Computing Bond Orders in Molecule Graphs

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

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

Abstract. In this paper, we deal with restoring missing information in molecule databases: Many 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 Order Assignment problem. We show that the Bond Order Assignment is NP-hard, and its maximization version is MAX SNP-hard. We then give two exact fixed-parameter algorithms for the problem, where bond orders 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 in practice.

This is a preprint of:
Sebastian Böcker, Quang Bao Anh Bui, and Anke Truss. Computing Bond Orders in Molecule Graphs. *Theor. Comput. Sci.* **412**(12–14):1184-1195, 2011.

1 Introduction

The structural formula of a chemical compound is a representation of the molecular structure, showing both how atoms are arranged and the chemical bonds between pairs of atoms. Throughout this paper, we refer to a structural formula as a *molecule graph*. An important aspect of structural formulas is *bond order* (referred to as *bond type* in [17]) information: Each bond between two atoms can be a single, double, or triple bond. See Fig. 1 for the molecule graph of phenylalanine, an amino acid. The sum of bonds of any atom is its valence, and each element allows for a certain set of admissible valence states.

Bond order information is essential for many applications in chemistry, such as computing the molecular mechanics force field [17]. Unfortunately, bond orders can be omitted in 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. So, many entries in public databases omit bond order information, whereas other entries have erroneous such information. Moreover, in combinatorial chemistry, the backbone

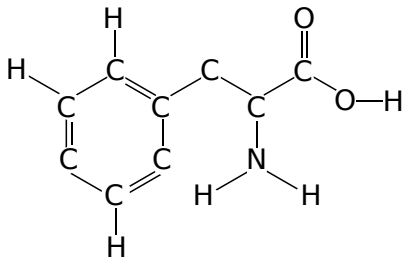

Fig. 1. Molecule graph of phenylalanine.

of a molecule (skeletal formula) may be drawn either manually or automatically, again omitting bond orders. Now, the question is how to (re-)assign bond orders to the molecule graph. Note that a molecular graph may allow for several admissible bond order assignments.

Previous work. Several approaches for this problem have been introduced during the last years, based on different optimization criteria [3, 10, 12]. For example, Froeyen and Herdewijn [8] presented an integer linear programming algorithm that minimizes formal charge on each atom. Wang *et al.* [17] formulated the bond order assignment problem as a minimization problem, where each atom contributes to the additive objective function based on its valence state. The authors also introduced a penalty score table to evaluate rare valence states, and presented a heuristic to search for the optimal bond order assignment. In our work, we concentrate on this optimization criterion. Recently, Dehof *et al.* [6] introduced a branch-and-bound algorithm and an integer linear program for this formulation of the problem.

Informally, the problem can be described as follows: We are given a graph where every vertex has one or more natural nummbers attached to it. We then have to assign weights zero, one, or two to the edges of the graph, such that for every vertex, the sum of weights of its incident edges equals one of the numbers attached to the vertex. For the optimization version, we additionally evaluate the possible choices at each vertex by a score, and sum up these scores.

Our contribution. In this paper, we show that the problem of assigning bond orders to molecule graphs, as introduced by Wang *et al.* [17], is NP-hard even on input graphs where both vertex degree and maximum valence are bounded by a constant. We infer that the problem cannot be approximated when minimizing the objective function. Furthermore, we show that assigning bond orders maximizing an objective function is MAX SNP-hard even on molecule graphs with the same restriction. This implies the non-existence of a polynomial time approximation scheme (PTAS) for the maximization version of the problem, unless P = NP [2]. We then introduce two *tree decomposition-based* algorithms that compute an exact solution of the problem with running times $O(\alpha^{2\omega} \cdot 3^{\beta} \cdot \omega \cdot m)$ and $O(\alpha^{3\omega} \cdot \omega \cdot m)$, where m is the number of nodes in the tree decomposition of the molecule graph, $\alpha - 1$ 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 $\beta := \min\{\binom{\omega}{2}, \omega d\}$. This shows that the problem of reassigning bond orders to molecule graphs is *fixed-parameter tractable* [7, 13] with respect to the treewidth of the molecule graph and the maximum open valence of an atom in the molecule. We implemented one of our algorithms and evaluated it on molecules of the MMFF94 dataset, a dataset originally used by Halgren *et al.* [9] to validate Merck Molecular Force Fields. As we expected, the treewidths of molecule graphs are rather small for biomolecules: for all graphs in our dataset, the treewidth is at most three, and our algorithm solves the problem in well under a second. To further confirm the practical use of our algorithms, we computed the treewidths of 135607 molecules in a PubChem dataset. We find that 99.99 % of these molecules have treewidths of at most three, and no molecule's treewidth exceeds four.

2 Preliminaries

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. Since each edge in G already consumes one valence

of the corresponding vertices, the set of admissible *open valences* we can assign to v is $A_v := \{a - \deg(v) : a \in \mathcal{A}_v,\ a - \deg(v) \geq 0\}$, where $\mathcal{A}_v$ denotes the set of *valences* of the atom at vertex v. We set $A_v^* := \max A_v$ and $\alpha := 1 + \max_v A_v^*$. In this paper, we mostly work with the abovementioned open valences. Therefore, "open valence" is referred to as *valence* for simplicity.

Let $b : E \to \{0, 1, 2\}$ be a weight function assigning a bond order lowered by one to every bond $uv \in E$. We call such b an *assignment*. An assignment b *consumes* $x_b(v) := \sum_{u \in N(v)} b(uv)$ valences of the atom at every vertex $v \in V$, where $N(v)$ denotes the set of neighbors of v. The assignment b is *feasible* if $x_b(v) \in A_v$ for every vertex $v \in V$.

A *scoring function* $s_v : A_v \to \mathbb{R}_{\geq 0}$ assigns a non-negative score $s_v(a)$ to an open valence $a \in A_v$ at every vertex $v \in V$. For every $v \in V$, we define $s_v(a) := \infty$ if $a \notin A_v$.

The score $S(b)$ of an assignment b is

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

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

The Bond Order Assignment Problem is defined as follows:

Bond Order Assignment Problem. Given a molecule graph $G = (V, E)$ with (open) valence sets $A_v \subseteq \{0, \ldots, \alpha - 1\}$ and scoring functions s_v for every $v \in V$. Find a feasible assignment b for G with minimum score $S(b)$.

3 Hardness of the Problem

Given an input for the Bond Order Assignment problem, the Bond Order Assignment *decision* problem asks if there is a feasible assignment b for the input graph. In this section, we first show that this problem is NP-hard, even if every vertex of the molecule graph has degree at most three and atoms have valences of at most four, and so is the Bond Order Assignment problem. For the proof, we use a polynomial-time reduction from a variant of 3-SAT problem to the Bond Order Assignment problem.

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

From this theorem, we can quite easily infer that the Bond order Assignment problem cannot be approximated, since a feasible solution can have score zero:

Lemma 1. *The* Bond Order Assignment *problem cannot be approximated in polynomial time, unless P = NP, even on input graphs where every vertex has degree at most three and atom valences are at most four, and s_v is binary.*

The previous lemma might be regarded as an artifact, as asking for a solution of minimum score is somewhat arbitrary, and Wang *et al.* [17] could have formulated the bond order assignment problem as a maximization problem. This is similar to MAX-3SAT, where we do not minimize the number of unsatisfied clauses, but maximize the number of satisfied clauses. Consequently, we can use a positive score instead of a penalty score, and ask for a bond order assignment with *maximum* score. By the following theorem, finding an assignment with maximum score is a MAX SNP-hard problem, which implies the non-existence of a polynomial time approximation scheme (PTAS) unless P = NP [2].

Theorem 2. *Computing a bond order assignment with maximum score is a MAX SNP-hard problem, even on input graphs where every vertex has degree at most three and atom valences are at most four, and s_v is binary.*

We first focus on the NP-hardness of the decision problem. In the proof of Theorem 1, we will use a reduction from a variant of the 3-SAT problem:

Definition 1 (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 2 (3-SAT*). *The variant of 3-SAT where each variable occurs at most three times, is called the 3-SAT* problem.*

In [4], Berman *et al.* proved the NP-hardness of 3-SAT* by a polynomial reduction from the 3-SAT problem.

Proof (of Theorem 1). Obviously, the problem is in NP. By a polynomial reduction from 3-SAT* to the Bond Order Assignment decision problem, we will show that the Bond Order 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 Order Assignment problem as follows:

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. 2.

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. 2. We use the trees shown in Fig. 2 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. 2 (left).

If one literal of a variable occurs once and the other twice, we connect the literal vertex that corresponds to the literal occurring in only 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. 2 (right).

Before proving that the constructed Bond Order 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. 2. Let $a_1, a_2, b_1, b_2, c_1, c_2, c_3, d_1, d_2$ denote the bond orders of

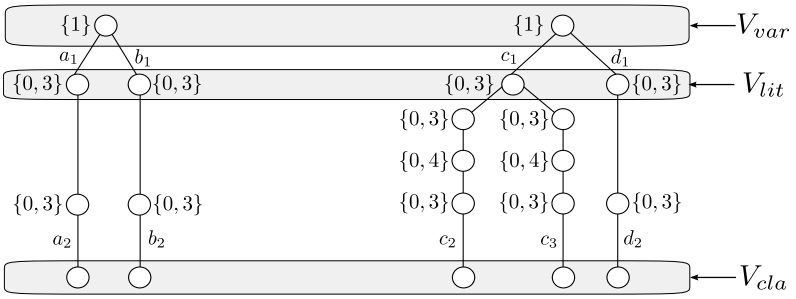

Fig. 2. The building blocks of G. Vertices outside V_{var}, V_{lit}, and V_{cla} are auxiliary vertices. Valence sets of clause vertices are omitted.

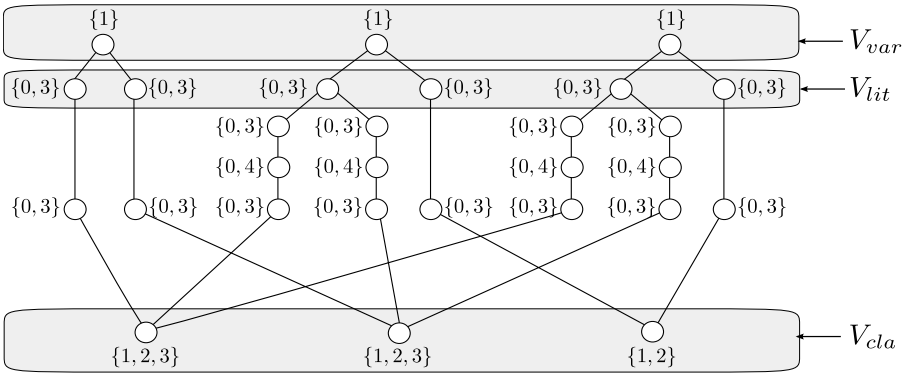

Fig. 3. The variable vertices represents variables x_1, x_2, x_3 from left to right. The literal vertices represent literals $x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}$, from left to right. The clause vertices represent clauses $(x_1 \vee x_2 \vee x_3)$, $(\overline{x_1} \vee x_2 \vee x_3)$, $(\overline{x_2} \vee \overline{x_3})$, from left to right.

the corresponding edges as shown in Fig. 2. In a feasible assignment of G, the following facts can be easily observed:

As all variable vertices have valence set $\{1\}$, the bond orders a_1, b_1, c_1, and d_1 can either be zero or one. Bond order one can only be assigned to either a_1 or b_1, and to either c_1 or d_1. The corresponding literal vertex has valence three, the other one has valence zero. Furthermore, we infer $a_1 = a_2$, $b_1 = b_2$, $c_1 = c_2 = c_3$ and $d_1 = d_2$.

The fact that exactly one of two edges incident to a variable vertex has bond type one, models 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 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 Order Assignment problem to assign bond order one 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 Order 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 Order Assignment decision problem is also NP-hard. □

Next, we prove that the Bond Order Assignment problem is not approximable in polynomial time, unless P = NP.

Proof (of Lemma 1). We modify the reduction described in the proof of Theorem 1 by allowing clause vertices to take valence zero, so that the valence set of a clause vertex is $\{0, 1, \ldots, d\}$ where $d \leq 3$ is the length of the clause. Valence zero at each clause vertex is penalized with score one, whereas the scores of all other valences of all vertices are set to zero. Doing so, we ensure that the score of an assignment is the number of clause vertices with valence zero and, hence, the number of unsatisfied clauses in the 3-SAT* problem instance. So, the 3-SAT* problem instance is satisfiable if and only if there is a bond order assignment for the SAT-graph with score zero.

Assume that there is a polynomial-time approximation algorithm for the Bond Order Assignment problem with an arbitrary approximation factor. This algorithm computes a bond order assignment with score zero for a Bond Order Assignment problem instance if and only if an optimal assignment has score zero. In particular, for Bond Order Assignment problem instances constructed from a 3-SAT* problem instance, this polynomial-time approximation algorithm computes an assignment with score zero if and only if the 3-SAT* problem instance is satisfiable. So, we can use this approximation algorithm to solve the NP-hard 3-SAT* problem in polynomial time. Thus, there is no polynomial-time approximation algorithm for the Bond Order Assignment problem, unless P = NP. □

Finally, we focus on the MAX SNP-hardness of the maximization version of the problem. The MAX SNP-hardness concept was introduced by Papadimitriou et al. [14], who also defined the L-reduction to show MAX SNP-hardness of an optimization problem. The L-reduction is defined as follows:

Definition 3 (L-reduction). *Let Π and Π' be two optimization (maximization or minimization) problems. We say that Π L-reduces to Π' if there are two polynomial-time algorithm f, g and constants $\delta, \gamma > 0$ such that for each instance I of Π:*

1. *Algorithm f produces an instance $I' = f(I)$ of Π', such that the optima of I and I', $OPT(I)$ and $OPT(I')$, respectively, satisfy $OPT(I') \leq \delta OPT(I)$.*
2. *Given any solution of I' with cost c', algorithm g produces a solution of I with cost c such that $|c - OPT(I)| \leq \gamma |c' - OPT(I')|$.*

To prove MAX SNP-hardness of computing a bond order assignment with maximum score, we introduce an L-reduction from the MAX-3SAT* problem, which is a MAX SNP-hard problem [14]. Papadimitriou *et al.* [14] show the MAX SNP-hardness of the MAX-3SAT problem where each variable can occur at most B times, for any given integer $B \geq 3$. Note that here, we cannot use the reduction from MAX-3SAT to MAX-3SAT* introduced for our NP-hardness proof, since this reduction is not an L-reduction. Our proof is straightforward, as we will set $\delta = \gamma = 1$.

Definition 4 (MAX-3SAT*). *Given a set of length three clauses over a set of boolean variables, where each variable occurs at most three times in the clause set, the MAX-3SAT* problem asks for an assignment of the variables that satisfies as many clauses as possible.*

Proof (of Theorem 2). We reduce MAX-3SAT* to the maximization version of Bond Order Assignment via an L-reduction. Given a MAX-3SAT* problem instance, we construct a SAT-graph as described in the proof of Theorem 1 and use the valence sets

and scoring functions defined in the proof of Lemma 1, but we swap the scoring functions at clause vertices, namely we assign score zero to valence zero and score one to non-zero valences. By doing this we can ensure that there is always a feasible solution for the constructed Bond Order Assignment problem instance, and the score of a bond order assignment is the number of clause vertices that have non-zero valences.

Since a clause vertex has non-zero valence if and only if the corresponding clause is satisfied, from a solution of the MAX-3SAT* problem instance that satisfied k clauses we can construct a solution of the Bond Order Assignment problem instance with score k, and vice versa. This can be done in polynomial time as described in the proof of Theorem 1. In the following, we use k to denote the number of satisfied clauses of the MAX-3SAT* problem instance, as well as the score of the corresponding Bond Order Assignment problem instance.

Let $OPT(BOA)$ denote the score of the optimal solution of the constructed Bond Order Assignment problem instance, and $OPT(M3S)$ denote the number of satisfied clauses in the optimal solution of the MAX-3SAT* problem instance. We infer that $OPT(BOA) = OPT(M3S)$.

To prove that our reduction from MAX-3SAT* problem to the maximization version of the Bond Order Assignment problem is an L-reduction, we have to show that $OPT(BOA) \leq \delta \cdot OPT(M3S)$ and $|k - OPT(M3S)| \leq \gamma \cdot |k - OPT(BOA)|$ hold for some constant δ and γ. Since $OPT(BOA) = OPT(M3S)$, both conditions hold for $\delta = \gamma = 1$.

All in all, our reduction from MAX-3SAT* to the maximization version of the Bond Order Assignment is an L-reduction. Since MAX-3SAT* is MAX SNP-hard, computing a bond order assignment with maximum score is also MAX SNP-hard, even on input graphs where every vertex has degree at most three and atom valences are at most four, and s_v is binary. □

4 Algorithms on graphs of bounded treewidth

While the Bond Order Assignment problem is computationally hard on general graphs, it can be solved in polynomial time on trees. To this end, we root the tree T at an arbitrary node r and set the direction of every edge to point away from r. We use the ordered pair uv to denote the directed edge from u to v. We assume that T is a binary tree, the general case can be solved similarly. We use dynamic programming, starting at the leaves of the tree. Let $D_v[a_v, e_{uv}]$ denote the optimum solution of the subtree rooted at v, by assuming that valence a_v is assigned to v and the bond order e_{uv} is assigned to edge uv. Let uv be an edge in T and w_1, w_2 be the two children of v. We can compute $D_v[\cdot, \cdot]$ using the recurrence

$$D_v[a_v, e_{uv}] = s_v(a_v) + \min_{\substack{e_1+e_2 \\ +e_{uv}=a_v}} \left\{ D_{w_1}[a_1, e_1] + D_{w_2}[a_2, e_2] \right\}$$

where the minimum is taken over all $a_1 \in A_{w_1}, a_2 \in A_{w_2}$ and all $e_1, e_2 \in \{0, 1, 2\}$. We initialize the recurrence for every leaf w with parent v of T: If $a_w = e_{vw}$ then set $D_w[a_w, e_{vw}] = s_w(a_w)$, and $D_w[a_w, e_{vw}] = \infty$ otherwise. Now, the value $\min_{a_r \in A_r} D_r[a_r, 0]$ is the minimum score for T and can be computed in polynomial time.

Since the Bond Order Assignment problem can be solved in polynomial time on trees, it is quite natural to extend the above algorithm to graphs of bounded treewidth. Here, we use dynamic programming on the *tree decomposition* of the input graph [15]. In

the following subsection, we give a short introduction to the tree decomposition concept. We follow Niedermeier's monograph [13] 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 Order 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 [1]. But there exist a variety of methods, both exact and heuristic, to compute tree decompositions in practice, see Sec. 6.

To improve legibility and to simplify description and analysis of our algorithm, we use *nice tree decompositions* instead of arbitrary tree decompositions in the following. We generally 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*.

Figure 4 illustrates the molecule graph of adenine, its tree- and nice tree-decomposition.

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 $O(m)$ bags in linear time using the algorithm introduced in [11] (Lemma 13.1.3). Then we execute our algorithm on the nice tree decomposition to compute the optimal bond order assignment for G.

We will now present our tree decomposition-based algorithm in two flavors: The one presented in Sec. 4.2 uses a dynamic programming matrix over bond strengths of *edges*, whereas the second algorithm presented in Sec. 4.3 uses a matrix over valences of *vertices*.

4.2 The $O(\alpha^{2\omega} \cdot 3^{\beta} \cdot \omega \cdot m)$ algorithm

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

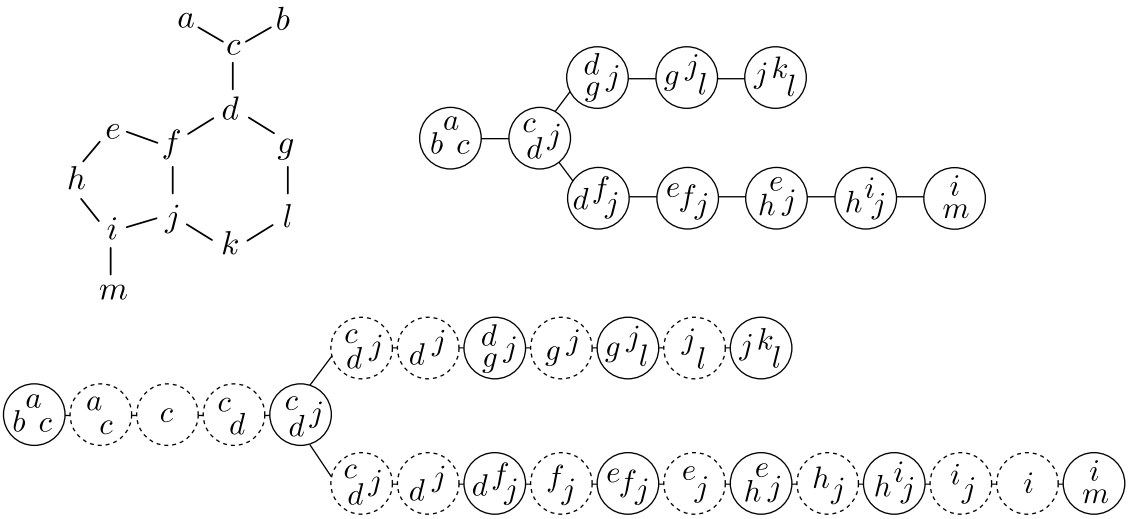

Fig. 4. Molecule graph of adenine (upper left, atom types are neglected), its tree decomposition (right), and nice tree decomposition (bottom, new bags are drawn dashed). The root of the nice tree decomposition is the left-most bag.

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 below each 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 $D_i[a_1, \ldots, a_k;\, e_{1,2}, \ldots, e_{k-1,k}]$ be the minimum score over all valence assignments to the vertices in $Y_i \setminus X_i$ if for every $l = 1, \ldots, k$, exactly a_l valences of vertex v_l have been consumed by the edges between v_l and vertices in $Y_i \setminus X_i$, and bond orders $e_{1,2}, \ldots, e_{k-1,k}$ 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 $\min_{a_1}\{s_{v_r}(a_1) + D_r[a_1]\}$.

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 $X_j = \{v_1\}$ 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 the parent node of X_j such that $X_j = \{v_1, \ldots, v_{k-1}\}$ and $X_i = \{v_1, \ldots, v_k\}$. Then

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

Figure 5 illustrates the execution of the algorithm at an introduced node.

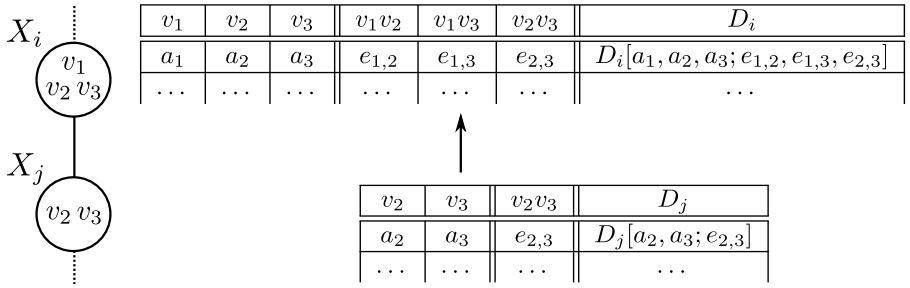

Fig. 5. X_i is an introduce node. After computing D_j-values, the algorithm computes D_i-values using the recursion (2).

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

$$D_i[a_1, \ldots, a_{k-1};\, e_{1,2}, \ldots, e_{k-2,k-1}] = \min_{\substack{e_{1,k}, \ldots, e_{k-1,k} \in \{0,1,2\} \\ a_k \in \{0, \ldots, A^*_{v_k}\}}} \left\{ s_{v_k}\left(a_k + \sum\nolimits_{l=1}^{k} e_{l,k}\right) + D_j[a_1 - e_{1,k}, \ldots, a_{k-1} - e_{k-1,k}, a_k;\, e_{1,2}, \ldots, e_{k-1,k}] \right\} \quad (3)$$

Figure 6 illustrates the execution of the algorithm at a forget node.

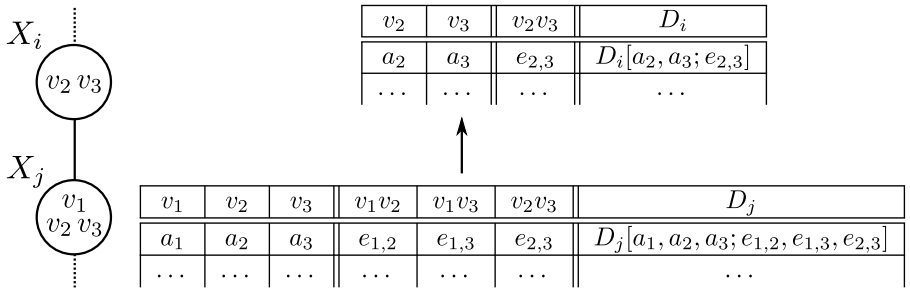

Fig. 6. X_i is a forget node. After computing D_j-values, the algorithm computes D_i-values using the recursion (3).

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

$$D_i[a_1, \ldots, a_k; e_{1,2}, \ldots, e_{k-1,k}] = \min_{\substack{a'_l=0,\ldots,a_l \\ \text{for } l=1,\ldots,k}} \left\{ D_j[a'_1, \ldots, a'_k;\, e_{1,2}, \ldots, e_{k-1,k}] + D_h[a_1 - a'_1, \ldots, a_k - a'_k;\, e_{1,2}, \ldots, e_{k-1,k}] \right\} \tag{4}$$

See Figure 7 for an illustration.

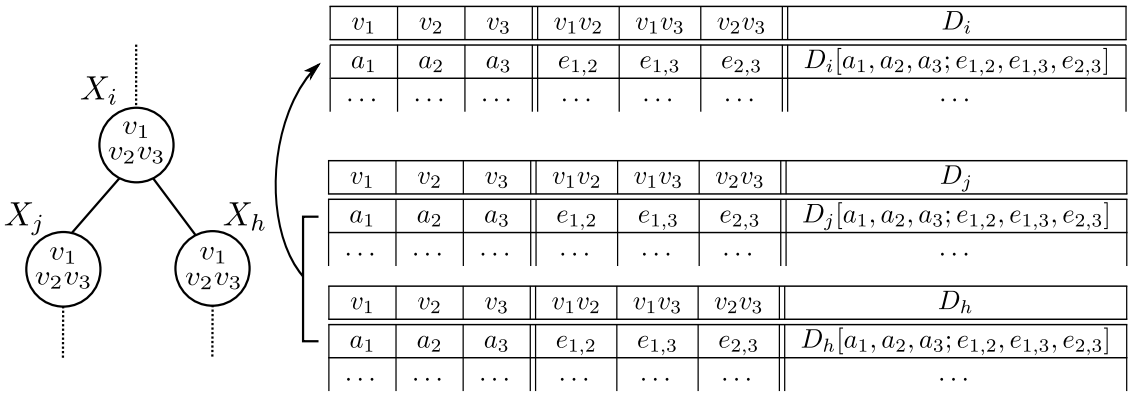

Fig. 7. X_i is a join node. After computing D_j- and D_h-values, the algorithm computes D_i-values using the recursion (4).

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

Proof. We first analyze the running time of the algorithm. Obviously, the number of bonds that can be assigned to a vertex is α. Therefore, there are at most α^ω possibilities to assign bonds to ω vertices in a bag of T. According to the definition of β, the number of edges induced by vertices in one bag of T is bounded by β, and there are at most 3^β possibilities to assign bond orders $b \in \{0, 1, 2\}$ to β edges. This implies that the table D_i of a node X_i contains at most $\alpha^\omega \cdot 3^\beta$ entries. We now consider the running time of computing each entry in the matrix D_i, assuming that the corresponding matrices of every child of X_i have already been calculated. We distinguish the following cases: If X_i is a leaf of T, computing matrices D_i takes constant time, since entries with score infinity do not have to be considered, and there is only one entry with score zero. If X_i is an introduce node, calculating each entry of D_i takes constant time. Again, we do not have to consider entries with score infinity.

If X_i is a forget node and X_j its child node, for naively calculating each entry of matrix D_i, we have to test all entries of the matrix D_j to find the minimum score as shown in (3). But the minimum score in (3) can be calculated on-the-fly when computing D_j: For each fixed index $(a_1, \ldots, a_{k-1})$, we compute and store the minimum score $s_{v_k}(a_k + \sum_{l=1}^{k} e_{l,k}) + D_j[a_1 - e_{1,k}, \ldots, a_{k-1} - e_{k-1,k}, a_k; e_{1,2}, \ldots, e_{k-1,k}]$ in the corresponding item in D_i. This only increases the running time for computing the matrix D_j by a constant factor. With this preprocessing, calculating each entry of D_i can be done in constant time.

Let X_i be a join node and X_j and X_h its children. To calculate an entry of D_i, the algorithm has to find a "partner entry" in D_h for every entry of D_j, such that the number of consumed bonds of a vertex in X_i is the sum of the consumed bonds of the same vertex in X_j and X_h, and calculate the minimum score of all such pairs, as shown in (4). Therefore, computing an entry of the matrix D_i can be done in time $O(\alpha^\omega \cdot \omega)$.

Now, the tree decomposition contains $O(m)$ nodes, the matrix of every node contains at most $\alpha^\omega \cdot 3^\beta$ entries. Computing each matrix entry takes time $O(\alpha^\omega \cdot \omega)$. Moreover, initializing the matrices of all leaves of the tree decomposition takes $O(m)$ time. In total, the running time of the algorithm is $O(\alpha^{2\omega} \cdot 3^\beta \cdot \omega \cdot m)$.

In the following, we prove the correctness of the algorithm. As additional forget nodes above the root of the tree decomposition are introduced until the new root contains a single vertex, and every vertex except the vertex in the new root has to be "forgotten" once. Note that because of the third property of tree decompositions, a vertex cannot be forgotten more than once. Whenever a vertex is "forgotten", it gets the valence that equals the sum of bonds it used up in the subtree below the child of the forget node, plus the bonds it used up inside this child node. The score assigned to the forget node in (3) is the minimum over all sums of the score of the sum of the consumed bonds and the bond orders of all edges between v_k and its neighbors, and the score of the corresponding entry in D_j where bond orders of edges between v_k and its neighbors are subducted.

Considering every leaf of T as an introduce node, we can see that a newly introduced vertex has not used up any of its bonds yet. Therefore, it is correct to set the score of entries, where a newly introduced vertex already uses bonds, to infinity. The correctness of the algorithm at introduce nodes is also obvious.

A vertex can be introduced more than once, but because of the third property of tree decompositions, there must be one join node that joins all occurrences of this vertex. At each join node, only two occurrences of a vertex are joined. Let v be a vertex of the molecule graph that is introduced twice. On the two paths from the corresponding introduce nodes to the join node, where two occurrences of v are joined, each occurrence of v may consume different amounts of bonds. Note that this can only happen if different vertices are forgotten on the two paths. Therefore, the total amount of consumed bonds of v in the subtree below the join node is the sum of consumed bonds in the subtrees below the children of the join node. Since we are interested only in the optimal solution, we only take the minimum score as shown in (4).

When the algorithm arrives at the root r of the tree decomposition, it holds that every vertex of the molecule graph has been considered, and the scores that correspond to the valences assigned to each vertex have been summed up in the corresponding entry in D_r. Except for the only vertex v_r in the root r, every vertex has been forgotten on some path in the tree, and the feasible valence with minimum score is assigned to the vertex. This means that the validity and optimality of the assignment for the subgraph

$G \setminus v_r$ of G is assured. Therefore, $\min_a s_{v_r}(a) + D_r[a]$ is the minimum score and thus the optimal solution for the Bond Order Assignment problem on G.

All in all, our algorithm computes an optimal bond order assignment of G using a nice tree decomposition of G in time $O(\alpha^{2\omega} \cdot 3^{\beta} \cdot \omega \cdot m)$. □

If, for forget nodes and join nodes, we also store where the minimum is obtained in (3) and (4) during the bottom-up processing, we can traverse the tree decomposition top-down afterwards to obtain all bond orders of the molecule. This can be done in time $O(m)$. We can also enumerate all optimal solutions, or slightly suboptimal solutions, by backtracking through the dynamic programming tables, what requires $O(\alpha^{\omega}\omega m)$ time per solution.

4.3 The $O(\alpha^{3\omega} \cdot \omega \cdot m)$ algorithm

The idea for this version of the algorithm is based on the observation that the information about the bond order assigned to each edge in a bag of the tree decomposition is not really necessary, but the number of bonds of an atom used up by edges within a bag of the tree decomposition is more important. To make use of this observation, we modify our algorithm described in Section 4.2 as follows:

Let $v_1, \ldots, v_k$ denote the vertices in a bag X_i, and let $D_i[a_1, \ldots, a_k; b_1, \ldots, b_k]$ be the minimum score over all valence assignments to the vertices in $Y_i \setminus X_i$ if, for every $l = 1, \ldots, k$, exactly a_l valences of vertex v_l have been consumed by edges between v_l and vertices in $Y_i \setminus X_i$, and b_l valences of vertex v_l are consumed by edges within the bag X_i. Recall that Y_i is the set of atoms occurring in the subtree rooted at X_i. Again, our algorithm starts at the leaves of the tree decomposition and computes the score matrix D_i for every node X_i when score matrices of all its child nodes have been computed. The score of the optimal bond order assignment is $min_{a_1}\{s_{v_r}(a_1) + D_r[a_1]\}$, where v_r is the only vertex in the root bag X_r of the tree decomposition.

We initialize the matrix D_j of each leaf $X_j = \{v_1\}$ with

$$D_j[a_1; b_1] = \begin{cases} 0 & \text{if } a_1 = b_1 = 0, \\ \infty & \text{otherwise.} \end{cases}$$

We distinguish if a bag X_i is an introduce node, a forget node, or a join node and use the corresponding recurrence to calculate D_i:

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

$$D_i[a_1, \ldots, a_k;\, b_1, \ldots, b_k] = \min_{\substack{e_1,\ldots,e_{k-1} \in \{0,1,2\} \\ \sum_l e_l = b_k}} \begin{cases} D_j[a_1, \ldots, a_{k-1};\, b_1 - e_1, \ldots, b_{k-1} - e_{k-1}] & \text{if } a_k = 0, \\ \infty & \text{otherwise.} \end{cases} \quad (5)$$

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

$$D_i[a_1, \ldots, a_{k-1};\, b_1, \ldots, b_{k-1}] = \min\Big\{ s_{v_k}(a_k + b_k) + D_j[a_1 - e_1, \ldots, a_{k-1} - e_{k-1}, a_k;\, b_1 + e_1, \ldots, b_{k-1} + e_{k-1}, b_k] \Big\} \quad (6)$$

where the minimum runs over all $e_1, \ldots, e_{k-1} \in \{0,1,2\}$ such that $\sum_{l=1}^{k-1} e_l = b_k$, and all $a_k = 0, \ldots, A^*_{v_k}$.

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

$$D_i[a_1, \ldots, a_k;\, b_1, \ldots, b_k] = \min_{\substack{a'_l=0,\ldots,a_l \\ \text{for } l=1,\ldots,k}} \Big\{ D_j[a'_1, \ldots, a'_k;\, b_1, \ldots, b_k] + D_h[a_1 - a'_1, \ldots, a_k - a'_k;\, b_1, \ldots, b_k] \Big\}. \quad (7)$$

Lemma 3. *Given a nice tree decomposition of a molecule graph G, the algorithm described above computes an optimal assignment for the* Bond Order Assignment *problem on G in time $O(\alpha^{3\omega} \cdot \omega \cdot m)$, where $\alpha = 1 + \max_v A^*_v$, and m and $\omega - 1$ are size and width of the tree decomposition.*

Proof. We first analyze the running time of our algorithm. Since each bag that is a leaf of the tree decomposition contains only one vertex, initializing the table of a leaf takes constant time, and initializing all leaves of the tree decomposition takes time $O(m)$. We now consider the running time of the algorithm at each inner node X_i of the tree decomposition. Obviously, the table D_i of each bag X_i contains at most $\alpha^{2\omega}$ items. Let X_j denote the child node of X_i, if X_i is an introduce node or a forget node, and X_j and X_h denote the children node of X_i if X_i is a join node.

At an introduce node X_i, let v_k be the newly introduced vertex and $e_l \in \{0,1,2\}$ denote the bond order of an edge $v_l v_k$. When calculating an item $D_i[a_1, \ldots, a_k;\, b_1, \ldots, b_k]$, the algorithm considers every item $D_j[a_1, \ldots, a_{k-1};\, b'_1, \ldots, b'_{k-1}]$ with fixed indices $a_1, \ldots, a_{k-1}$ and $b'_l = b_l - e_l$ for $1 \le l \le k-1$ and $\sum_{1 \le l \le k-1} e_l = b_k$. Since there are at most $\alpha^{\omega-1}$ such items in D_j and testing if $b'_l = b_l - e_l$ for $1 \le l \le k-1$ takes time $O(\omega)$, calculating an item of D_i takes time $O(\alpha^\omega \cdot \omega)$. Therefore, the table of an introduce node can be calculated in time $O(\alpha^{3\omega} \cdot \omega)$.

Let X_i be a forget node where vertex v_k is forgotten. Again, let $e_l \in \{0,1,2\}$ denote the bond order of an edge $v_l v_k$. To analyze the running time of the algorithm at a forget node, we describe the execution of our algorithm at a forget node in detail. To calculate an item $D_i[a_1, \ldots, a_{k-1};\, b_1, \ldots, b_{k-1}]$, the algorithm tests for all possible valences $a'_1, \ldots, a'_{k-1}, a'_k$ if $e_l = a_l - a'_l \in \{0,1,2\}$ holds for all $1 \le l \le k-1$. If this is true, the algorithm sets $b_k := \sum_{1 \le l \le k-1} e_l$ and computes the score $s_{v_k}(a'_k + b_k) + D_j[a'_1, \ldots, a'_{k-1}, a'_k;\, b_1 + e_1, \ldots, b_{k-1} + e_{k-1}, b_k]$. This can be done in time $O(\omega)$. The minimum score over all such scores is assigned to $D_i[a_1, \ldots, a_{k-1};\, b_1, \ldots, b_{k-1}]$. Since there are at most α^ω possibilities of indices $a'_1, \ldots, a'_{k-1}, a'_k$, calculating an item of a forget node can be done in time $O(\alpha^\omega \cdot \omega)$. Therefore, the running time of our algorithm at a forget node is bounded by $O(\alpha^{3\omega} \cdot \omega)$.

When calculating an item $D_i[a_1, \ldots, a_k;\, b_1, \ldots, b_k]$ of a join node, the algorithm has to test for each item $D_j[a'_1, \ldots, a'_k;\, b_1, \ldots, b_k]$ in D_j, if this item and its partner $D_h[a_1 - a'_1, \ldots, a_k - a'_k;\, b_1, \ldots, b_k]$ in D_h minimize score at $D_i[a_1, \ldots, a_k;\, b_1, \ldots, b_k]$. Since there are at most α^ω items in D_j with fixed indices $b_1, \ldots, b_k$, calculating an item of the table of a join node takes $O(\alpha^\omega \cdot \omega)$, and thus the running time for calculating the table of a join node is bounded by $O(\alpha^{3\omega} \cdot \omega)$. In total, since the tree decomposition contains m nodes, the running time of our modified algorithm is bounded by $O(\alpha^{3\omega} \cdot \omega \cdot m)$.

Next, we prove the correctness of our algorithm. The initialization at the leaves of the tree decomposition is obviously correct, since no valence of any vertex is used up at

this stage. At an introduce node, no valence of the newly introduced vertex v_k is used up in the subtree rooted at this introduce node. Furthermore, each bond order e_l between v_k and a vertex v_l in the introduce node increases the number of consumed valences of v_l from $b_l - e_l$ to b_l, and the bond orders of all edges between v_k and other vertices in this node sum up to b_k valences of v_k, that are consumed within this node. This intuition confirms the correctness of our algorithm at introduce nodes.

Let X_i be a forget node and v_k be the vertex that is forgotten at X_i. Since v_k is forgotten, we increase the number of used up valences of each vertex $v_l \in X_i$ from $a_l - e_l$ to a_l. Since v_k does not occur in X_i, we reduce the valences of vertices in X_i, which are consumed by bonds within X_i, by the bond order of bonds between these vertices and v_k. Furthermore, the algorithm also assigns v_k the valence minimizing the corresponding item in D_i. Therefore, our algorithm is correct at forget node.

At a join node X_i, the total number of valences consumed outside X_i of a vertex results from the number of valences of this vertex that are consumed in the subtree rooted at X_j and the subtree rooted at X_h. Recall that $Y_j \setminus X_j$ and $Y_h \setminus X_h$ are disjoint. This confirms the correctness of our algorithm at join nodes.

The correctness of this algorithm at the root of the tree decomposition is analogous to the correctness of our previous algorithm introduced in Section 4.2.

All in all, our algorithm compute the optimal solution of the Bond Type Assignment in time $O(\alpha^{3\omega} \cdot \omega \cdot m)$. □

To compute not only the optimal score but also the optimal assignment, we again store where the minimum is obtained for forget nodes and join nodes during the bottom-up processing. We then traverse the tree decomposition top-down to obtain all bond orders of the molecule in time $O(m)$. Again, we can enumerate optimal or suboptimal solutions.

From the theoretical point of view, this algorithm is an important improvement of the algorithm introduced in Section 4.2. Whereas the running time of the algorithm in Section 4.2 exponentially depends on the square of treewidths of molecule graphs, the running time of this algorithm only exponentially depends on treewidths of molecule graphs.

In practice, this algorithm could be more efficient if subgraphs induced by vertices in a bag of the tree decomposition are dense and the maximum valence of atom in the molecule graph is small. However, this does not usually occur, therefore we only implemented the algorithm with running time $O(\alpha^{2\omega} \cdot 3^{\beta} \cdot \omega \cdot m)$ introduced in Section 4.2.

5 Algorithm engineering

In this section, we describe a few heuristic improvements we included in the implementation of the algorithm introduced in Section 4.2.

Instead of directly computing the optimal solution, we use our algorithm to test if the score of the optimal solution exceeds an integer $k \geq 0$. In case the score of the optimal solution is at most k, the algorithm will find the optimal solution. Otherwise, we repeat calling our algorithm with increasing k, until the optimal solution is found. By doing this, we forbid all valences of atoms that have score larger than k. Furthermore, we do not store entries of D matrices with score exceeding k. Since the scores of the optimal solutions are usually very small in practice, this strategy accelerates the performance of our algorithm drastically.

During the course of the dynamic programming algorithm, 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 $a_l + \sum_j b_{l,j} > A_l^*$ for some l, 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.

6 Computational results

To compute optimal tree decompositions of molecule graphs, we use the method QuickBB in the library LibTW implemented by van Dijk *et al.* (`http://www.treewidth.com`). We transform the computed optimal tree decompositions into nice tree decompositions.

Although the running times of our algorithms depend (super-)exponentially on the treewidth, we claim that we can efficiently solve practical instances of the Bond Order Assignment problem. We justify our claim by the following observation: A graph is *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 of the boundary face, the remaining graph is $(r-1)$-outerplanar. Now, every r-outerplanar graph has treewidth at most $3r - 1$ [5]. Together with our algorithms, it holds that the Bond Order Assignment problem is solvable in polynomial time on r-outerplanar molecule graph with fixed r and fixed maximum valence.

Moreover, we find that molecule graphs of biomolecules are usually r-outerplanar for some small integer r, such as $r = 2$ for proteins and DNA.

To empirically confirm our claim, we tested it on molecules from the PubChem database at `http://pubchem.ncbi.nlm.nih.gov/` [16], which contains more than 60 million entries in Jan 2010. We computed the treewidths of all molecule graphs in eight files randomly chosen from 1 782 files found at `ftp://ftp.ncbi.nlm.nih.gov/pubchem/Compound/CURRENT-Full/XML`. For all 135 607 connected molecule graphs in these files, we computed the exact treewidth using the QuickBB method of the LibTW library. We found that 12 004 (8.85%) molecule graphs have treewidth one, 121 267 (89.43%) have treewidth two, 2 192 (1.62%) have treewidth three, and for seven (0.01%) molecules, the QuickBB method cannot determine the treewidth after ten minutes of computation. According to the upper bound computed by the QuickBB method, the treewidth of these seven molecule graphs is at most four. The database also contains 137 (0.1%) molecules consisting of a single ion, for which the Bond Order Assignment problem is trivial. There are no molecule graphs in the eight files with treewidth exceeding four.

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 evaluation, we used the MMFF94 dataset[3] by Halgren *et al.* [9], which consists of 761 molecule graphs predominantly derived from the Cambridge Structural Database. This dataset has been suggested to us by experts, as it is considered to contain "hard" instances of the problem, where atoms have non-standard valences. Bond orders are given in the dataset but we ignored this information and reassigned the bond orders to all molecule graphs. We removed four molecule graphs that contain elements such as iron not covered in our scoring table (see below), or that have atom bindings

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

instance size		number of		average	running time		average
$\|V\|$	$\|E\|$	instances	treewidth	treewidth	TD	DP	# solutions
3–10	2–11	73	1–2	1.15	0.3	0.3	1.32
11–20	10–22	214	1–3	1.77	0.47	1.71	1.57
21–30	20–33	333	1–3	1.97	0.74	2.97	1.81
31–40	30–43	129	1–3	1.95	0.91	14.58	2.22
41–50	40–53	5	1–2	1.8	1.4	3.8	4.8
51–59	53–61	3	2	2.0	1.33	5	6

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.

such as chlorine atoms connected to four other atoms, which is also not covered in our scoring. The largest molecule graphs contains 59 atoms, the smallest 3 atoms, the average 23 atoms. We find that 20.21 % of the remaining 757 molecule graphs have treewidth one, 96.69 % have treewidth ≤ 2, and all molecule graphs have treewidth at most three. The average treewidth is 1.83.

For scoring assignments, we use the scoring table from Wang *et al.* [17]. 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 [17] for details.

See Table 1 for computational results. Total running times are always below one second, and 5 ms on average.

We compared the solutions computed by our algorithm to the solutions computed by Antechamber[4], a software package for computing molecular mechanics force fields, which also implements the heuristic by Wang *et al.* [17] for computing bond order assignments minimizing penalty scores. While our algorithm always computed the bond order assignment with minimum penalty score, Antechamber was not able to find the optimal solutions for nine molecules in the data set. In some cases, our algorithm found up to 13 optimal solutions because of symmetries and aromatic rings in the molecule graph. Furthermore, although being an exact algorithm, our algorithm is almost as fast as the heuristic algorithm of Wang *et al.* [17] and an order of magnitude faster than the exact algorithms in [6] when applying to biomolecules. Detailed comparison of running time with [17] and [6] is in preparation.

7 Conclusion

We considered the problem of assigning bond orders to a molecule graph and showed that the problem is NP-hard on molecule graphs with bounded vertex degrees and bounded valences, but can be solved in polynomial time if the molecule graph is a tree. Furthermore, we also proved MAX SNP-hardness of the maximization version of the problem. Based on the tree decomposition concept, we introduced two dynamic programming algorithms

[4] `http://ambermd.org/antechamber/antechamber.html`

with running time practically linear in the size of the molecule. In contrast to previous approaches, our algorithms compute exact solutions in a guaranteed running time. We expect that the algorithms can be applied to large molecules if the molecule graph has small treewidth.

In the future, we want to evaluate the quality of solutions and the running time of our algorithm against other approaches. In particular, we want to verify that our algorithm finds more chemically or biologically relevant solutions than heuristic approaches. Note that the quality of the solution depends, for the most part, on the quality of the underlying scoring table. From the theoretical point of view, it might be interesting to investigate the approximability of the maximization version of the Bond Order Assignment problem.

Acknowledgment

We thank Anna K. Dehof and Andreas Hildebrandt for introducing us to the problem, and Johannes Uhlmann for helpful discussions. Implementation by Kai Dührkop and Patric Seeber. Antechamber results and scoring tables provided by A. Dehof. Q.B.A. Bui gratefully acknowledges financial support from the Deutsche Forschungsgemeinschaft, research group "Parameterized Algorithms in Bioinformatics" (BO 1910/5).

References

1. S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embedding in a k-tree. *SIAM J. Algebra. Discr.*, 8:277–284, 1987.
2. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.
3. J. C. Baber and E. E. Hodgkin. Automatic assignment of chemical connectivity to organic molecules in the cambridge structural database. *J. Chem. Inf. Model.*, 32:401–406, 1992.
4. P. Berman, M. Karpinski, and A. D. Scott. Computational complexity of some restricted instances of 3-SAT. *Discrete Appl. Math.*, 155:649–653, 2007.
5. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998.
6. A. K. Dehof, A. Rurainski, H.-P. Lenhof, and A. Hildebrandt. Automated bond order assignment as an optimization problem. In *Proc. of German Conference in Bioinformatics (GCB 2009)*, Lecture Notes in Informatics, pages 201–210. Gesellschaft für Informatik, 2009.
7. R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, Berlin, 1999.
8. M. Froeyen and P. Herdewijn. 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.
9. T. A. Halgren. MMFF VI. MMFF94s option for energy minimization studies. *J. Comp. Chem.*, 17(5-6):490–519, 1996.
10. M. Hendlich, F. Rippmann, and G. Barnickel. BALI: automatic assignment of bond and atom types for protein ligands in the brookhaven protein databank. *J. Chem. Inf. Model.*, 37:774–778, 1997.
11. T. Kloks. *Treewidth, Computation and Approximation.* Springer, Berlin, 1994.
12. P. Labute. On the perception of molecules from 3D atomic coordinates. *J. Chem. Inf. Model.*, 45(2):215–221, 2005.
13. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.
14. C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. System Sci.*, 43(3):425–440, 1991.
15. N. Robertson and P. Seymour. Graph minors: algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.
16. E. W. Sayers, T. Barrett, D. A. Benson, E. Bolton, S. H. Bryant, K. Canese, V. Chetvernin, D. M. Church, M. Dicuccio, S. Federhen, M. Feolo, L. Y. Geer, W. Helmberg, Y. Kapustin, D. Landsman, D. J. Lipman, Z. Lu, T. L. Madden, T. Madej, D. R. Maglott, A. Marchler-Bauer, V. Miller, I. Mizrachi, J. Ostell, A. Panchenko, K. D. Pruitt, G. D. Schuler, E. Sequeira, S. T. Sherry,

M. Shumway, K. Sirotkin, D. Slotta, A. Souvorov, G. Starchenko, T. A. Tatusova, L. Wagner, Y. Wang, W. J. Wilbur, E. Yaschenko, and J. Ye. Database resources of the national center for biotechnology information. *Nucleic Acids Res.*, 38(Database issue):D5–16, 2010.

17. J. Wang, W. Wang, P. A. Kollmann, and D. A. Case. Automatic atom type and bond type perception in molecular mechanical calculations. *J. Mol. Graph. Model.*, 25:247–260, 2006.