == 2:  # AND_H
00779             contacts = get_native_contacts(init_prot_only, files_for_trjcat, ndx_file_init, goal_contacts,
00780                                            atom_num, cont_dist, np.logical_and, pool=cpu_pool)[1]
00781             # although it is possible to get h_contacts from the get_native_contacts, then I'll not be able to get pure contacts to store
00782             contacts_h = [np.logical_and(arr_elem, h_filter_init) for arr_elem in contacts]
00783             goal_cont_dist_and_h = [np.logical_and(arr_elem, goal_cont_h).sum() for arr_elem in contacts_h]
00784             prev_contacts_h = np.logical_and(prev_contacts.toarray(), h_filter_init)
00785             prev_cont_dist_and_h_1 = [np.logical_xor(arr_elem, prev_contacts_h).sum() for arr_elem in contacts_h]
00786             prev_cont_dist_and_h_2 = [arr_elem.sum() for arr_elem in contacts_h] + prev_contacts_h.sum()
00787             prev_cont_dist_and_h_2 = prev_cont_dist_and_h_2 / 2 - \
00788                 [elem.sum() for elem in [np.logical_and(arr_elem, prev_contacts_h) for arr_elem in contacts_h]]
00789             total_cont_dist_and_h = node_info['AND_H_dist_total'] + prev_cont_dist_and_h_1
00790             for i in range(tot_seeds):
00791                 new_nodes[i]['AND_H_to_goal'] = goal_cont_dist_and_h[i]
00792                 new_nodes[i]['AND_H_from_prev'] = prev_cont_dist_and_h_1[i]
00793                 new_nodes[i]['AND_H_dist_total'] = total_cont_dist_and_h[i]
```

```
00794                 new_nodes[i]['contacts'] = csc_matrix(contacts[i])
00795
00796             return new_nodes, goal_cont_dist_and_h, prev_cont_dist_and_h_1, total_cont_dist_and_h
00797
00798         elif cur_metric == 3:  # AND
00799             goal_cont_dist_and, contacts = get_native_contacts(init_prot_only, files_for_trjcat, ndx_file_init, goal_contacts,
00800                                                                atom_num, cont_dist, np.logical_and, pool=cpu_pool)
00801             prev_cont_dist_and_1 = [np.logical_xor(arr_elem, prev_contacts.toarray()).sum() for arr_elem in contacts]
00802             prev_cont_dist_and_2 = [arr_elem.sum() for arr_elem in contacts] + prev_contacts.sum()
00803             prev_cont_dist_and_2 = prev_cont_dist_and_2 / 2 - \
00804                 [elem.sum() for elem in [np.logical_and(arr_elem, prev_contacts.toarray()) for arr_elem in contacts]]
00805             total_cont_dist_and = node_info['AND_dist_total'] + prev_cont_dist_and_1
00806             for i in range(tot_seeds):
00807                 new_nodes[i]['AND_to_goal'] = goal_cont_dist_and[i]
00808                 new_nodes[i]['AND_from_prev'] = prev_cont_dist_and_1[i]
00809                 new_nodes[i]['AND_dist_total'] = total_cont_dist_and[i]
00810                 new_nodes[i]['contacts'] = csc_matrix(contacts[i])
00811
00812             return new_nodes, goal_cont_dist_and, prev_cont_dist_and_1, total_cont_dist_and
00813
00814         elif cur_metric == 4:  # XOR
00815             goal_cont_dist_xor, contacts = get_native_contacts(init_prot_only, files_for_trjcat, ndx_file_init, goal_contacts,
00816                                                                atom_num, cont_dist, np.logical_xor, pool=cpu_pool)
00817             prev_cont_dist_sum_xor = [np.logical_xor(arr_elem, prev_contacts.toarray()).sum() for arr_elem in contacts]
00818             total_cont_dist_xor = node_info['XOR_dist_total'] + prev_cont_dist_sum_xor
00819             for i in range(tot_seeds):
00820                 new_nodes[i]['XOR_to_goal'] = goal_cont_dist_xor[i]
00821                 new_nodes[i]['XOR_from_prev'] = prev_cont_dist_sum_xor[i]
00822                 new_nodes[i]['XOR_dist_total'] = total_cont_dist_xor[i]
00823                 new_nodes[i]['contacts'] = csc_matrix(contacts[i])
00824
00825             return new_nodes, goal_cont_dist_xor, prev_cont_dist_sum_xor, total_cont_dist_xor
00826
00827     raise Exception("You cant be here")
00828
00829
00830 def compute_init_metric(past_dir: str, tot_seeds: int, init_xtc: str, goal_xtc: str, goal_prot_only: str, angl_num: int,
00831                         init_bb_ndx: str, goal_angles: np.ndarray, init_prot_only: str, ndx_file_init: str,
00832                         goal_cont_h: np.ndarray, atom_num: int, cont_dist: float, h_filter_init: np.ndarray,
00833                         goal_contacts: np.ndarray, goal_contacts_and_h_sum: np.int64, goal_contacts_and_sum: np.int64) -> list:
00834     """Special case of the "compute_metric"
00835
00836     Computes metric distances to the goal (NMR) conformation and sets previous distances to 0
00837
00838     Args:
00839         :param str past_dir: path to the directory with prior computation results
00840         :param int tot_seeds: total number of seed in the current run
00841         :param str init_xtc: initial (unfolded) conformation with water and salt
00842         :paramstr  goal_xtc: NMR (folded) conformation with water and salt
00843         :param str goal_prot_only: NMR (folded) conformation without water and salt (protein only)
00844         :param int angl_num: number of dihedral angles in the protein
00845         :param str init_bb_ndx: index file with backbone atom positions for the initial conformation
00846         :param np.ndarray goal_angles: angle values of the NMR structure
00847         :param str init_prot_only: initial (unfolded) conformation without water and salt (protein only)
00848         :param str ndx_file_init: index file with backbone atom positions for the NMR conformation
00849         :param np.ndarray goal_cont_h: contact values of the NMR structure (hydrogens only)
00850         :param int atom_num: total number of atoms in the protein (same for folded and unfolded)
00851         :param float cont_dist: distance between atoms treated as 'contact'
00852         :param np.ndarray h_filter_init: positions of the hydrogen atoms in the initial (unfolded) conformation
00853         :param np.ndarray goal_contacts: list of correct contacts in the NMR (folded) conformation
00854         :param np.int64 goal_contacts_and_h_sum: total sum of the contacts between hydrogents in the NMR (folded) conformation
00855         :param np.int64 goal_contacts_and_sum: total sum of the contacts in the NMR (folded) conformation
00856
00857     Returns:
00858         :return: node structure with the initial metrics
00859         :rtype: list
00860     """
00861     init_node = [None] * tot_seeds
00862     dim = 1 if tot_seeds > 1 else 0
00863     # *********  RMSD ************
00864     rmsd_to_goal = get_knn_dist_mdsctk(init_xtc, goal_xtc, goal_prot_only)
00865     # *********  ANG ************
00866     cur_angles = compute_phipsi_angles(angl_num, init_xtc.split('.')[0], init_bb_ndx)
00867
00868     angl_sum_to_goal = ang_dist(cur_angles, goal_angles)
00869
00870     contacts = get_native_contacts(init_prot_only, [init_xtc], ndx_file_init, None, atom_num, cont_dist, None, just_contacts=True)[1]
00871     # print(init_prot_only, init_xtc, ndx_file_init, atom_num, cont_dist)
00872     #  Cont prep
00873     contacts_h = np.logical_and(contacts, h_filter_init)
00874     # *********  AND_H ************
```

```
00875     goal_cont_dist_and_h = goal_contacts_and_h_sum - np.logical_and(contacts_h, goal_cont_h).sum(axis=dim)
00876     # *********  AND ************
00877     goal_cont_dist_and = goal_contacts_and_sum - np.logical_and(contacts, goal_contacts).sum(axis=dim)
00878     # *********  XOR ************
00879     goal_cont_dist_sum_xor = np.logical_xor(contacts, goal_contacts).sum(axis=dim)
00880
00881     if dim == 0:
00882         contacts = [contacts]
00883         # contacts_h = [contacts_h]
00884         angl_sum_to_goal = [angl_sum_to_goal]
00885         goal_cont_dist_and_h = [goal_cont_dist_and_h]
00886         goal_cont_dist_and = [goal_cont_dist_and]
00887         goal_cont_dist_sum_xor = [goal_cont_dist_sum_xor]
00888
00889     # store all metrics
00890     for i in range(tot_seeds):
00891         init_node[i] = dict()
00892         init_node[i]['digest_name'] = get_digest('s')
00893
00894         init_node[i]['RMSD_to_goal'] = np.float32(rmsd_to_goal[i])
00895         init_node[i]['RMSD_from_prev'] = np.uint32(0)
00896         init_node[i]['RMSD_dist_total'] = np.uint32(0)
00897
00898         init_node[i]['ANGL_to_goal'] = np.float32(angl_sum_to_goal[i])
00899         init_node[i]['ANGL_from_prev'] = np.uint32(0)
00900         init_node[i]['ANGL_dist_total'] = np.uint32(0)
00901
00902         init_node[i]['AND_H_to_goal'] = np.uint32(goal_cont_dist_and_h[i])
00903         init_node[i]['AND_H_from_prev'] = np.uint32(0)
00904         init_node[i]['AND_H_dist_total'] = np.uint32(0)
00905
00906         init_node[i]['AND_to_goal'] = np.uint32(goal_cont_dist_and[i])
00907         init_node[i]['AND_from_prev'] = np.uint32(0)
00908         init_node[i]['AND_dist_total'] = np.uint32(0)
00909
00910         init_node[i]['XOR_to_goal'] = np.uint32(goal_cont_dist_sum_xor[i])
00911         init_node[i]['XOR_from_prev'] = np.uint32(0)
00912         init_node[i]['XOR_dist_total'] = np.uint32(0)
00913         # init_node[i]['contacts'] = csc_matrix(contacts[i])
00914         save_npz(os.path.join(past_dir, '{}.cont'.format(init_node[i]['digest_name'])),
00915                  csc_matrix(contacts[i]), compressed=True)
00916
00917         init_node[i]['native_name'] = zlib.compress('s'.encode(), 9)
00918
00919         # init_node[i]['angles'] = cur_angles[i]
00920         cur_angles.astype('float32').tofile(os.path.join(past_dir, '{}.angl'.format(init_node[i]['digest_name'])))
00921
00922     if len(init_node) == 1:
00923         return init_node[0]
00924     return init_node
00925
00926
00927 def select_metrics_by_snr(cur_nodes: list, prev_node: dict, metric_names: list, tol_error: dict,
00928                           compute_all_at_once: bool, alowed_metrics: list, cur_metr: str) -> str:
00929     """SNR approach to a metric selection.
00930
00931     Metrics that had the highest SNR ratio (metric distance from the prev point)/(ambient noise) is selected next
00932     However, this approach does not always work and while you may a high SNR with contacts, there may be no real decrease in the rmsd.
00933     It is affected by the previous point performance.
00934
00935     Args:
00936         :param list cur_nodes: recent nodes
00937         :param dict prev_node: previous node
00938         :param list metric_names: list of metrics implemented (I want to know whole statistics, not only allowed metrics)
00939         :param dict tol_error: dict with noise data
00940         :param bool compute_all_at_once: toggle left as a reminder to not implement all at once
00941         :param list alowed_metrics: list of metrics that we allow to be used during the current run
00942         :param str cur_metr: name of the current metric
00943
00944     Returns:
00945         :return: metric name with the highest SNR
00946     """
00947     if not compute_all_at_once:
00948         # easy to implement, but I do not have plans to use it since 'all at once' is very fast
00949         # just take last node and compute all metrics
00950         raise Exception('Not implemented')
00951
00952     snr = False
00953     if snr:  # SNR approach may be biased. Additionally, prev_node should be computed here as prev point in name: s_1 is prev to s_1_3
00954         signal = dict()
00955         best_metr = metric_names[0]
```

```
00956         best_val = -1
00957         for metr in metric_names:
00958             cur_name = '{}_to_goal'.format(metr)
00959             signal[metr] = 0
00960             for i in range(len(cur_nodes)):
00961                 signal[metr] += (cur_nodes[i][cur_name] - prev_node[cur_name]) / tol_error[metr]
00962             if metric_names != metric_names[0] and signal[metr] > best_val and metr in alowed_metrics:
00963                 best_val = signal[metr]
00964                 best_metr = metr
00965 
00966         if best_metr == cur_metr:
00967             print('New metric is the same as previous. Switching to next metric')
00968             while len(metric_names) > 1 and (best_metr == cur_metr or best_metr not in alowed_metrics):
00969                 best_metr = metric_names[(metric_names.index(best_metr) + 1) % len(metric_names)]
00970 
00971         print('SNR for metrics:')
00972         for metr in metric_names:
00973             if metr == best_metr:
00974                 print(' >*{}: {}'.format(metr, signal[metr]))
00975             elif best_val == signal[metr]:
00976                 print('  +{}: {}'.format(metr, signal[metr]))
00977             elif metr not in alowed_metrics:
00978                 print('   {}: {} # ignored'.format(metr, signal[metr]))
00979             else:
00980                 print('   {}: {}'.format(metr, signal[metr]))
00981     else:  # use round-robin
00982         best_metr = metric_names[(metric_names.index(cur_metr) + 1) % len(metric_names)]
00983         while best_metr not in alowed_metrics:
00984             print('Skipping {} since it is not in allowed list'.format(best_metr))
00985             best_metr = metric_names[(metric_names.index(cur_metr) + 1) % len(metric_names)]
00986         print('Switching to {}'.format(best_metr))
00987 
00988     return best_metr
```

4.33 parse_topology_for_hydrogens.py File Reference

Namespaces

- parse_topology_for_hydrogens

Functions

- list parse_topology_for_hydrogens.parse_top_for_h (str top_filename)

  Reads the topology file and finds positions of the hydrogen atoms.

4.34 parse_topology_for_hydrogens.py

```
00001 def parse_top_for_h(top_filename: str) -> list:
00002     """Reads the topology file and finds positions of the hydrogen atoms
00003 
00004     Args:
00005         :param top_filename: topology file .top
00006 
00007     Returns:
00008         :return: list of hydrogen atoms position
00009         :rtype: list
00010     """
00011     good_ind = list()
00012     with open(top_filename, 'r') as f:
00013         line = f.readline()
00014         while '[ atoms ]' not in line:
00015             line = f.readline()
00016         line = f.readline()
00017         atom_ind = line[1:].strip().split().index('atom')
00018         while ';' == line[0]:
00019             line = f.readline()
00020         line = line.strip()
00021         while len(line):
00022             if line[0] != ';':
00023                 parsed_line = line.split(';')[0].split()
00024                 if parsed_line[atom_ind][0] == 'H':
00025                     good_ind.append(int(parsed_line[0]))
00026                     # good_ind.append(int(parsed_line[0]) - 1)  # -1 for corr indexing
00027             line = f.readline().strip()
00028     return good_ind
00029 
00030 
00031 # parse_top_for_h('./prot_dir/topol.top')
```

4.35 plot_energy.py File Reference

Namespaces

- plot_energy

Functions

- def plot_energy.main ()

4.36 plot_energy.py

```
00001 #!/usr/bin/env python3
00002
00003 import sqlite3 as lite
00004 import os
00005 # import matplotlib.pyplot as plt
00006 # import scipy
00007 # from scipy.optimize import curve_fit
00008 # import numpy as np
00009 # from matplotlib.ticker import NullFormatter  # useful for 'logit' scale
00010 # from matplotlib import gridspec
00011 # from PIL import Image
00012 # from matplotlib import figure
00013 # from matplotlib.figure import figaspect
00014 from gmx_wrappers import gmx_eneconv, gmx_energy
00015
00016 def main():
00017     past_dir = './past'
00018     db_to_connect = 'results_12'
00019     polynomial = False
00020     font = {'family': 'serif',
00021             'color': 'darkred',
00022             'weight': 'normal',
00023             'size': 16,
00024             }
00025     if not os.path.exists(db_to_connect + '.sqlite3'):
00026         raise Exception('DB not found')
00027
00028     con = lite.connect(db_to_connect + '.sqlite3', check_same_thread=False, isolation_level=None)
00029     cur = con.cursor()
00030
00031     qry = "select a.name, a.hashed_name from main_storage a  where a.goal_dist= ( select min(b.goal_dist) from main_storage b)"
00032     result = cur.execute(qry)
00033     all_res = result.fetchone()
00034     name = all_res[0]
00035     spname = name.split('_')
00036     all_prev_names = ['\'{}\"'.format('_'.join(spname[:i])) for i in range(1, len(spname))]
00037     long_line = ", ".join(all_prev_names)
00038
00039     qry = "select name, hashed_name from main_storage where name in ({})".format(long_line)
00040     result = cur.execute(qry)
00041     _ = result.fetchone()
00042     all_res = result.fetchall()
00043     names, hashed_names = zip(*all_res)
00044     wave = 100
00045     tot_chunks = int((len(hashed_names) + 1) / wave)
00046     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00047     gmx_eneconv(f=[os.path.join("./past", hashed_name) + '.edr' for hashed_name in hashed_names[:wave]], o='./combinded_energy.edr')
00048     for i in range(wave, len(hashed_names) + 1 - wave, wave):
00049         os.rename('./combinded_energy.edr', './combinded_energy_prev.edr')
00050         gmx_eneconv(f=["./combinded_energy_prev.edr"] + [os.path.join("./past", hashed_name + '.edr') for hashed_name in hashed_names[i:i +
      wave if i + wave < len(hashed_names) else -1]],
00051                     o='./combinded_energy.edr')
00052         if int(i / wave) % 10 == 0:
00053             print('{}/{} ({:.1f}%)'.format(int(i / wave), tot_chunks, 100 * int(i / wave) / tot_chunks))
00054
00055     os.rename('./combinded_energy.edr', './combinded_energy_best.edr')
00056     print('Done with best')
00057
00058
00059
00060     qry = "select a.name, a.hashed_name from main_storage a "
00061     result = cur.execute(qry)
00062     _ = result.fetchone()
00063     all_res = result.fetchall()
00064     names, hashed_names = zip(*all_res)
00065
00066     # gmx_eneconv(f=[os.path.join(past_dir, hash_name+'.edr') for hash_name in hashed_names], o='./combinded_energy.edr')
00067
00068     wave = 100
```

```
00069     tot_chunks = int((len(hashed_names)+1)/wave)
00070     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00071     gmx_eneconv(f=[os.path.join("./past", hashed_name)+'.edr' for hashed_name in hashed_names[:wave]], o='./combinded_energy.edr')
00072     for i in range(wave, len(hashed_names)+1-wave, wave):
00073         os.rename('./combinded_energy.edr', './combinded_energy_prev.edr')
00074         gmx_eneconv(f=["./combinded_energy_prev.edr"] +[os.path.join("./past", hashed_name + '.edr') for hashed_name in hashed_names[i:i+wave
      if i+wave < len(hashed_names) else -1]], o='./combinded_energy.edr')
00075         if int(i/wave) % 10 == 0:
00076             print('{}/{} ({:.1f}%)'.format(int(i/wave), tot_chunks, 100*int(i/wave)/tot_chunks))
00077
00078     os.rename('./combinded_energy.edr', './combinded_energy_all_main.edr')
00079     print('Done with all main')
00080
00081
00082     qry = "select a.name, a.hashed_name from main_storage a join log b on a.id=b.id where b.dst='VIZ' order by b.timestamp"
00083     result = cur.execute(qry)
00084     _ = result.fetchone()
00085     all_res = result.fetchall()
00086     names, hashed_names = zip(*all_res)
00087
00088     wave = 100
00089     tot_chunks = int((len(hashed_names)+1)/wave)
00090     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00091     gmx_eneconv(f=[os.path.join("./past", hashed_name)+'.edr' for hashed_name in hashed_names[:wave]], o='./combinded_energy.edr')
00092     for i in range(wave, len(hashed_names)+1-wave, wave):
00093         os.rename('./combinded_energy.edr', './combinded_energy_prev.edr')
00094         gmx_eneconv(f=["./combinded_energy_prev.edr"] +[os.path.join("./past", hashed_name + '.edr') for hashed_name in hashed_names[i:i+wave
      if i+wave < len(hashed_names) else -1]], o='./combinded_energy.edr')
00095         if int(i/wave) % 10 == 0:
00096             print('{}/{} ({:.1f}%)'.format(int(i/wave), tot_chunks, 100*int(i/wave)/tot_chunks))
00097
00098     os.rename('./combinded_energy.edr', './combinded_energy_all_viz.edr')
00099     print('Done with viz')
00100
00101
00102     # gmx_energy('./combinded_energy.edr', './combinded_energy.xvg', fee=True, fetemp=300)
00103
00104
00105
00106 if __name__ == '__main__':
00107     main()
```

4.37 plot_matplot_energy.py File Reference

Namespaces

- plot_matplot_energy

Functions

- def plot_matplot_energy.main ()
- int plot_matplot_energy.single_plot (int fig_num, dict ax_prop, list arr_A, list arr_B, list filenames_db, str marker, float mark_size, bool bsf, bool rev, bool shrink, str xlab, str ylab, str title, str filename, list extra_line=None, int mdpi=400)

4.38 plot_matplot_energy.py

```
00001 #!/usr/bin/env python3
00002 import numpy as np
00003 import os
00004 import matplotlib.pyplot as plt
00005 import numpy as np
00006 from matplotlib.figure import figaspect
00007
00008
00009 def main():
00010     filenames_found = [f.split("/")[-1] for f in os.listdir('./') if '.npy' in f]
00011     fig_num = 0
00012     for file in filenames_found:
00013         cur_arr = np.load(file)
00014         cur_arr = cur_arr.swapaxes(0, 1)
00015         new_name = file.split('.')[0]
00016         ax_prop = {"min_lim_x": min(cur_arr[0]), "max_lim_x": max(cur_arr[0]) + max(cur_arr[0]) / 80, "min_lim_y": min(cur_arr[1]),
      "max_lim_y": max(cur_arr[1]) - max(cur_arr[1]) / 80,
00017                    "min_ax_x": 0, "max_ax_x": max(cur_arr[0]) + max(cur_arr[0]) / 80, "min_ax_y": min(cur_arr[1]) + min(cur_arr[1]) / 80,
      "max_ax_y": max(cur_arr[1]) - max(cur_arr[1]) / 80,
00018                    "ax_step_x": (max(cur_arr[0]) - 0) / 16,
00019                    "ax_step_y": (max(cur_arr[1]) - min(cur_arr[1])) / 20}
00020         extra_line = [{"ax_type": 'ver', "val": 0, "name": "simulation origin", "col": "darkmagenta"}]
00021         fig_num = single_plot(fig_num, ax_prop, [cur_arr[0]], [cur_arr[1]], ['LJ interaction value'], '-', 1.0, True, True, False, 'Time, ps',
      'LJ-SR, kJ/mol', 'Lennard-Jones Short Range Protein-Protein Interaction', new_name, extra_line=extra_line)
```

```
00022         plt.close('all')
00023
00024
00025 def single_plot(fig_num: int, ax_prop: dict, arr_A: list, arr_B: list, filenames_db: list, marker: str, mark_size: float,
00026                 bsf: bool, rev: bool, shrink: bool, xlab: str, ylab: str,
00027                 title: str, filename: str, extra_line: list = None, mdpi: int = 400) -> int:
00028     """
00029
00030     Args:
00031         :param int fig_num:
00032         :param dict ax_prop:
00033         :param list arr_A:
00034         :param list arr_B:
00035         :param list filenames_db:
00036         :param str marker:
00037         :param float mark_size:
00038         :param bool bsf:
00039         :param bool rev:
00040         :param bool shrink:
00041         :param str xlab:
00042         :param str ylab:
00043         :param str title:
00044         :param str filename:
00045         :param list extra_line:
00046         :param int mdpi:
00047
00048     Returns:
00049         :return: last figure number.
00050         :rtype: int
00051     """
00052     fig_num += 1
00053
00054     w, h = figaspect(0.5)
00055     fig = plt.figure(fig_num, figsize=(w, h))
00056
00057     ax = fig.gca()
00058     plt.xlim(ax_prop["min_lim_x"], ax_prop["max_lim_x"])
00059     plt.ylim(ax_prop["min_lim_y"], ax_prop["max_lim_y"])
00060
00061     major_xticks = np.arange(ax_prop["min_ax_x"], ax_prop["max_ax_x"], ax_prop["ax_step_x"])
00062     major_yticks = np.arange(ax_prop["min_ax_y"], ax_prop["max_ax_y"], ax_prop["ax_step_y"])
00063
00064     if major_xticks is not None:
00065         ax.set_xticks(major_xticks)
00066     if major_yticks is not None:
00067         ax.set_yticks(major_yticks)
00068     # if minor_xticks is not None:
00069     #     ax.set_xticks(minor_xticks, minor=True)
00070     # if minor_yticks is not None:
00071     #     ax.set_yticks(minor_yticks, minor=True)
00072
00073     plt.grid(which='both')
00074     plt.xticks(rotation=30)
00075     plt.subplots_adjust(top=0.95, bottom=0.14, left=0.09, right=0.98)
00076
00077     lines_b = []
00078     for i, bsf_trav_to_goal in enumerate(arr_A):
00079         if not shrink:  # use provided array arr_B
00080             if rev:
00081                 line_b, = plt.plot(arr_A[i], arr_B[i], marker, markersize=mark_size)
00082             else:
00083                 line_b, = plt.plot(arr_B[i], arr_A[i], marker, markersize=mark_size)
00084         else:  # generate array from 0 to len(arr_A)
00085             if rev:
00086                 if bsf:
00087                     line_b, = plt.plot(arr_A[i], range(len(arr_A[i])), marker, markersize=mark_size)
00088                 else:
00089                     line_b, = plt.plot(arr_A[i], arr_B[i], marker, markersize=mark_size)
00090             else:
00091                 line_b, = plt.plot(range(len(arr_A[i])), arr_A[i], marker, markersize=mark_size)
00092         lines_b.append(line_b)
00093
00094     if extra_line is not None:
00095         for el in extra_line:
00096             if el["ax_type"] == 'ver':
00097                 straight_line = plt.axvline(x=el["val"], color=el["col"], linestyle='-')  #
00098             elif el["ax_type"] == 'hor':
00099                 straight_line = plt.axhline(y=el["val"], color=el["col"], linestyle='-')
00100             else:
00101                 raise Exception('Wrong ax type')
00102             lines_b.append(straight_line)
```

```
00103             filenames_db.append(el["name"])
00104         # if el["ax_type"] == 'ver':
00105         #     if not rev:
00106         #         ax.annotate('folding direction', xytext=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
      ax_prop["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 5 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 * ax_prop["ax_step_y"]),
      arrowprops={'arrowstyle': '->', 'lw': 1.5, 'color': 'mediumblue'}, va='center')  # -->
00107         #     else:
00108         #         ax.annotate('folding direction', xytext=(ax_prop["max_ax_x"] - 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
      ax_prop["ax_step_y"]), xy=(ax_prop["max_ax_x"] - 5 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 * ax_prop["ax_step_y"]),
      arrowprops={'arrowstyle': '->', 'lw': 1.5, 'color': 'mediumblue'}, va='center')  # -->
00109         # else:
00110         #     if not rev:
00111         #         ax.annotate('folding direction', xytext=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
      ax_prop["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 4 * ax_prop["ax_step_y"]),
      arrowprops={'arrowstyle': '->', 'lw': 1.5, 'color': 'mediumblue'}, ha='center')  # <--
00112         #     else:
00113         #         pass # does not exist
00114               # ax.annotate('folding direction', xytext=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 1 *
      ax_prop["ax_step_y"]), xy=(ax_prop["min_ax_x"] + 1 * ax_prop["ax_step_x"], ax_prop["max_lim_y"] - 4 * ax_prop["ax_step_y"]),
      arrowprops={'arrowstyle': '->', 'lw': 1.5, 'color': 'mediumblue'}, ha='center')  # -->
00115
00116     ax.legend(lines_b, filenames_db)
00117     plt.xlabel(xlab)
00118     plt.ylabel(ylab)
00119     plt.title(title)
00120     try:
00121         plt.savefig(filename, dpi=mdpi)
00122     except:
00123         plt.show()
00124     plt.close('all')
00125     return fig_num
00126
00127
00128 if __name__ == '__main__':
00129     main()
```

4.39 print_best_frame.py File Reference

Namespaces

- print_best_frame

Functions

- def print_best_frame.main ()

4.40 print_best_frame.py

```
00001 #!/usr/bin/env python3
00002
00003 import sqlite3 as lite
00004 import os
00005 import sys
00006 from gmx_wrappers import gmx_trjcat
00007
00008
00009 def main():
00010     if len(sys.argv) < 2:
00011         raise Exception('Not enough arguments')
00012     # db_to_connect = 'results_12'
00013     db_to_connect = sys.argv[1]
00014     past_dir = './past'
00015     if not os.path.exists(db_to_connect + '.sqlite3'):
00016         raise Exception('DB not found')
00017
00018     con = lite.connect(db_to_connect + '.sqlite3', check_same_thread=False, isolation_level=None)
00019     cur = con.cursor()
00020
00021     qry = "select a.name, a.hashed_name from main_storage a  where a.goal_dist= ( select min(b.goal_dist) from main_storage b)"
00022     result = cur.execute(qry)
00023     all_res = result.fetchone()
00024     name = all_res[0]
00025     spname = name.split('_')
00026     all_prev_names = ['\'{}\"'.format('_'.join(spname[:i])) for i in range(1, len(spname))]
00027     long_line = ", ".join(all_prev_names)
00028
00029     qry = "select name, hashed_name from main_storage where name in ({})".format(long_line)
00030     result = cur.execute(qry)
00031     all_res = result.fetchall()
00032     names, hashed_names = zip(*all_res)
```

```
00033     wave = 100
00034     tot_chunks = int((len(hashed_names) + 1) / wave)
00035     print('wave={}, tot_chunks={}'.format(wave, tot_chunks))
00036     if os.path.exists('./combinded_traj.xtc'):
00037         os.remove('./combinded_traj.xtc')
00038     if os.path.exists('./combinded_traj_prev.xtc'):
00039         os.remove('./combinded_traj_prev.xtc')
00040 
00041     gmx_trjcat(f=[os.path.join(past_dir, hashed_name) + '.xtc' for hashed_name in hashed_names[:wave]], o='./combinded_traj.xtc',
       n='./prot_dir/prot.ndx', cat=True, vel=False, sort=False, overwrite=True)
00042     for i in range(wave, len(hashed_names), wave):
00043         os.rename('./combinded_traj.xtc', './combinded_traj_prev.xtc')
00044         gmx_trjcat(f=[" ./combinded_traj_prev.xtc "] + [os.path.join(past_dir, hashed_name) + '.xtc' for hashed_name in
       hashed_names[i:i+wave]], o='./combinded_traj.xtc', n='./prot_dir/prot.ndx', cat=True, vel=False, sort=False, overwrite=True)
00045         if int(i / wave) % 10 == 0:
00046             print('{}/{} ({:.1f}%)'.format(int(i / wave), tot_chunks, 100 * int(i / wave) / tot_chunks))
00047 
00048     if os.path.exists('./combinded_traj.xtc'):
00049         os.rename('./combinded_traj.xtc', './{}_traj_best.xtc'.format(db_to_connect))
00050     if os.path.exists('./combinded_traj_prev.xtc'):
00051         os.remove('./combinded_traj_prev.xtc')
00052     print('Done with best for {}'.format(db_to_connect))
00053 
00054 
00055 if __name__ == '__main__':
00056     main()
```

4.41 print_nat_cont.py File Reference

Namespaces

- print_nat_cont

Functions

- def print_nat_cont.main ()

4.42 print_nat_cont.py

```
00001 #!/bin/env python3
00002 import matplotlib.pyplot as plt
00003 from matplotlib.figure import figaspect
00004 import numpy as np
00005 from functools import reduce
00006 
00007 def main():
00008 
00009     # with open('output.dat', 'r') as infile:
00010     #     arr = infile.readlines()
00011     #
00012     # arr = [int(val.strip()) for val in arr]
00013     arr = np.load('nat_cont_300_1_9_AND_H.npz')
00014     arr = arr[arr.files[0]]
00015     avg = reduce(lambda a, b: a + b, arr) / len(arr)
00016     # arr = [elem for elem in arr if elem < avg*5]
00017     max_val = max(arr)
00018     min_val = min(arr)
00019 
00020 
00021     fig_num = 0
00022     mdpi = 400
00023     major_xticks = None
00024     minor_xticks = None
00025     major_yticks = None
00026     minor_yticks = None
00027     w, h = figaspect(0.5)
00028     fig = plt.figure(fig_num, figsize=(w, h))
00029     plt.xlim(0, len(arr))
00030     ax = fig.gca()
00031     major_xticks = np.arange(0, len(arr) + len(arr) / 10, len(arr) / 10)
00032     if max_val - min_val > 0:
00033         major_yticks = np.arange(min_val, max_val + max_val / 16, (max_val - min_val) / 16)
00034     if major_xticks is not None:
00035         ax.set_xticks(major_xticks)
00036     if minor_xticks is not None:
00037         ax.set_xticks(minor_xticks, minor=True)
00038     if major_yticks is not None:
00039         ax.set_yticks(major_yticks)
00040     if minor_yticks is not None:
00041         ax.set_yticks(minor_yticks, minor=True)
```

```
00042     plt.grid(which='both')
00043     lines = []
00044
00045     line, = plt.plot(range(len(arr)), arr, '-', markersize=1)
00046     lines.append(line)
00047     ax.legend(lines, 'full cont')
00048     plt.xlabel("frame")
00049     plt.ylabel("contacts AND goal")
00050     plt.title('nat Hydrogen contacts (AND) for 20ns gb1 simulation for 300K d=1.9 (higher is better)')
00051     plt.savefig('nat_cont_300_1_9_AND_H.png', dpi=mdpi)
00052
00053 main()
```

4.43 rebuild.py File Reference

Namespaces

- rebuild

Variables

- string rebuild.filename = 's_5_6_5_2_5_2_7_6_7_4_6_3_4_4_7_4_3_7_5_6_5_1_2_7_1_1_5_1_5_6_1_1_4_6_3_4_2_3_0_1_0_4_4_7_5_5_1_0_3_2_2_2_5_2_7_4_0_0_7_1_0_6_6_7_3_6_7_3_4_3_2_4_1_1_3_4_6_4_4_1_6_3_4_7_0_2_6_3_0_2_1_0_0_4_7_1_3_6_0_5_5_5_0_4_5_3_7_5_7_4_3_0_6.xtc'
- string rebuild.ext = filename.split('.')[1]
- string rebuild.arr = filename.split('.')[0].split('_')
- list rebuild.good_arr = [ ]
- string rebuild.cummulative = ''
- rebuild.f
- rebuild.o
- rebuild.n
- rebuild.cat
- rebuild.True
- rebuild.vel
- rebuild.False
- rebuild.sort
- rebuild.overwrite

4.44 rebuild.py

```
00001 #!/usr/bin/env python3
00002 from main import gmx_trjcat
00003 filename =
       's_5_6_5_2_5_2_7_6_7_4_6_3_4_4_7_4_3_7_5_6_5_1_2_7_1_1_5_1_5_6_1_1_4_6_3_4_2_3_0_1_0_4_4_7_5_5_1_0_3_2_2_2_5_2_7_4_0_0_7_1_0_6_6_7_3_6_7_3_4_3_2_4_1_1_3_4_6_4_4_1_6_3
00004 ext = filename.split('.')[1]
00005 arr = filename.split('.')[0].split('_')
00006 good_arr = []
00007 cummulative = ''
00008 for i, j in enumerate(arr):
00009     cummulative = '_'.join(arr[0:i+1])
00010     good_arr.append('./past/{}.{}'.format(cummulative, ext))
00011
00012 print(good_arr)
00013 gmx_trjcat(f=good_arr, o='./final_comb.xtc', n='./prot_dir/prot.ndx', cat=True, vel=False, sort=False, overwrite=True)
```

4.45 recompute_and.py File Reference

Namespaces

- recompute_and

Functions

- def recompute_and.main ()

4.46 recompute_and.py

```
00001 #!/bin/env python3
00002 import matplotlib.pyplot as plt
00003 from matplotlib.figure import figaspect
00004 import numpy as np
00005 from functools import reduce
00006 import math
00007 import multiprocessing as mp
00008 from parse_topology_for_hydrogens import parse_top_for_h
00009
00010 def main():
00011     cont_corr = np.load('cor_cont_300_1_9.npz')
```

```
00012     cont_corr = cont_corr[cont_corr.files[0]]
00013
00014     contacts = np.load('full_cont_300_1_9.npz')
00015     contacts = contacts[contacts.files[0]]
00016     print('Corr contacts count: {}'.format(np.sum(cont_corr)))
00017     compute_h_only = False
00018     if compute_h_only:
00019         h_pos = parse_top_for_h('./prot_dir/topol.top')
00020         num_atoms = int(math.sqrt(len(contacts[0])))
00021         h_filter = np.zeros(num_atoms * num_atoms, dtype=np.uint8)
00022         for pos in h_pos:
00023             h_filter[(pos-1)*num_atoms:pos*num_atoms] = 1
00024         cont_corr_h = np.logical_and(cont_corr, h_filter)
00025         cont_corr = cont_corr_h
00026     pool = mp.Pool(mp.cpu_count())
00027     nat_cont_arr = [pool.apply(np.logical_xor, args=(cont_arr, cont_corr)) for cont_arr in contacts]
00028     print('Done with and')
00029     nat_cont_arr = [pool.apply(np.sum, args=(elem,)) for elem in nat_cont_arr]
00030     np.savez('nat_cont_300_1_9_XOR.npz', nat_cont_arr)
00031
00032
00033 main()
```

4.47 test.py File Reference

Namespaces

- test

Functions

- def test.add_task (task, priority=0)
- def test.pop_task ()

Variables

- list test.pq = [ ]
- dict ionary test.entry_finder = {}
- string test.REMOVED = '<removed-task>'
- test.counter = itertools.count()

4.48 test.py

```
00001 import heapq
00002 import itertools
00003
00004 pq = []                         # list of entries arranged in a heap
00005 entry_finder = {}               # mapping of tasks to entries
00006 REMOVED = '<removed-task>'      # placeholder for a removed task
00007 counter = itertools.count()     # unique sequence count
00008
00009 def add_task(task, priority=0):
00010     'Add a new task or update the priority of an existing task'
00011     count = next(counter)
00012     entry = [priority, count, task]
00013     entry_finder[task] = entry
00014     heapq.heappush(pq, entry)
00015
00016
00017
00018 def pop_task():
00019     'Remove and return the lowest priority task. Raise KeyError if empty.'
00020     while pq:
00021         priority, count, task = heapq.heappop(pq)
00022         if task is not REMOVED:
00023             del entry_finder[task]
00024             return task
00025     raise KeyError('pop from an empty priority queue')
00026
00027 add_task('kva10', 10)
00028 add_task('kva12', 12)
00029 add_task('kva7', 7)
00030 add_task('kva10', 10)
00031 add_task('kva10', 10)
00032 add_task('kva10', 10)
00033 add_task('kva10', 10)
00034 add_task('kva10', 10)
00035 add_task('kva10', 10)
00036 add_task('kva10', 10)
00037 add_task('kva10', 10)
```

4.49 testll.py File Reference

Namespaces

- testll

Functions

- def testll.permute (word)
- def testll.permute_driver (word)
- def testll.main ()

4.50 testll.py

```
00001 def permute(word):
00002     if len(word) == 1: return [word]
00003     a = list()
00004     for i in range(len(word)):
00005         res = permute(word[0:i]+word[i+1:])
00006         for j in range(len(res)):
00007             res[j] = word[i] + res[j]
00008         a.extend(res)
00009     return a
00010 
00011 
00012 def permute_driver(word):
00013     a = list()
00014     for i in range(len(word)):
00015         res = permute(word[0:i]+word[i+1:])
00016         for j in range(len(res)):
00017             res[j] = word[i] + res[j]
00018         a.extend(res)
00019     print(len(a))
00020 
00021 def main():
00022     permute_driver('abcdefr')
00023 
00024 
00025 
00026 if __name__ == '__main__':
00027     main()
```

4.51 threaded_funcs.py File Reference

Namespaces

- threaded_funcs

Functions

- NoReturn threaded_funcs.print_async (str info_form_str, tuple tup)

  Test function used for async printing.
- NoReturn threaded_funcs.threaded_print (mp.JoinableQueue pipe)

  Prints statement provided from the pipe.
- NoReturn threaded_funcs.threaded_db_input (mp.JoinableQueue pipe, int len_seeds)

  Runs DB operation in a separate process.
- NoReturn threaded_funcs.threaded_copy (mp.JoinableQueue pipe)

  Recieves filenames (A, B) from the pipe and tries to copy A into B.
- NoReturn threaded_funcs.threaded_rm (mp.JoinableQueue pipe)

  Recieves filename from the pipe and tries to remove them.

4.52 threaded_funcs.py

```
00001 """This file contains functions executed in a separate process to reduce I/O.
00002 While I know that there is asyncio, but I believe that kernel can handle processes much better than Python.
00003 Additionally, you do not create context for a function during each call, but only once - during the initial call.
00004 
00005     :platform: linux
00006 
00007 .. moduleauthor:: Ivan Syzonenko <is2k@mtmail.mtsu.edu>
00008 """
00009 __license__ = "MIT"
00010 __docformat__ = 'reStructuredText'
00011 
00012 
```

```
00013 import multiprocessing as mp
00014 import os
00015 from shutil import copy2 as cp2
00016 from typing import NoReturn
00017 from db_proc import get_db_con
00018
00019
00020 def print_async(info_form_str: str, tup: tuple) -> NoReturn:
00021     """Test function used for async printing
00022
00023     Args:
00024         :param str info_form_str: formatting string.
00025         :param tuple tup: data to print.
00026
00027     Returns:
00028     Simply prints the string.
00029     """
00030     print(info_form_str.format(*tup))
00031
00032
00033 def threaded_print(pipe: mp.JoinableQueue) -> NoReturn:
00034     """Prints statement provided from the pipe.
00035
00036     Typically, you supply formating string and options
00037
00038     Args:
00039         :param mp.JoinableQueue pipe: source of the perforated strings and values (str, vals).
00040
00041     Returns:
00042     Simply prints the string.
00043     """
00044     stmt = pipe.get(timeout=3600)
00045     while stmt is not None:
00046         try:
00047             # with PRINT_LOCK:
00048             #     print(stmt[0].format(*stmt[1]))
00049             print(stmt[0].format(*stmt[1]))
00050         except Exception as e:
00051             print(e)
00052         finally:
00053             pipe.task_done()
00054             stmt = pipe.get()
00055     print('Print thread exiting...')
00056
00057
00058 def threaded_db_input(pipe: mp.JoinableQueue, len_seeds: int) -> NoReturn:
00059     """Runs DB operation in a separate process
00060
00061     Args:
00062         :param pipe: connection with the parent.
00063         :param len_seeds: total number of seeds.
00064
00065     Returns:
00066     Executes the queries from the queue.
00067     """
00068     con, dbname = get_db_con(len_seeds)
00069     stmt = pipe.get(timeout=3600)
00070     pid = None
00071     while stmt is not None:
00072         try:
00073             pid.join()
00074         except Exception as e:
00075             if pid:
00076                 print(e)
00077         # try:
00078         # con = con = lite.connect(dbname, timeout=3000, check_same_thread=False, isolation_level=None)
00079         # con.commit()
00080         pid = mp.Process(target=stmt[0], args=(con,)+stmt[1])
00081         pid.start()
00082         # except Exception as e:
00083         #     print('Found exception in db input:')
00084         #     print(e)
00085         #     print('Arguments that caused exception: ')
00086         #     print(stmt)
00087         # finally:
00088         pipe.task_done()
00089         stmt = pipe.get()
00090     print('DB thread exiting...')
00091     con.close()
00092
00093
```

```
00094 def threaded_copy(pipe: mp.JoinableQueue) -> NoReturn:
00095     """Recieves filenames (A, B) from the pipe and tries to copy A into B
00096
00097     Args:
00098         :param pipe: connection with the parent
00099
00100     Returns:
00101     Copies files in the background.
00102     """
00103     stmt = pipe.get(timeout=3600)
00104     while stmt is not None:
00105         # with COPY_LOCK:
00106         cp2(stmt[0], stmt[1])
00107         pipe.task_done()
00108         stmt = pipe.get(timeout=1800)
00109
00110
00111 def threaded_rm(pipe: mp.JoinableQueue) -> NoReturn:
00112     """Recieves filename from the pipe and tries to remove them
00113
00114     Args:
00115         :param pipe: connection with the parent
00116
00117     Returns:
00118     Removes files in the background.
00119     """
00120     stmt = pipe.get(timeout=3600)
00121     while stmt is not None:
00122         # with RM_LOCK:
00123         try:
00124             os.remove(stmt)
00125         except Exception as e:
00126             print('Was not able to remove {}, Error: {}'.format(stmt, e))
00127         pipe.task_done()
00128         stmt = pipe.get(timeout=1800)
```

Index