Computing Bond Types in Molecule Graphs

Sebastian Böcker[1,2], Quang B.A. Bui[1], Patrick Seeber[1], and Anke Truss[1]

[1] Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, 07743 Jena, Germany
sebastian.boecker@uni-jena.de
{bui,patrick.seeber,truss}@minet.uni-jena.de
[2] Jena Centre for Bioinformatics, Jena, Germany

Abstract. In this paper, we deal with restoring missing information in molecule databases: Some data formats only store the atoms' configuration but omit bond multiplicities. As this information is essential for various applications in chemistry, we consider the problem of recovering bond type information using a scoring function for the possible valences of each atom—the Bond Type Assignment problem. We prove the NP-hardness of Bond Type Assignment and give an exact fixed-parameter algorithm for the problem where bond types are computed via dynamic programming on a tree decomposition of the molecule graph. We evaluate our algorithm on a set of real molecule graphs and find that it works fast and accurately.

1 Introduction

The *structural formula* of a chemical compound is a representation of the molecular structure, showing both how the atoms are arranged and the chemical bonds within the molecule. Throughout this paper, we refer to structural formula as *molecule graph*. An important information on a molecule graph is the bond type. It is essential for many applications in chemistry, for example to compute the molecular mechanics force field [12]. Unfortunately, this information is omitted by many data formats that represent molecule graphs, such as Gaussian file formats and Mopac file formats, and even by the widely used Protein Data Bank format *pdb* [12]. Moreover, in combinatorial chemistry, the backbone of the molecule (skeletal formula) may be drawn either manually or automatically, again omitting bond types. Now the question is how to reassign such bond types to the molecule graph. This problem is aggravated by the fact that several elements can have different states of valence, enabling them to form different numbers of bonds.

Previous work. Several heuristic approaches have been introduced for this problem [8,12,3]. An integer linear programming based algorithm can solve the problem exactly for small molecules [6]. Recently, Dehof *et al.* introduced a running time heuristic for this problem (paper in preparation).

Our contribution. In this paper, we prove that the problem of reassigning bond types to molecule graph is NP-hard. We then introduce a *tree decomposition-based* algorithm that computes an exact solution for the problem with running

H.Q. Ngo (Ed.): COCOON 2009, LNCS 5609, pp. 297–306, 2009.

time $O(a_{max}^{2\omega} \cdot 3^{\alpha} \cdot \omega \cdot m + m)$, where m is the number of nodes in the tree decomposition of the molecule graph, a_{max} is the maximum *open valence* of an atom, d is the maximum degree of an atom in the molecule, $\omega - 1$ is the treewidth of the molecule graph, and $\alpha := \min\{\binom{\omega}{2}, \omega\, d\}$. With our algorithm, we prove that the problem of reassigning bond types to molecule graphs is *fixed-parameter tractable* (FPT) [5,10] in the treewidth of the molecule graph and the maximum open valence of an atom in the molecule. Furthermore, we implemented our algorithm and evaluated it on real molecules of the MMFF94 dataset. As we expected, the treewidth of molecule graph is rather small for biomolecules: for all molecule graphs in our dataset, the treewidth is at most three, and our algorithm solves the problem mostly in well under a second.

2 Definitions

A *molecule graph* is a graph $G = (V, E)$ where each vertex in V corresponds to an atom and each edge in E corresponds to a chemical bond between the respective atoms. We denote an edge from u to v by uv. For every vertex $v \in V$, let $\mathcal{A}_v \subseteq \{1, \ldots, max\_valence\}$ be the set of *valences* of the atom at vertex v, where *max_valence* denotes the maximum valence of an atom in the molecule. Then $A_v := \mathcal{A}_v - \deg(v) = \{a - \deg(v) : a \in \mathcal{A}_v,\, a - \deg(v) \geq 0\}$ is the set of admissible *open valences* we can still assign to v. We set $A_v^* := \max A_v$.

Let $b : E \to \{0, 1, 2\}$ be a weight function assigning a bond type, represented by the bond multiplicity lowered by one, to every bond $uv \in E$. We call such b an *assignment*. An assignment b *determines* a valence $x_b(v) := \sum_{u \in N(v)} b(uv) + \deg v$ for every vertex $v \in V$, where $N(v)$ denotes the set of neighbors of v. The assignment b is *feasible* if $x_b(v) \in \mathcal{A}_v$ for every vertex $v \in V$.

A *scoring function* $S_v : \mathcal{A}_v \to \mathbb{R}^+$ assigns a finite positive score $S_v(a)$ to a valence $a \in \mathcal{A}_v$ at vertex v. The score $S(b)$ of an assignment b is

$$S(b) := \sum_{v \in V} S_v(x_b(v)). \tag{1}$$

Given a molecule graph $G = (V, E)$, our task is to find a feasible assignment b for G with minimum score $S(b)$.

For open valences, we define a scoring function $s_v : \{0, \ldots, A_v^*\} \to \mathbb{R}^+$ via $s_v(a) := S_v(a + \deg v)$ for $a + \deg v \in \mathcal{A}_v$, and $s_v(a) := \infty$ otherwise. We can express (1) in terms of open valences: For an assignment b, the atom at vertex v take valence $y_b(v) := \sum_{u \in N(v)} b(uv)$ and we can compute

$$S(b) = \sum_{v \in V} s_v(y_b(v)).$$

By this definition, $S(b) = \infty$ holds for assignments b that are not feasible.

In this paper, we mostly work with the abovementioned open valences. Therefore open valence is referred to as *valence* for simplicity. The Bond Type Assignment Problem is defined as follows:

Bond Type Assignment Problem. Given a molecule graph $G = (V, E)$ with (open) valence sets $A_v \subseteq \{1, \ldots, a_{max}\}$ and scoring functions s_v for every $v \in V$. Find a feasible assignment b for G with minimum score $S(b)$.

In Sect. 3, we show that the Bond Type Assignment problem is NP-hard even if every vertex of the molecule graph is of degree at most three and the atom at every vertex has a valence of at most four.

3 Hardness of the Problem

Given an input for the Bond Type Assignment problem, the Bond Type Assignment *decision* problem asks if there is a feasible assignment b for the input graph. In this section, we show that this problem is NP-hard. Therefore, Bond Type Assignment is also NP-hard.

Theorem 1. *The* Bond Type Assignment *decision problem is NP-hard, even on input graphs where every vertex has degree at most three and atom valences are at most four.*

In our hardness proof, we will use reduction from a variant of the 3-sat problem:

Definition (3-sat). Given a set X of n boolean variables $\{x_1, \ldots, x_n\}$ and a set C of m clauses $\{c_1, \ldots, c_m\}$. Each clause is a disjunction of at most three literals over X, for example $(x_1 \vee \overline{x_2} \vee x_3)$. Is there an assignment $X \to \{true, false\}$ that satisfies all clauses in C, i.e., at least one literal in every clause is *true*?

Definition (3-sat*). The variant of 3-sat where every variable occurs at most three times and every literal at most twice is called the 3-sat* problem.

To transform a 3-sat problem instance into a 3-sat* problem instance, we first replace every t-th occurrence, $t \geq 3$, of a variable with a new variable and some auxiliary clauses of length two, which assure that new variables are consistently assigned. Then we remove all variables that only occur in either positive or negative literals and clauses containing such variables. The resulting 3-sat instance is a 3-sat* instance and it is satisfiable if and only if the original 3-sat instance is satisfiable. This 3-sat* instance is only larger than the original 3-sat instance by a polynomial factor. Since 3-sat is NP-hard [7], the 3-sat* problem is also NP-hard.

Proof (of Theorem 1). By a polynomial reduction from 3-sat* to the Bond Type Assignment decision problem, we will show that the Bond Type Assignment decision problem is NP-hard, even if every vertex is of degree at most three and valence at most four.

Given a 3-sat* formula, we can safely discard all clauses containing variables that only occur in either positive or negative literals. Afterwards, every variable occurs at least twice and at most three times in at least one positive and one negative literal. We then construct the sat-graph $G = (V, E)$ for the Bond Type Assignment problem as follows:

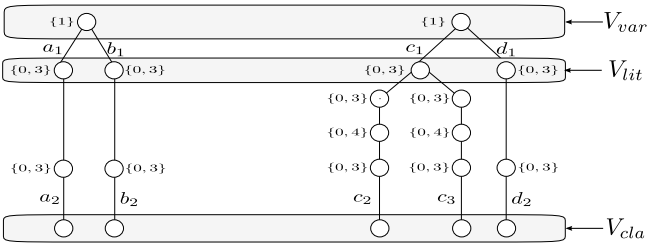

Fig. 1. The building blocks of G. The unmasked nodes are auxiliary vertices. The valence sets of the clause vertices are not shown here.

The vertex set V consists of four subsets V_{var}, V_{lit}, V_{cla} and V_{aux}. For each variable x_i of the 3-SAT* instance, the vertex set V_{var} contains a *variable vertex* v_i and the vertex set V_{lit} contains two *literal vertices* u_i and u'_i corresponding to the literals x_i and $\overline{x_i}$. The set V_{cla} contains, for every clause c_j of the 3-SAT* instance, a *clause vertex* w_j. Finally, we need a couple of auxiliary vertices subsumed in V_{aux} as shown in Fig. 1.

The valence set of each variable vertex is $\{1\}$, of each literal vertex $\{0, 3\}$, and of a clause vertex $\{1, \ldots, d\}$, where $d \leq 3$ is the number of literals contained in the corresponding clause. The valence sets of auxiliary vertices are set as shown in Fig. 1. We use the trees shown in Fig. 1 as building blocks to connect the vertices of G.

If both literals of a variable occur once, we connect each of the literal vertices to the clause vertex that corresponds to the clause containing this literal via an auxiliary vertex with valence set $\{0, 3\}$. See Fig. 1(left).

If one literal of a variable occurs once and the other twice, we connect the literal vertex, which corresponds to the literal occurring in one clause, to the corresponding clause vertex via an auxiliary vertex with valence set $\{0, 3\}$. The literal vertex corresponding to the literal occurring in two clauses is connected to each of the corresponding clause vertices via a chain of three auxiliary vertices with valence sets $\{0, 3\}, \{0, 4\}, \{0, 3\}$. See Fig. 1 (right).

Before proving that the constructed Bond Type Assignment instance has a feasible assignment if and only if 3-SAT* instance is satisfiable, we consider the two building blocks of G shown in Fig. 1. Let a_1, a_2, b_1, b_2, c_1, c_2, c_3, d_1, d_2 denote the bond type of the corresponding edges as shown in Fig. 1. In a feasible assignment of G, the following facts can be easily observed:

The bond types a_1, a_2, b_1, b_2, c_1, c_2, c_3, d_1, d_2 can take a value of one or two. The bond type two can only be assigned to either a_1 or b_1, and to either c_1 or d_1, and the corresponding literal vertex has to take valence three, the other one has to take valence zero. Furthermore, it holds that $a_1 = a_2$, $b_1 = b_2$, $c_1 = c_2 = c_3$ and $d_1 = d_2$.

The fact that exactly one of two edges incident to a variable vertex is a double binding models the rule that exactly one of the literals $x_i, \overline{x_i}$ of a variable x_i is satisfied. The valence of a clause vertex takes a value of at least one if and only if the corresponding clause contains literals whose literal vertices have valence

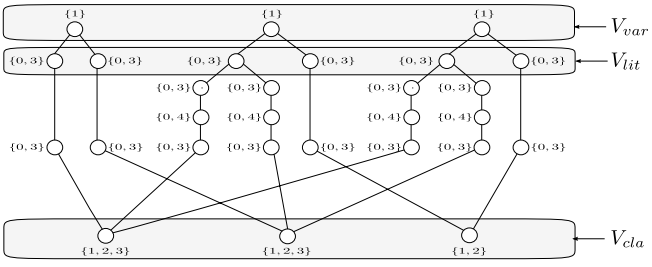

Fig. 2. The unmasked vertices are auxiliary vertices. The variable vertices represents variables x_1, x_2, x_3 from left to right. The literal vertices represent literals $x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}$ from left to right. The clause vertices represent clauses $(x_1 \vee x_2 \vee x_3)$, $(\overline{x_1} \vee x_2 \vee x_3)$, $(\overline{x_2} \vee \overline{x_3})$ from left to right.

three. This implies that a clause is satisfied if and only if it contains a *true* literal. Furthermore, the valence set $\{1, \ldots, d(w)\}$ of a clause vertex w forces any algorithm for the Bond Type Assignment problem to assign a double binding to at least one of the edges incident to w. This implies that at least one of the literals contained in each clause has to be *true*.

Therefore, there is a feasible solution for the constructed Bond Type Assignment instance if and only if the 3-sat* instance is satisfiable. Since the reduction can be done in polynomial time and the 3-sat* problem is NP-hard, the Bond Type Assignment decision problem is also NP-hard. □

4 The Algorithm

To solve the Bond Type Assignment problem exactly, we use the dynamic programming approach on the *tree decomposition* of the input graph [11]. In the following subsection, we give a short introduction to the tree decomposition concept. We follow Niedermeier's monograph [10] in our presentation.

4.1 Tree Decompositions

Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $\langle \{X_i \mid i \in I\}, T \rangle$ where each X_i is a subset of V, called a bag, and T is a tree containing the elements of I as nodes and the three following properties must hold:

1. $\bigcup_{i \in I} X_i = V$;
2. for every edge $\{u, v\} \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$; and
3. for all $i, j, k \in I$, if j lies on the path between i and k in T then $X_i \cap X_k \subseteq X_j$.

The *width* of the tree decomposition $\langle \{X_i \mid i \in I\}, T \rangle$ equals $\max\{|X_i| \mid i \in I\} - 1$. The *treewidth* of G is the minimum number $\omega - 1$ such that G has a tree decomposition of width $\omega - 1$.

Given a molecule graph G, we first compute the tree decomposition T of G before executing our algorithm on T to solve the Bond Type Assignment

problem on G. As we show later, the running time and the required space of our algorithm grow exponentially with the treewidth of G. Therefore, the smaller the width of the tree decomposition of G, the better running time our algorithm will achieve. Unfortunately, computing a tree decomposition with minimum width is an NP-hard problem [2]. But our input graphs usually show a particular structure that allows us to efficiently compute optimal tree decompositions.

A graph is called *outerplanar* if it admits a crossing-free embedding in the plane such that all vertices are on the same face. A graph is 1-*outerplanar* if it is outerplanar; and it is r-*outerplanar* for $r > 1$ if, after removing all vertices on the boundary face, the remaining graph is an $(r-1)$-outerplanar graph. Every r-outerplanar graph has treewidth at most $3r-1$ [4], and we can compute optimal tree decompositions of r-outerplanar graphs in time $O(r \cdot n)$ [1], where n is the number of vertices in the graph. The important observation here is that most molecule graphs of biomolecules are r-outerplanar for some small integer r, such as $r = 2$. For such molecules, we can first compute the optimal tree decomposition of the molecule graph, and since the treewidth is rather small, our tree decomposition-based algorithm can be used to solve the Bond Type Assignment problem efficiently.

To improve the legibility and to simplify description and analysis of our algorithm, we use *nice tree decompositions* instead of arbitrary tree decompositions in the remaining part of this paper. Here, we assume the tree T to be rooted. A tree decomposition is a *nice tree decomposition* if it satisfies the following conditions:

1. Every node of the tree has at most two children.
2. If a node i has two children j and k, then $X_i = X_j = X_k$; in this case i is called a *join node*.
3. If a node has one child j, the one of the following situations must hold:
   (a) $|X_i| = |X_j| + 1$ and $X_j \subset X_i$; in this case X_i is called an *introduce node*.
   (b) $|X_i| = |X_j| - 1$ and $X_i \subset X_j$; in this case X_i is called a *forget node*.

After computing a tree decomposition of width k and m nodes for the input graph G, we transform this tree decomposition into a nice tree decomposition with the same treewidth and $O(m)$ nodes in linear time using the algorithm introduced in [9] (Lemma 13.1.3). Then we execute our algorithm on the nice tree decomposition to compute the optimal bond type assignment for G.

4.2 Tree Decomposition-Based Algorithm

Assume that a nice tree decomposition $\langle \{X_i \mid i \in I\}, T \rangle$ of width $\omega - 1$ and $O(m)$ nodes of the molecule graph G is given. In this section, we describe a dynamic programming algorithm that solves the Bond Type Assignment problem using the nice tree decomposition of the molecule graph G.

The tree T is rooted at an arbitrary bag. Above this root we add additional forget nodes, such that the new root contains a single vertex. Let X_r denote the new root of the tree decomposition and v_r denote the single vertex contained in

X_r. Analogously, we add additional introduce nodes under every leaf of T, such that the new leaf also contains a single vertex.

The vertices inside a bag X_i are referred to as $v_1, v_2, \ldots, v_k$ where $k \leq \omega$. For simplicity of presentation, we assume that all edges $v_1v_2, v_1v_3, \ldots, v_{k-1}v_k$ are present in each bag. Otherwise, the recurrences can be simplified accordingly.

Let Y_i denote the vertices in G that are contained in the bags of the subtree below bag X_i. We assign a score matrix D_i to each bag X_i of the tree decomposition: let $D_i[a_1, \ldots, a_k;\, b_{1,2}, \ldots, b_{k-1,k}]$ be the minimum score over all valency assignments to the vertices in $Y_i \setminus X_i$ if for every $l = 1, \ldots, k$, a_l bonds of vertex v_l have been consumed by the vertices in $Y_i \setminus X_i$, and bond types $b_{1,2}, \ldots, b_{k-a1,k}$ are assigned to edges $v_1v_2, v_1v_3, \ldots, v_{k-1}v_k$. Using this definition, we delay the scoring of any vertex to the forget node where it is removed from a bag. This is advantageous since every vertex except for the root vertex v_r is forgotten exactly once, and since the exact valence of a vertex is not known until it is forgotten in the tree decomposition. Finally, we can compute the minimum score among all assignments using the root bag $X_r = \{v_r\}$ as $\min_a s_{v_r}(a) + D_r[a]$.

Our algorithm begins at the leaves of the tree decomposition and computes the score matrix D_i for every node X_i when score matrices of its children nodes have been computed. We initialize the matrix D_j of each leaf $X_j = \{v\}$ with

$$D_j[a_1; \cdot] = \begin{cases} 0 & \text{if } a_1 = 0, \\ \infty & \text{otherwise.} \end{cases}$$

During the bottom-up travel, the algorithm distinguishes if X_i is a forget node, an introduce node, or a join node, and computes D_i as follows:

Introduce nodes. Let X_i be a parent node of X_j such that $X_j = \{v_1, \ldots, v_{k-1}\}$ and $X_i = \{v_1, \ldots, v_k\}$. Then

$$D_i[a_1, \ldots, a_k; b_{1,2}, \ldots, b_{k-1,k}] = \begin{cases} D_j[a_1, \ldots, a_{k-1};\, b_{1,2}, \ldots, b_{k-2,k-1}] & \text{if } a_k = 0, \\ \infty & \text{otherwise.} \end{cases}$$

Forget nodes. Let X_i be a parent node of X_j such that $X_j = \{v_1, \ldots, v_k\}$ and $X_i = \{v_1, \ldots, v_{k-1}\}$. Then

$$D_i[a_1, \ldots, a_{k-1};\, b_{1,2}, \ldots, b_{k-2,k-1}] = \min_{\substack{b_{1,k}, \ldots, b_{k-1,k} \in \{0,1,2\} \\ a_k \in \{0, \ldots, A^*_{v_k}\}}} \Bigg\{ s_{v_k}\Big(a_k + \sum\nolimits_{l=1}^{k} b_{l,k}\Big) + D_j[a_1 - b_{1,k}, \ldots, a_{k-1} - b_{k-1,k}, a_k;\, b_{1,2}, \ldots, b_{k-1,k}] \Bigg\}$$

Join nodes. Let X_i be a parent node of X_j and X_h such that $X_i = X_j = X_h$. Then

$$D_i[a_1, \ldots, a_k; b_{1,2}, \ldots, b_{k-1,k}] = \min_{\substack{a'_l = 0, \ldots, a_l \\ \text{for } l = 1, \ldots, k}} \Big\{ D_j[a'_1, \ldots, a'_k;\, b_{1,2}, \ldots, b_{k-1,k}] + D_h[a_1 - a'_1, \ldots, a_k - a'_k;\, b_{1,2}, \ldots, b_{k-1,k}] \Big\}$$

For simplicity of the presentation of our algorithm, we assumed above that every two vertices in each bag of the tree decomposition are connected by an edge, but in reality, the degree of a vertex in a molecule graph cannot exceed the maximum valence $d \leq 7$ of an atom in the molecule graph. Therefore, the number of edges in a bag is upper-bounded by ωd.

Lemma 1. *Given a nice tree decomposition of a molecule graph G, the algorithm described above computes an optimal assignment for the* Bond Type Assignment *problem on G in time $O(a_{max}^{2\omega} \cdot 3^{\alpha} \cdot \omega \cdot m + m)$, where $a_{max} = \max_v A_v^*$ is the maximum (open) valence of an atom, m and $\omega - 1$ are size and width of the tree decomposition, d is the maximum degree in the molecule graph, and $\alpha := \min\{\binom{\omega}{2}, \omega\, d\}$.*

Due to space constraints, we defer the proof of Lemma 1 to the full paper. If the optimal solution is saved during the bottom-up processing, we can traverse the tree decomposition top-down afterwards to obtain all bond types of the molecule. This can be done in time $O(m)$.

5 Algorithm Engineering and Computational Results

Clearly, we do not have to compute or store entries $D_j[a_1, \ldots, a_k; b_{1,2}, \ldots, b_{k-1,k}]$ with $a_i + \sum_j b_{i,j} > A_i^*$ for some i, because such entries will never be used for the computation of minima in forget nodes or the root. We may implicitly assume that all such entries are set to infinity. Instead of an array, we use a hash map and store only those entries of D that are smaller than infinity. This reduces both running times and memory of our approach in applications.

To evaluate the performance of our algorithm, we implemented the algorithm in Java. All computations were done on an AMD Opteron-275 2.2 GHz with 6 GB of memory running Solaris 10. For our experiment, we used the MMFF94 dataset,[1] which consists of 760 molecule graphs predominantly derived from the Cambridge Structural Database. Bond types are given in the dataset but we removed this information and reassigned the bond types to those molecule graphs. We removed 30 molecule graphs that contain elements such as lithium or chlorine not covered in our scoring table (see below), or that have atom bindings such as oxygen atoms connected to three other atoms, that are also not covered in our scoring. The largest molecule graphs contains 59 atoms, the smallest 3 atoms, the average 23 atoms.

To compute the optimal tree decompositions of the molecule graphs, we used the method QuickBB in the library LibTW implemented by van Dijk *et al.* (`http://www.treewidth.com`). We implemented a method to transform the computed optimal tree decompositions into nice tree decompositions. In view of the near-outerplanarity of molecule graphs, we expected treewidths to be

[1] `http://www.ccl.net/cca/data/MMFF94/`, source file `MMFF94_dative.mol2`, of Feb. 5, 2009.

Table 1. Overview on the data used in our experiment. "Treewidth" gives the range of treewidths in this group, and "TD" and "DP" are average running times for tree decomposition and dynamic programming in milliseconds, respectively. "average # solutions" is the number of solutions our algorithm found on average.

instance size		number of		average	running time		average
$\|V\|$	$\|E\|$	instances	treewidth	treewidth	TD	DP	# solutions
3–10	2–11	63	1–2	1.2	4	5	1.4
11–20	10–22	206	1–3	1.8	6	36	1.2
21–30	20–33	328	1–3	2.0	6	68	1.3
31–40	30–43	125	1–3	2.0	8	76	1.3
41–50	40–53	5	1–2	1.8	9	87	1.4
51–59	53–61	3	2	2.0	5	234	1.0

rather small: In fact, we find that 18.6 % of all molecules have treewidth one, 96.6 % have treewidth ≤ 2, and all molecules have treewidth at most three. The average treewidth is 1.85.

For scoring an assignment, we use the scoring table from Wang *et al.* [12]. This scoring allows atoms to have rather "exotic" valences, but gives an *atomic penalty score* (aps) to these rare valence states. As an example, carbon is allowed to take valence two (with aps 64), three (aps 32), four (aps 0), five (aps 32), or six (aps 64). In addition, different scores can be applied for the same element, depending on the local neighborhood: For example, carbon in a carboxylate group COO^- can take valence four (aps 32), five (aps 0), or six (aps 32). See Table 2 in [12] for details.

See Table 1 for computational results. Total running times are usually well below one second, and 56 ms on average. We were able to recover the correct assignment in all cases. In some cases, the algorithm found up to six optimal solutions because of symmetries and aromatic rings in the molecule graph.

6 Conclusion

We considered the problem of assigning bond types to a molecule graph and showed that the problem is NP-hard. Based on the tree decomposition concept, we introduced a dynamic programming algorithm with running time practically linear in the size of the molecule. In contrast to the previous heuristic and integer linear programming based algorithms, our algorithm is the first algorithm that computes exact solutions for the problem in a guaranteed running time. Furthermore, the running time of our algorithm depends strongly on the structure of the graph, but not on the size of the graph. We expect that the algorithm can be applied to solve the problem on large molecules if the molecule graph has small treewidth.

As a next step of algorithm design, we will do further algorithm engineering to improve the running time of our algorithm for practical uses. Furthermore, we want to evaluate the quality of solutions and the running time of our algorithm

against other algorithms. In particular, we want to verify that our algorithm finds more chemically or biologically relevant solutions than heuristic approaches.

Acknowledgment

We thank Anne Dehof and Andreas Hildebrandt for introducing us to the problem. Q.B.A. Bui gratefully acknowledges financial support from the DFG, research group "Parameterized Algorithms in Bioinformatics" (BO 1910/5).

References

1. Alber, J., Dorn, F., Niedermeier, R.: Experimental evaluation of a tree decomposition based algorithm for Vertex Cover on planar graphs. Discrete Appl. Math. 145(2), 219–231 (2005)
2. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embedding in a k-tree. SIAM J. Algebra. Discr. 8, 277–284 (1987)
3. Baber, J.C., Hodgkin, E.E.: Automatic assignment of chemical connectivity to organic molecules in the cambridge structural database. J. Chem. Inf. Model. 32, 401–406 (1992)
4. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theor. Comput. Sci. 209, 1–45 (1998)
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
6. Froeyen, M., Herdewijn, P.: Correct bond order assignment in a molecular framework using integer linear programming with application to molecules where only non-hydrogen atom coordinates are available. J. Chem. Inf. Model. 5, 1267–1274 (2005)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability (A Guide to Theory of NP-Completeness). Freeman, New York (1979)
8. Hendlich, M., Rippmann, F., Barnickel, G.: BALI: automatic assignment of bond and atom types for protein ligands in the brookhaven protein databank. J. Chem. Inf. Model. 37, 774–778 (1997)
9. Kloks, T.: Treewidth, Computation and Approximation. Springer, Heidelberg (1994)
10. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
11. Robertson, N., Seymour, P.: Graph minors: algorithmic aspects of tree-width. J. Algorithms 7, 309–322 (1986)
12. Wang, J., Wang, W., Kollmann, P.A., Case, D.A.: Automatic atom type and bond type perception in molecular mechanical calculations. J. Mol. Graph. Model. 25, 247–260 (2006)