Better partitions of protein graphs for subsystem quantum chemistry

Moritz von Looz[1], Mario Wolter[2], Christoph R. Jacob[2], and Henning Meyerhenke[1]

[1] {moritz.looz-corswarem, meyerhenke}@kit.edu
Institute of Theoretical Informatics
Karlsruhe Institute of Technology (KIT), Germany
[2] {m.wolter, c.jacob}@tu-braunschweig.de
Institute of Physical and Theoretical Chemistry
TU Braunschweig, Germany

Abstract. Determining the interaction strength between proteins and small molecules is key to analyzing their biological function. Quantum-mechanical calculations such as *Density Functional Theory* (DFT) give accurate and theoretically well-founded results. With common implementations the running time of DFT calculations increases quadratically with molecule size. Thus, numerous subsystem-based approaches have been developed to accelerate quantum-chemical calculations. These approaches partition the protein into different fragments, which are treated separately. Interactions between different fragments are approximated and introduce inaccuracies in the calculated interaction energies.
To minimize these inaccuracies, we represent the amino acids and their interactions as a weighted graph in order to apply graph partitioning. None of the existing graph partitioning work can be directly used, though, due to the unique constraints in partitioning such protein graphs. We therefore present and evaluate several algorithms, partially building upon established concepts, but adapted to handle the new constraints. For the special case of partitioning a protein along the main chain, we also present an efficient dynamic programming algorithm that yields provably optimal results. In the general scenario our algorithms usually improve the previous approach significantly and take at most a few seconds.

1 Introduction

Context. The biological role of proteins is largely determined by their interactions with other proteins and small molecules. Quantum-chemical methods, such as *Density Functional Theory* (DFT), provide an accurate description of these interactions based on quantum mechanics. A major drawback of DFT is its time complexity, which has been shown to be cubic with respect to the protein size in the worst case [?,?]. For special cases this complexity can be reduced to being linear [?,?]. DFT implementations used for calculations on proteins are in between these bounds and typically show quadratic behavior with significant constant factors, rendering proteins bigger than a few hundred amino acids prohibitively expensive to compute [?,?]. As an example, Figure 4 in Appendix A shows an excerpt from experimental running times of quantum-chemical calculations on protein fragments which support this quadratic dependence.

To mitigate the computational cost, quantum-chemical subsystem methods have been developed [?,?]. In such approaches, large molecules are separated into fragments (= subsystems) which are then treated individually. A common way to deal with individual fragments is to assume that they do not interact with each other. The error this introduces for protein–protein or protein–molecule interaction energies (or for other local molecular properties of interest) depends on the size and location of fragments: A partition that cuts right through the strongest interaction in a molecule will give worse results than one that carefully avoids this. It should also be considered that a protein consists of a *main chain* (also called *backbone*) of amino acids. This main chain folds into 3D-secondary-structures, stabilized by non-bonding interactions (those not on the backbone) between the individual amino acids. These different connection types (backbone vs non-backbone) have different influence on the interaction energies.

Motivation. Subsystem methods are very powerful in quantum chemistry [?,?] but so far require manual cuts with chemical insight to achieve good partitions [?]. Currently, when automating the process, domain scientists typically cut every X amino acids along the main chain (which we will call the *naive approach* in the following). This gives in general suboptimal and unpredictable results (see Figure 2 in Appendix A).

By considering amino acids as nodes connected by edges weighted with the expected error in the interaction energies, one can construct (dense) graphs representing the proteins. Graph partitions with a light cut, i. e. partitions of the vertex set whose inter-fragment edges have low total weight, should then correspond to a low error for interaction energies. A general solution to this problem has high significance, since it is applicable to any subsystem-based method and since it will enable such calculations on larger systems with controlled accuracy. Yet, while several established graph partitioning algorithms exist, none of them is directly applicable to our problem scenarios due to additional domain-specific optimization constraints (which are outlined in Section 2).

Contributions. For the first of two problem scenarios, the special case of continuous fragments along the main chain, we provide in Section 4 a dynamic programming (DP) algorithm. We prove that it yields an optimal solution with a worst-case time complexity of $\mathcal{O}(n^2 \cdot \text{maxSize})$.

For the general protein partitioning problem, we provide three algorithms using established partitioning concepts, now equipped with techniques for adhering to the new constraints (see Section 5): (i) a greedy agglomerative method, (ii) a multilevel algorithm with Fiduccia-Mattheyses [?] refinement, and (iii) a simple postprocessing step that "repairs" traditional graph partitions.

Our experiments (Section 6) use several protein graphs representative for DFT calculations. Their number of nodes is rather small (up to 357), but they are complete graphs. The results show that our algorithms are usually better in quality than the naive approach. While none of the new algorithms is consistently the best one, the DP algorithm can be called most robust since it is always better in quality than the naive approach. A meta algorithm that would run all single algorithms and pick the best solution would still take only about ten seconds per instance and improves the naive approach on average by 13.5% to 20%, depending on the imbalance. In the whole quantum-chemical workflow the total partitioning time of this meta algorithm is still small.

2 Problem Description

Given an undirected connected graph $G = (V, E)$ with n nodes and m edges, a set of k disjoint non-empty node subsets $V_1, V_2, ...V_k$ is called a k-partition of G if the union of the subsets yields V ($V = \bigcup_{1 \leq i \leq k} V_i$). We denote partitions with the letter Π and call the subsets *fragments* in this paper.

Let $w(u, v)$ be the weight of edge $\{u, v\} \in E$, or 1 in an unweighted graph. Then, the *cut weight* of a graph partition is the sum of the weights of edges with endpoints in different subsets: $\text{cutweight}(\Pi, G) = \sum_{u \in V_i, v \in V_j, i \neq j, V_i, V_j \in \Pi} w(u, v)$. The largest fragment's size should not exceed $\text{maxSize} := (1 + \epsilon) \cdot \lceil n/k \rceil$, where ϵ is the so-called *imbalance* parameter. A partition is balanced iff $\epsilon = 0$.

Given a graph $G = (V, E)$ and $k \in \mathbb{N}_{\geq 2}$, *graph partitioning* is often defined as the problem of finding a k-partition with minimum cut weight while respecting the constraint of maximum imbalance ϵ. This problem is $\mathcal{NP}$-hard [**?**] for general graphs and values of ϵ. For the case of $\epsilon = 0$, no polynomial time algorithm can deliver a constant factor approximation guarantee unless $\mathcal{P}$ equals $\mathcal{NP}$ [**?**].

2.1 Protein Partitioning

We represent a protein as a weighted undirected graph. Nodes represent amino acids, edges represent bonds or other interactions. (Note that our graphs are different from protein interaction networks [**?**].) Edge weights are determined both by the strength of the bond or interaction and the importance of this edge to the protein function. Such a graph can be constructed from the geometrical structure of the protein using chemical heuristics whose detailed discussion is beyond our scope. Partitioning into fragments yields faster running time for DFT since the time required for a fragment is quadratic in its size. The cut weight of a partition corresponds to the total error caused by dividing this protein into fragments. A balanced partition is desirable as it maximizes this acceleration effect. However, relaxing the constraint with a small $\epsilon > 0$ makes sense as this usually helps in obtaining solutions with a lower error.

Note that the positions on the main chain define an ordering of the nodes. From now on we assume the nodes to be numbered along the chain.

New Constraints. Established graph partitioning tools using the model of the previous section cannot be applied directly to our problem since protein partitioning introduces additional constraints and an incompatible scenario due to chemical idiosyncrasies:

- The first constraint is caused by so-called *cap molecules* added for the subsystem calculation. These cap molecules are added at fragment boundaries (only in the DFT, not in our graph) to obtain chemically meaningful fragments. This means for the graph that if node i and node $i+2$ belong to the same fragment, node $i+1$ must also belong to that fragment. Otherwise the introduced cap molecules will overlap spatially and therefore not represent a chemically meaningful structure. We call this the *gap* constraint.
- More importantly, some graph nodes can have a charge. It is difficult to obtain robust convergence in quantum-mechanical calculations for fragments with more than one charge. Therefore, together with the graph a (possibly empty) list of charged nodes is given and two charged nodes must not be in the same fragment. This is called the *charge* constraint.

We consider here **two problem scenarios** (with different chemical interpretations) in the context of protein partitioning:

- **Partitioning along the main chain:** The main chain of a protein gives a natural structure to it. We thus consider a scenario where partition fragments are forced to be continuous on the main chain. This minimizes the number of cap molecules necessary for the simulation and has the additional advantage of better comparability with the naive partition.
  Formally, the problem can be stated like this: Given a graph $G = (V, E)$ with ascending node IDs according to the node's main chain position, an integer k and a maximum imbalance ϵ, find a k-partition with minimum cut weight such that $v_j \in V_i \wedge v_j + l \in V_i \rightarrow v_j + 1 \in V_i, 1 \leq j \leq n, l \in \mathbb{N}^+, 1 \leq i \leq k$ and which respects the balance, gap, and charge constraints.
- **General protein partitioning:** The general problem does not require continuous fragments on the main chain, but also minimizes the cut weight while adhering to the balance, gap, and charge constraints.

3 Related Work

3.1 General-purpose graph partitioning

General-purpose graph partitioning tools only require the adjacency information of the graph and no additional problem-related information. For special inputs (very small n or $k = 2$ and small cuts) sophisticated methods from mathematical programming [**?**] or using branch-and-bound [**?**] are feasible – and give provably optimal results. To be of general practical use, in particular for larger instances, most widely used tools employ local heuristics within a multilevel approach, though (see the survey by Buluc et al. [**?**]).

The multilevel metaheuristic, popularized for graph partitioning in the mid-1990s [**?**], is a powerful technique and consists of three phases: First, one computes a hierarchy of graphs $G_0, \ldots, G_l$ by recursive coarsening in the first phase. G_l ought to be small in size, but topologically similar to the input graph G_0. A very good initial solution for G_l is computed in the second phase. After that, the recursive coarsening is undone and the solution prolongated to the next-finer level. In this final phase, in successive steps, the respective prolongated solution on each level is improved using local search.

A popular local search algorithm for the third phase of the multilevel process is based on the method by Fiduccia and Mattheyses (FM) [**?**] (many others exist, see [**?**]). The main idea of FM is to exchange nodes between blocks in the order of the cost reductions possible, while maintaining a balanced partition. After every node has been moved once, the solution with the best cost improvement is chosen. Such a phase is repeated several times, each running in time $\mathcal{O}(m)$.

3.2 Methods for subsystem quantum chemistry

While this work is based on the *molecular fractionation with conjugate cap* (MFCC) scheme [**?**,**?**], several more sophisticated approaches have been developed which allow to decrease the size of the error in subsystem quantum-mechanical calculations [**?**,**?**,**?**]. The general idea is to reintroduce the interactions missed by the fragmentation of the supermolecule. A prominent example is the *frozen density embedding* (FDE) approach [**?**,**?**,**?**]. All these methods strongly depend on the underlying fragmentation of the supermolecule and it is therefore desirable to minimize the error in the form of the cut weight itself. Thus, the implementation shown in this paper is applicable to all quantum-chemical subsystem methods needing molecule fragments as an input.

4 Solving Main Chain Partitioning Optimally

As discussed in the introduction, a protein consists of a main chain, which is folded to yield its characteristic spatial structure. Aligning a partition along the main chain uses the locality information in the node order and minimizes the number of cap molecules necessary for a given number of fragments. The problem description from Section 2 – finding fragments with continuous node IDs – is equivalent to finding a set of $k-1$ *delimiter nodes* $v_{d_1}, v_{d_2}, ...v_{d_{k-1}}$ that separate the fragments. Note that this is not a vertex separator, instead the delimiter nodes induce a set of cut edges due to the continuous node IDs. More precisely, delimiter node v_{d_j} belongs to fragment j, $1 \leq j \leq k-1$.

Consider the delimiter nodes in ascending order. Given the node v_{d_2}, the optimal placement of node v_{d_1} only depends on edges among nodes $u < v_{d_2}$, since all edges $\{u, v\}$ from nodes $u < v_{d_2}$ to nodes $v > v_{d_2}$ are cut no matter where v_{d_1} is placed. Placing node v_{d_2} thus induces an optimal placement for v_{d_1}, using only information from edges to nodes $u < v_{d_2}$. With this dependency of the positions of v_{d_1} and v_{d_2}, placing node v_{d_3} similarly induces an optimal choice for v_{d_2} and v_{d_1}, using only information from nodes smaller than v_{d_3}. The same argument can be continued inductively for nodes $v_{d_4} \ldots v_{d_k}$.

Algorithm 1 is our dynamic-programming-based solution to the main chain partitioning problem. It uses the property stated above to iteratively compute the optimal placement of $v_{d_{j-1}}$ for all possible values of v_{d_j}. Finding the optimal placements of $v_{d_1}, \dots v_{d_{j-1}}$ given a delimiter v_{d_j} at node i is equivalent to the subproblem of partitioning the first i nodes into j fragments, for increasing values of i and j. If n nodes and k fragments are reached, the desired global solution is found. We allocate (Line 3) and fill an $n \times k$ table partCut with the optimal values for the subproblems. More precisely, the table entry partCut$[i][j]$ denotes the minimum cut weight of a j-partition of the first i nodes:

Lemma 1. *After the execution of Algorithm 1,* partCut$[i][j]$ *contains the minimum cut value for a continuous j-partition of the first i nodes. If such a partition is impossible,* partCut$[i][j]$ *contains ∞.*

We prove the lemma after describing the algorithm. After the initialization of data structures in Lines 2 and 3, the initial values are set in Line 4: A partition consisting of only one fragment has a cut weight of zero.

All further partitions are built from a *predecessor partition* and a new fragment. A j-partition $\Pi_{i,j}$ of the first i nodes consists of the jth fragment and a $(j-1)$-partition with fewer than i nodes. A valid predecessor partition of $\Pi_{i,j}$ is a partition $\Pi_{l,j-1}$ of the first l nodes, with l between $i - \text{maxSize}$ and $i-1$. Node charges have to be taken into account when compiling the set of valid predecessors. If a backwards search for $\Pi_{i,j}$ from node i encounters two charged nodes a and b with $a < b$, all valid predecessors of $\Pi_{i,j}$ contain at least node a (Line 7).

The additional cut weight induced by adding a fragment containing the nodes $[l+1, i]$ to a predecessor partition $\Pi_{l,j-1}$ is the weight sum of edges connecting nodes in $[1, l]$ to nodes in $[l+1, i]$: $c[l][i] = \sum_{\{u,v\} \in E, u \in [1,l], v \in [l+1,i]} w(u,v)$. Line 8 computes this weight difference for the current node i and all valid predecessors l.

For each i and j, the partition $\Pi_{i,j}$ with the minimum cut weight is then found in Line 10 by iterating backwards over all valid predecessor partitions and selecting the one leading to the minimum cut. To reconstruct the partition, we store the predecessor in each step (Line 11). If no partition with the given values is possible, the corresponding entry in partCut remains at ∞.

After the table is filled, the resulting minimum cut weight is at partCut$[n][k]$, the corresponding partition is found by following the predecessors (Line 16).

We are now ready to prove Lemma 1 and the algorithm's correctness and time complexity.

Proof (of Lemma 1). By induction over the number of partitions j.

Base Case: $j = 1, \forall i$. A 1-partition is a continuous block of nodes. The cut value is zero exactly if the first i nodes contain at most one charge and i is not larger than maxSize. This cut value is written into partCut in Lines 3 and 4 and not changed afterwards.

Inductive Step: $j-1 \to j$. Let i be the current node: A cut-minimal j-partition $\Pi_{i,j}$ for the first i nodes contains a cut-minimal $(j-1)$-partition $\Pi_{i',j-1}$ with continuous node blocks. If $\Pi_{i',j-1}$ were not minimum, we could find a better partition $\Pi'_{i',j-1}$ and use it to improve $\Pi_{i,j}$, a contradiction to $\Pi_{i,j}$ being cut-minimal. Due to the induction hypothesis, partCut$[l][j-1]$ contains the minimum

Algorithm 1: Main Chain Partitioning with Dynamic Programming

```
   Input: Graph $G = (V, E)$, fragment count $k$, bool list $isCharged$, imbalance $\epsilon$
   Output: partition $\Pi$
1  maxSize= $\lceil |V|/k \rceil \cdot (1 + \epsilon)$;
2  allocate empty partition $\Pi$;
3  partCut[i][j] $= \infty, \forall i \in [1, n], \forall j \in [1, k]$;
   /* initialize empty table partCut with $n$ rows and $k$ columns       */
4  partCut[i][1] $= 0$, $\forall i \in [1, \text{maxSize}]$;
5  for $1 \leq i \leq n$ do
6      windowStart $= \max(i - \text{maxSize}, 1)$;
7      if necessary, increase windowStart so that [windowStart, i] contains at most
       one charged node;
8      compute column $i$ of cut cost table $c$;
9      for $2 \leq j \leq k$ do
10         partCut[i][j] $= \min_{l \in [windowStart, i]}$ partCut$[l][j-1] + c[l][i]$;
11         pred[i][j] $= \text{argmin}_{l \in [windowStart, i]}$ partCut$[l][j-1] + c[l][i]$;
12     end
13 end
14 $i = n$;
15 for $j = k$; $j \geq 2$; $j -= 1$ do
16     $nextI$ = pred$[i][j]$;
17     assign nodes between $nextI$ and $i$ to fragment $\Pi_j$;
18     i = $nextI$;
19 end
20 return $\Pi$
```

cut value for all node indices l, which includes i'. The loop in Line 10 iterates over possible predecessor partitions $\Pi_{l,j-1}$ and selects the one leading to the minimum cut after node i. Given that partitions for $j-1$ are cut-minimal, the partition whose weight is stored in partCut$[i][j]$ is cut-minimal as well.

If no allowed predecessor partition with a finite weight exists, partCut$[i][j]$ remains at infinity. □

Theorem 1. *Algorithm 1 computes the optimal main chain partition in time* $\mathcal{O}(n^2 \cdot \text{maxSize})$.

Proof. The correctness in terms of optimality follows directly from Lemma 1. We thus continue with establishing the time complexity. The nested loops in Lines 5 and 9 require $\mathcal{O}(n \cdot k)$ iterations in total. Line 7 is executed n times and has a complexity of maxSize. At Line 10 in the inner loop, up to maxSize predecessor partitions need to be evaluated, each with two constant time table accesses. Computing the cut weight column $c[\cdot][i]$ for fragments ending at node i (Line 8) involves summing over the edges of $\mathcal{O}(\text{maxSize})$ predecessors, each having at most $\mathcal{O}(n)$ neighbors. Since the cut weights constitute a reverse prefix sum, the column $c[\cdot][i]$ can be computed in $\mathcal{O}(n \cdot \text{maxSize})$ time by iterating backwards. Line 8 is executed n times, leading to a total complexity of $\mathcal{O}(n^2 \cdot \text{maxSize})$.

Following the predecessors and assigning nodes to fragments is possible in linear time, thus the $\mathcal{O}(n^2 \cdot \text{maxSize})$ to compile the cut cost table dominates the running time. $\square$

5 Algorithms for General Protein Partitioning

As discussed in Section 2, one cannot use general-purpose graph partitioning programs due to the new constraints required by the DFT calculations. Moreover, if the constraint of the previous section is dropped, the DP-based algorithm is not optimal in general any more. Thus, we propose three algorithms for the general problem in this section: The first two, a greedy agglomerative method and Multilevel-FM, build on existing graph partitioning knowledge but incorporate the new constraints directly into the optimization process. The third one is a simple postprocessing repair procedure that works in many cases. It takes the output of a traditional graph partitioner and fixes it so as to fulfill the constraints.

5.1 Greedy Agglomerative Algorithm

The greedy agglomerative approach, shown in Algorithm 2 in Appendix B, is similar in spirit to Kruskal's MST algorithm and to approaches proposed for clustering graphs with respect to the objective function modularity [**?**]. It initially sorts edges by weight and puts each node into a singleton fragment. Edges are then considered iteratively with the heaviest first; the fragments belonging to the incident nodes are merged if no constraints are violated. This is repeated until no edges are left or the desired fragment count is achieved.

The initial edge sorting takes $\mathcal{O}(m \log m)$ time. Initializing the data structures is possible in linear time. The main loop (Line 5) has at most m iterations. Checking the size and charge constraints is possible in constant time by keeping arrays of fragment sizes and charge states. The time needed for checking the gaps and merging is linear in the fragment size and thus at most $\mathcal{O}(\text{maxSize})$.

The total time complexity of the greedy algorithm is thus:

$$T(\text{Greedy}) \in \mathcal{O}(m \cdot \max\{\text{maxSize}, \log m\}).$$

5.2 Multilevel Algorithm with Fiduccia-Mattheyses Local Search

Algorithm 3 (Appendix B) is similar to existing multilevel partitioners using non-binary (i.e. $k > 2$) Fiduccia-Mattheyses (FM) local search. Our adaptation incorporates the constraints throughout the whole partitioning process, though. First a hierarchy of graphs $G_0, G_1, \ldots G_l$ is created by recursive coarsening (Line 1). The edges contracted during coarsening are chosen with a local matching strategy. An edge connecting two charged nodes stays uncontracted, thus ensuring that a fragment contains at most one charged node even in the coarsest partitioning phase. The coarsest graph is then partitioned into Π_l using region growing or recursive bisection. If an optional input partition Π' is given, it is used as a guideline during coarsening and replaces Π_l if it yields a better cut. We execute both our greedy and DP algorithm and use the partition with the better cut as input partition Π' for the multilevel algorithm.

After obtaining a partition for the coarsest graph, the graph is iteratively uncoarsened and the partition projected to the next finer level. We add a rebalancing step at each level (Line 6), since a non-binary FM step does not guarantee balanced partitions if the input is imbalanced. A Fiduccia-Mattheyses step is then performed to yield local improvements (Line 10): For a partition with k fragments, this non-binary FM step consists of one priority queue for each fragment. Each node v is inserted into the priority queue of its current fragment, the maximum gain (i. e. reduction in cut weight when v is moved to another fragment) is used as key. While at least one queue is non-empty, the highest vertex of the largest queue is moved if the constraints are still fulfilled, and the movement recorded. After all nodes have been moved, the partition yielding the minimum cut is taken. In our variant, nodes are only moved if the charge constraint stays fulfilled.

5.3 Repair Procedure

As already mentioned, traditional graph partitioners produce in general solutions that do not adhere to the constraints for protein partitioning. To be able to use existing tools, however, we propose a simple repair procedure for an existing partition which possibly does not fulfill the charge, gap, or balance constraints. To this end, Algorithm 4 in Appendix B performs one sweep over all nodes (Line 6) and checks for every node v whether the constraints are violated at this point. If they are and v has to be moved, an FM step is performed: Among all fragments that could possibly receive v, the one minimizing the cut weight is selected. If no suitable target fragment exists, a new singleton fragment is created. Note that due to the local search, this step can lead to more than k fragments, even if a partition with k fragments is possible.

The cut weight table allocated in Line 1 takes $\mathcal{O}(n \cdot k + m)$ time to create. Whether a constraint is violated can be checked in constant time per node by counting the number of nodes and charges observed for each fragment. A node needs to be moved when at least one charge or at least maxSize nodes have already been encountered in the same fragment. Finding the best target partition (Line 13) takes $\mathcal{O}(k)$ iterations, updating the cut weight table after moving a node v is linear in the degree $\deg(v)$ of v. The total time complexity of a repair step is thus: $\mathcal{O}(n \cdot k + m + n \cdot k + \sum_v \deg(v)) = \mathcal{O}(n \cdot k + m)$.

6 Experiments

6.1 Settings

We evaluate our algorithms on graphs derived from several proteins and compare the resulting cut weight. As main chain partitioning is a special case of general protein partitioning, the solutions generated by our dynamic programming algorithm are valid solutions of the general problem, though perhaps not optimal. Other algorithms evaluated are Algorithm 2 (Greedy), 3 (Multilevel), and the external partitioner KaHiP [**?**], used with the repair step discussed in Section 5.3. The algorithms are implemented in C++ and Python using the NetworKit tool suite [**?**], the source code is available from a (before acceptance private) hg repository[3] with login "rev-sea16" and password "ubiquitin".

We use graphs derived from five common proteins, covering the most frequent structural properties. Ubiquitin [**?**] (also see Figure 3 in Appendix A) and the Bubble Protein [**?**] are rather small proteins with 76 and 64 amino acids, respectively. Due to their biological functions, their overall size and their diversity in the contained structural features, they are commonly used as test cases for quantum-chemical subsystem methods [**?**]. The Green Fluorescent Protein (GFP) [**?**] plays a crucial role in the bioluminescence of marine organisms and is widely expressed in other organisms as a fluorescent label for microscopic techniques. Like the latter one, Bacteriorhodopsin (bR) [**?**] and the Fenna-Matthews-Olson protein (FMO) [**?**] are large enough to render quantum-chemical calculations on the whole proteins practically infeasible. Yet, investigating them with quantum-chemical methods is key to understanding the photochemical processes they are involved in. The graphs derived from the latter three proteins have 225, 226 and 357 nodes, respectively. They are complete graphs with weighted $n(n-1)/2$ edges. All instances can be found in the mentioned hg repository in folder `input/`.

In our experiments we partition the graphs into fragments of different sizes (i. e. we vary the fragment number k). The small proteins ubiquitin and bubble are partitioned into 2, 4, 6 and 8 fragments, leading to fragments of average size 8-38. The other proteins are partitioned into 8, 12, 16, 20 and 24 fragments, yielding average sizes between 10 and 45. As maximum imbalance, we use values for ϵ of 0.1 and 0.2. While this may be larger than usual values of ϵ in graph partitioning, fragment sizes in our case are comparably small and an imbalance of 0.1 is possibly reached with the movement of a single node.

On these proteins, the running time of all partitioning implementations is on the order of a few seconds on a commodity laptop, we therefore omit detailed time measurements.

Charged Nodes. Depending on the environment, some of the amino acids are charged. As discussed in Section 2, at most one charge is allowed per fragment. We repeatedly sample $\lfloor 0.8 \cdot k \rfloor$ random charged nodes among the potentially charged, under the constraint that a valid main chain partition is still possible. To smooth out random effects, we perform 20 runs with different random nodes charged. Introducing charged nodes may cause the naive partition to become invalid. In these cases, we use the repair procedure on the invalid naive partition and compare the cut weights of other algorithms with the cut weight of the repaired naive partition.

[3] `https://algohub.iti.kit.edu/parco/NetworKit/NetworKit-chemfork/`

6.2 Results

For the uncharged scenario, Figure 1 shows a comparison of cut weights for different numbers of fragments and a maximum imbalance of 0.1. The cut weight is up to 34.5% smaller than with the naive approach (or 42.8% with $\epsilon = 0.2$, see Figure 5). The best algorithm choice depends on the protein: For ubiquitin, green fluorescent protein, and Fenna-Matthew-Olson protein, the external partitioner KaHiP in combination with the repair step described in Section 5.3 gives the lowest cut weight when averaged over different fragment sizes. For the bubble protein, the multilevel algorithm from Section 5.2 gives on average the best result, while for bacteriorhodopsin, the best cut weight is achieved by the dynamic programming (DP) algorithm. The DP algorithm is always as least as good as the naive approach. This already follows from Theorem 1, as the naive partition is aligned along the main chain and thus found by DP in case it is optimal. DP is the only algorithm with this property, all others perform worse than the naive approach for at least one combination of parameters.

The general intuition that smaller fragment sizes leave less room for improvements compared to the naive solution is confirmed by our experimental results. Figure 5 (Appendix A) shows the comparison with imbalance $\epsilon = 0.2$. While the general trend is similar and the best choice of algorithm depends on the protein, the cut weight is usually more clearly improved. Moreover, a meta algorithm that executes all single algorithms and picks their best solution yields average improvements (geometric mean) of $13.5\%, 16\%$, and 20% for $\epsilon = 0.1, 0.2$, and 0.3, respectively, compared to the naive reference. Such a meta algorithm requires only about ten seconds per instance, negligible in the whole DFT workflow.

Randomly charging nodes changes the results only insignificantly, as seen in Figure 6. The necessary increase in cut weight for the algorithm's solutions is likely compensated by a similar increase in the naive partition due to the necessary repairs.

7 Conclusions

Partitioning protein graphs for subsystem quantum-chemistry is a new problem with unique constraints which general-purpose graph partitioning algorithms were unable to handle. We have provided several algorithms for this problem and proved the optimality of one in the special case of partitioning along the main chain. With our algorithms chemists are now able to address larger problems in an automated manner with smaller error. Larger proteins, in turn, in connection with a reasonable imbalance, may provide more opportunities for improving the quality of the naive solution further.

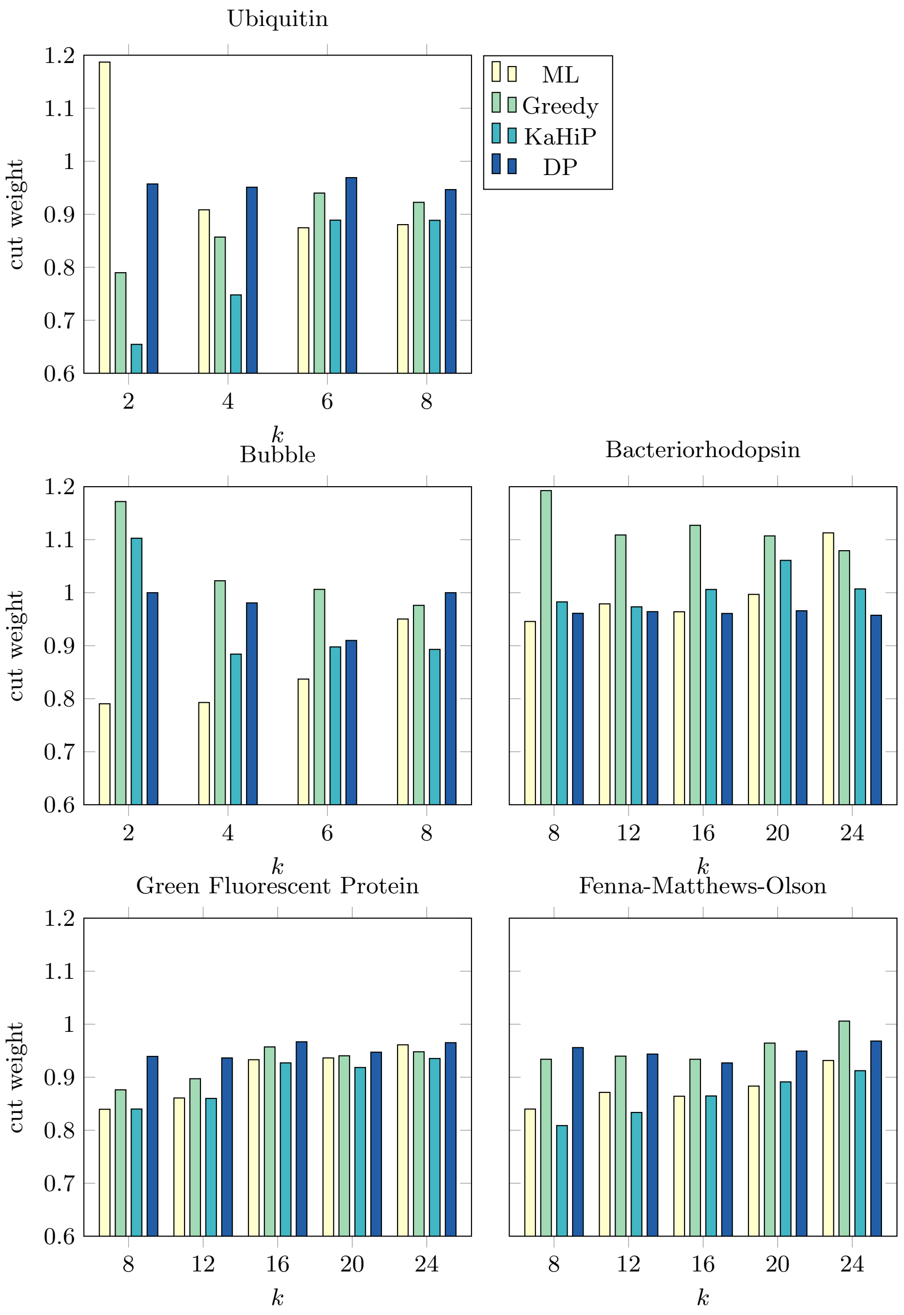

Fig. 1. Comparison of partitions given by several algorithms and proteins, for $\epsilon = 0.1$. The partition quality is measured by the cut weight in comparison to the naive solution.

Appendix

A Illustrations and Additional Experimental Results

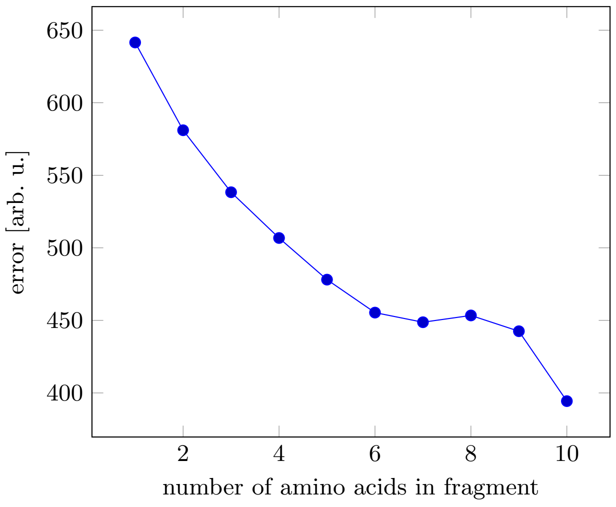

Fig. 2. Predicted error for interaction energies with naive fragmentation every X amino acids for the small protein ubiquitin. Unpredictable minima and maxima depending on the location of the uniformly distributed cuts occur along the main chain.

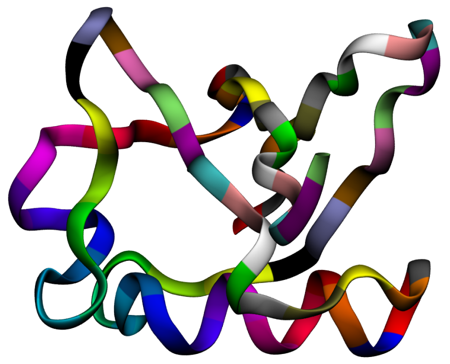

Fig. 3. 3D-Visualization of Ubiquitin. Single amino acids in different colors. Helical secondary structure at the bottom, beta-sheet like secondary structures in the upper left and right.

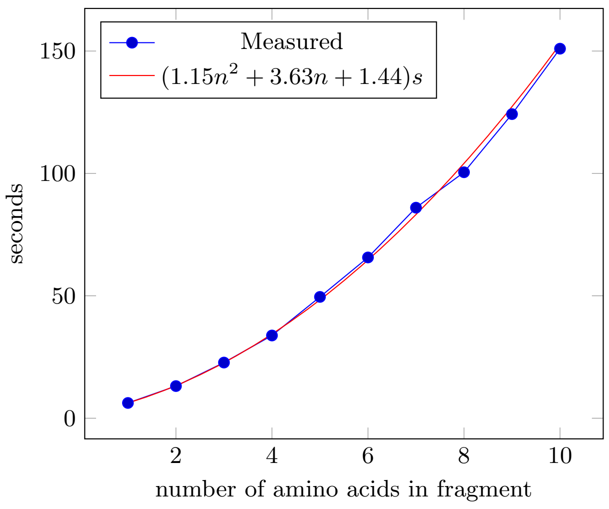

Fig. 4. Time in seconds required for quantum chemical density functional (DFT, BP86, DZP) calculations of protein fragments on 16 Intel Xeon cores (2x Haswell-EP/2640v3/2.6 GHz) executed with pyADF [**?**] and ADF program package [**?**]. As seen by the close match of the red fit line, time grows quadratically.

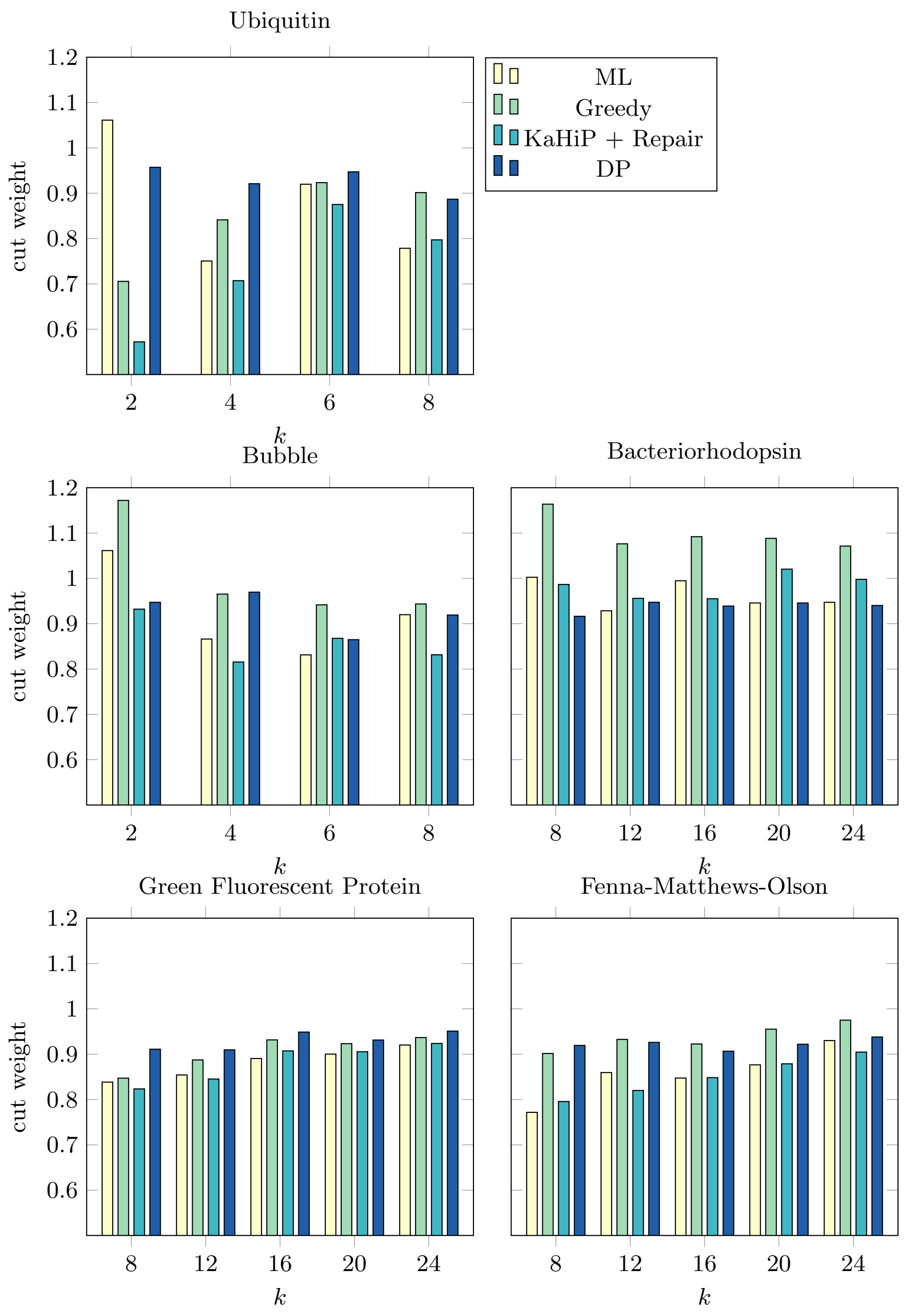

Fig. 5. Comparison of cut weights for $\epsilon = 0.2$.

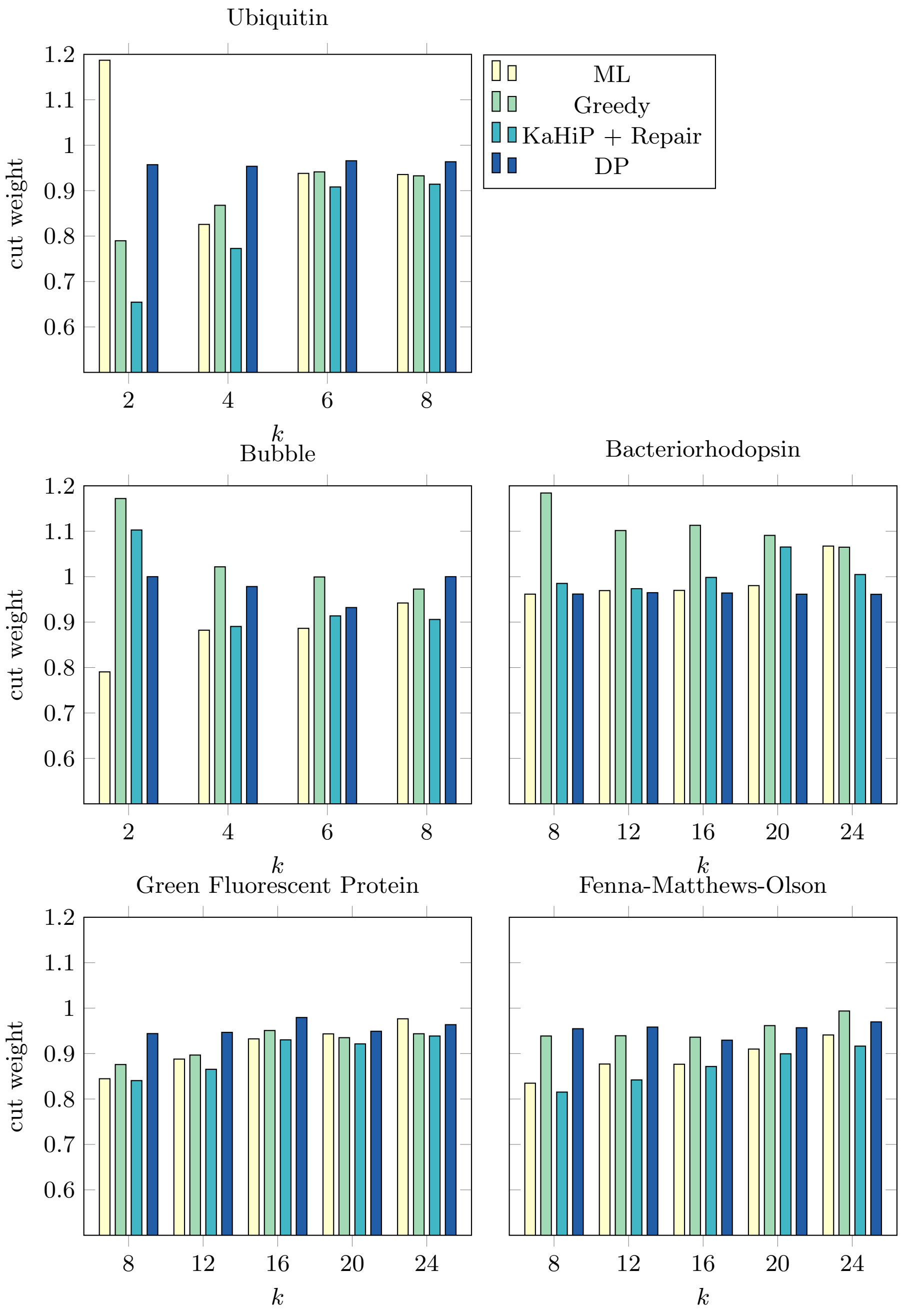

Fig. 6. Comparison of cut weights for $\epsilon = 0.1$ and node charges.

B Additional Pseudocodes

Algorithm 2: Greedy Agglomerative Algorithm

```
Input: Graph $G = (V, E)$, fragment count $k$, list charged, imbalance $\epsilon$
Output: partition $\Pi$
sort edges by weight, descending;
$\Pi$ = create one singleton partition for each node;
chargedPartitions = partitions containing a charged node;
maxSize= $\lceil |V|/k \rceil \cdot (1 + \epsilon)$;
for edge $\{u, v\}$ do
    allowed = True;
    if $\Pi[u] \in$ chargedPartitions and $\Pi[v] \in$ chargedPartitions then
        allowed = False;
    end
    if $|\Pi[u]| + |\Pi[v]| >$ maxSize then
        allowed = False;
    end
    for node $x \in \Pi[u] \cup \Pi[v]$ do
        if $x + 2 \in \Pi[u] \cup \Pi[v]$ and $x + 1 \notin \Pi[u] \cup \Pi[v]$ then
            allowed = False;
        end
    end
    if allowed then
        merge $\Pi[u]$ and $\Pi[v]$;
        update chargedPartitions;
    end
    if number of fragments in $\Pi$ equals $k$ then
        break;
    end
end
return $\Pi$
```

Algorithm 3: Multilevel-FM

```
   Input: Graph $G = (V, E)$, fragment count $k$, list charged, imbalance $\epsilon$, $[\Pi']$
   Output: partition $\Pi$
 1 $G_0, \ldots, G_l$ = hierarchy of coarsened Graphs, $G_0 = G$;
 2 $\Pi_l$ = partition $G_l$ with region growing or recursive bisection;
 3 for $0 \leq i < l$ do
 4     uncoarsen $G_i$ from $G_{i+1}$;
 5     $\Pi_i$ = projected partition from $\Pi_{i+1}$;
 6     rebalance $\Pi_i$, possibly worsen cut weight;
       /* Local improvements                                  */
 7     gain = NaN;
 8     repeat
 9         oldcut = cut($\Pi'_i, G$);
10         $\Pi'_i$ = Fiduccia-Mattheyses-Step of $\Pi_i$ with constraints;
11         gain = cut($\Pi'_i, G$) - oldcut;
12     until gain == 0;
13 end
```

Algorithm 4: Repairing a partition

```
   Input: Graph $G = (V, E)$, $k$-partition $\Pi$, list charged, imbalance $\epsilon$
   Output: partition $\Pi'$
 1 cutWeight[$i$][$j$] $= 0, 1 \leq i \leq n, 1 \leq j \leq k$;
 2 for edge $\{u, v\}$ in $E$ do
 3     cutWeight[$u$][$\Pi(u)$]$+= w(u, v)$;
 4     cutWeight[$v$][$\Pi(v)$]$+= w(u, v)$;
 5 end
 6 for node $v$ in $V$ do
       /* Check whether node can stay                         */
 7     if charge violated or size violated or gap of size 1 then
 8         $\Psi$ = set of allowed target fragments;
 9         if $\Psi$ is empty then
10             create new fragment for $v$;
11         end
12         else
               /* Fiduccia-Mattheyses-step: To minimize the cut weight,
                  move the node to the fragment to which it has the
                  strongest connection                             */
13             target = $\mathrm{argmax}_{i \in \Psi}$\{cutWeight[$v$][$i$]\};
14             move $v$ to target;
15         end
16         update charge counter, size counter and cutWeight;
17     end
18 end
```