Automatic Conversion of Molecular Models

By
Ahmad Shahwan

Supervised by
Stephane Redon

A Thesis Submitted in Partial Fulfillment of the Requirements of the
Degree of Master of Research in Informatics
Université Joseph Fourier
September 10, 2010

Acknowledgment
The author takes this chance to express his gratitude to his supervisor, Stephane Redon, without his assistance and guidance this work would not have been possible. He also conveys thanks to all members of the NANO-D team, for their valuable support throughout this project.
Deepest gratitude is also due to INRIA Rhône-Alpes for the generous support that made this work possible.

Abstract Molecular mechanics are powerful tools to predict and simulate physical phenomena at the nano-level, such approaches are defined by forcefields models that describes the possible topology of the molecular system, as well interactions between its interior parts, in terms of potential energies and forces as a function of the systems geometrical conformation. In order to apply a model to a nano-system, it must thus be ensured that the topology of such a system conforms to that prescribed by the model.

However, as geometrical properties of nano-systems might be arbitrarily defined by the user, or be obtained experimentally, or even come from a predefined system, a method to automatically detect topological and geometrical patterns suggested by a model in a given nano-system must exist.

This research comes to fill this gap, by introducing a general algorithm and data structure that are capable of finding predefined patterns –for which forcefields parameters are known in advance– in a given nano-system, enabling the seamless assignment of one molecular model or other.

Résumé La mécanique moléculaire est un puissant outil pour prévoir et simuler les phénomènes physiques à l'échelle nanoscopique. Cette approche repose sur des modèles de champs de forces qui décrivent la topologie possible du système moléculaire, ainsi les interactions entre ses parties, en termes d'énergies potentielles et de forces qui sont fonction de la conformation géométrique des systèmes. Afin d'appliquer un modèle à un nano-système, il faut donc veiller à ce que la topologie d'un tel système soit conforme à celle prévue par le modèle.

Toutefois, comme les propriétés géométriques des nano-systèmes peuvent être arbitrairement définies par l'utilisateur, ou être obtenues expérimentalement, ou encore provenir d'un système prédéfini, une méthode permettant de détecter automatiquement les modèles topologiques et géométriques suggéré par le champ de forces dans un nano-système donné doit exister.

Cette recherche vient combler cette lacune, en introduisant une structure de données et un algorithme général qui sont capables de trouver des modèles prédéfinis –pour lesquels les paramtres des champs de forces sont connus à l'avance– dans un nano-système donné, afin de permettre l'affectation d'un champ de forces à un nano-système.

Contents

Chapter 1

Introduction

Last few decades have brought to the body of human knowledge a great deal of new achievements. Those on turn spawned new scientific branches, breaking barriers that kept us for ages from answering fundamental questions, and solving vital problems. Amongst such accomplishments pioneers the ability to scale down to the tiniest levels of perception of the world, a thing that gave us a better understanding of the environment we are living within, allowing for control over challenging physical phenomena.

Nanotechnology, a science brought to existence in the second half of the last century, was enabled by earlier scientific realizations such as quantum mechanics. Today, this domain is promising new openings with diverse application amongst which are contributions to the simulation of biological phenomena such as protein folding, the engineering of new materials on the molecular lever, the construction of nano-devices, and the synthesis of life[1].

1.1 Approaches to Molecular Modeling

To enable such applications, different approaches exist to study the interaction between particles on the nano-level (mainly, between atoms inside a molecule, a complex, or a nano-system). In general, such approaches try to define the energy of a nano-system as a function of its geometrical configurations, in order to simulate the nano-system dynamics, and predict stable states, that is those whose energies are minimal.

On one extreme, the ab initio approach is highly based on theoretical background, chiefly, quantum mechanics equations, which are used to derive molecular properties. Unfortunately, the resolution of such equations involves heavy calculations that make it difficult to apply such a technique for polyatomic molecules[3]. Thus, approximations had to be introduced to relax such complexities. Albeit those approximations, ab initio approaches are still limited to systems with only few number of atoms (up to 20 according to [3]).

Semi-empirical approaches have been proposed to allow for further relaxation of computational and spatial complexity. One way to do so is by suggesting additional approximations that cut short heavy calculations. Another approach is to substitute a large portion of complicated computations with values derived from experimental results. Semi-empirical models allow for the resolution of larger-scale molecules (up to 100 atoms). However, they are still too resource-intensive for big molecules and complexes such as proteins and drugs. That means that additional simplifications should come into effect before we can consider such systems.

Molecular mechanics, also referred to as forcefield methods, come to fill this gap. Such approaches relax costly computations in favor of experimentally derived parameters. In a matter of fact, molecular mechanics replaces heavy quantum mechanic calculations by a mathematical model, based on Newtonian physics to predict molecular properties. They look at a nano-system as an assembly of atoms that seeks an equilibrium state that minimizes its steric energy[1] by changing bond lengths, bond angles and dihedral angles. Besides system geometrical settings in the three-dimensional space, potential energies are calculated based on the atom types and hybridization and bond orders.

Molecular mechanics forcefields come in flavors, called forcefields. Each model works fine for problems belonging to certain category, the empirical nature of such model makes it harder to be generalized, and strongly associates each model to a specific application. For instance, AMBER forcefield is usually used for proteins and DNA, CHARMM[4] is widely used for macro-molecules, OPLS is designed for liquid simulation, while ReaxFF is meant for simulating chemical reactions. Despite attempts to come up with a unified model, e.g. the UFF, a Universal Force Field model[2], the high dependency of estimated parameters value per model on the nature of molecular system at hands has always kept those approaches from being widely accepted.

1.2 Problem and Motivation

As mentioned earlier, molecular mechanics provide a feasible solution for large molecular systems simulation and resolution. However, such a solution may not be unique. One drawback that molecular mechanics may impose is the fact that models are usually closely related to a certain application, leading to a vast variety of forcefields, which leads in turn to more than one possible solution for the same system. Comparing those solution enables a more enlightened understanding of system's behavior. However, forcefields may vary diversely in the way they represent data. For example,

[1] Steric energy is the difference in energy between the system in a given geometrical settings, and the system in an idle state.

while CHARMM19 uses united-atom, i.e. it may represent more than one atom as a single particle, CHARMM22 uses all-atom, representing each and every atom as a particle, in a different approach VAMM[5] forcefield uses a coarse-grained structure of protiens, based on contact information between residues. This diversity makes the assignment of different forcefields to a single molecular system a tedious job.

The assignment of one forcefield parameter to a given molecular system represented in another forcefield enables studying the system at hand from different perspectives. However, the above mentioned issues make the conversion between one forcefield model into another a burden.

In this work, we suggest a method allowing the automatic assignment of a particular forcefield to a given molecular system, to enable automatic conversion between forcefield models. The basic idea is to match patterns suggested by the model (the forcefield) in the given representation of the molecular system, then once the mapping is done, and the system is decomposed into smaller patterns, precomputed particle values for the pattern are transmitted to their respective matches in the given system. The proposed algorithm and data structure are kept as general as it could be, matching criteria are meant to be invariant to the model, allowing the use of the same algorithm for all forcefields.

Chapter 2

Literature Overview

As demonstrated in the first chapter, this work falls under the class of pattern recognition, where certain patterns are looked for into some input data. Amongst others, graph matching techniques are often efficiently applied in this particular category of computer science. Along with the fact that our patterns, as well as input data, are naturally presented in graph-like data structures, this makes those technique an intuitive choice to handle the problem at hand.

In this chapter, we will first demonstrate some basic concepts that are central to the graph matching problem, then different approaches to tackle this problem are presented, addressing the pros and cons of each.

2.1 The Graph Matching Problem

In the rest of this chapter, we will consider a graph as a triple $G(V, E, \lambda)$ where V is the set of graph vertices, $E \in V \times V$ is the set of graph edges, and $\lambda : V \to L$ is a labeling function that associates a label to each vertex, where L is the label alphabet.

The graph matching problem in its conceptually simplest forms is to answer the question of whether or not an isomorphism exists between two graphs, and in the case of a positive answer, to find that isomorphism. That is given two graphs $G_1(V_1, E_1, \lambda_1)$ ans $G_1(V_2, E_2, \lambda_2)$ we are looking for a bijection $\phi : V_1 \to V_2$ that satisfies the following;

$$\lambda(v) = \lambda(\phi(v)),$$

and

$$(v, w) \in E_1 \iff (\phi(v), \phi(w)) \in E_2.$$

This case is referred to as graph isomorphism or exact graph matching. However, there are only few application where finding and exact isomorphism is important. A more general problem is to be able to tell whether

one graph is completely contained in another graph, that is, whether or not there exists a subgraph of the later that is isomorphic to the former. This problem is called exact subgraph matching, and it has many application in different domains, however, the precise nature of isomorphism makes this approach too strict for error prone application.

A more tolerant approach is to estimate graph similarity based on a distance function between two graphs. Two basic approach to the distance function exist, the fist is to find what is called the maximal common subgraph between two graphs, and calculate similarity based on its relative size to the matched graphs. As the name suggests, the maximal common subgraph is the largest maximum common subgraph, which is a common subgraph that is contained in no other common subgraph, a common subgraph is, in turn, a subgraph of one of the two graphs, that is isomorphic to a subgraph of the other.

The the second approach is to calculate the edit distance between two graphs, edit distance (as borrowed from Hamming distance in information theory) is the minimal cost needed to transform one graph into another. Transformation is done through a series of steps of addition, deletion, or substitution of nodes or edges, a cost function associates each step with a positive number, then the total cost is evaluated by summing individual costs over all required steps. An equivalence is shown to exist between the relevant size of the common maximum subgraph and the edit distance, under specific classes of cost functions[15].

When applied to pattern recognition, graph matching is typically the problem of finding a subgraph of the data graph that best fits a given pattern graph, with a certain extent of error tolerance. Fitness is usually estimated according to a metric that measures the difference between the pattern and the corresponding subgraph.

2.2 Early Approaches

Because of its centric importance to vast variety of theoretical and applied domains, the problem of graph matching was heavily studied, a rich literature exists confronting to this issue. Efforts were devoted fist to tackle the problem of graph isomorphism and the maximum common subgraph finding. Barrow and Burstall[13] reduce the problem of maximum/maximal common subgraph to the problem of maximum/maximal clique. The later problem addresses the detection of maximal clique in a graph, a clique is a fully connected subgraph, while a maximal clique is a clique that is not a subgraph of any other clique, the maximum clique is the clique with the largest number of vetices (thus, it is maximal). This reduction was pointed out in the work of Massaro and Pellillo[12], in their work, they also build on Motzkin-Straus theorem[14] that linked the maximal clique detection to quadratic program-

ing. The problem of maximum clique detection is known to be NP-hard for arbitrary graphs[11], however, many approaches to alleviate this hardness were proposed, such approaches generally apply heuristic schemes and try to view the problem from an combinatorial optimization point of view, where exact result are compromised with efficient computations.

2.3 Graph Matching as Combinatorial Optimization Problem

In this section we review algorithms that viewed the problem of graph matching as an optimization problem, trying to either minimize the distance between a query and a data graph, or to maximize its relevance. Methods that fall in this category include genetic algorithms, the expectation-minimization process, decision trees, and estimation of distribution algorithms. They are usually based on a probabilistic model to estimate the goodness or the badness of a candidate solution.

Genetic search algorithms provide a useful framework for solving combinatorial optimization problems, they were frequently applied to graph matching, Cross, Wilson and Hancock show in their work[19] a novel approach in this context, they augment their approach with hill-climbing process, enlightened crossover, and an objective Bayesian measure for selection, to enhance populations and speed up convergence. In a similar approach, the work of Singh et al.[20] presents a genetic algorithm to match structural shapes based on geometrical relation between different parts. Auwatanamongkol[18] in turn apply graph matching by means of genetic algorithms to the image recognition problem.

Beside its spacial complexity, genetic algorithms are known to have many parameters to configure, which highly affect the soundness of its results. Estimation of distribution algorithms come to alleviate such drawbacks, and they seem to be a good fit for the graph matching problem as the work of Bengoetxea et al.[21] suggests, where authors aim to build a probabilistic model to estimate the correlation between candidate solutions in the population of selected individuals at each iteration (generation) of the algorithm.

An expectation-minimization process was applied to the graph matching problem in [16] where authors propose an iterative method based on probabilistic models to guess the configuration of the match that best fits a pattern with a subgraph. The same principle is applied in [17] where the goal is to construct a geometrically valid transformation between two 2D point sets, represented as relational graphs.

2.4 Graph Matching as Pattern Recognition Problem

Pattern recognition aims to detect models provided beforehand in some giving input data. Models, referred to as patterns, can be analyzed, indexed and restructure ahead of the detection time, to enhance the results of, and reduce the time needed by such a process.

Graph matching is frequently applied to pattern detection, where pattern and input data are presented in graph-like representations. In this case, the problem is to map pattern graphs to subgraphs of the input data that they best resemble. This is slightly different form the case of trying to fit individual subgraphs, each at once. The a-priori knowledge about patterns allows to analyze them ahead of time, this "off-line" analysis usually allows for a more time-efficient matching, with better results. Moreover, the existence of relatively big number of patterns makes it impractical to use the same techniques to match individual queries as those of matching multiple patterns.

One example of applying pattern recognition to graph matching is the work presented by Irniger and Bunke[22] where decision trees techniques are applied to enable fast graph matching. The idea is to reduce the number of patterns for which the goodness function is calculated based on early elimination of obvious negatives. To this end, a decision tree is built that classifies pattern graph according to their features. Features are the histograms of labels occurrences in graphs. This approach highly relies on the fact that patterns are vastly variant in term of nominal properties.

An innovative approach in applying graph matching to pattern recognition is the work of Tian et al.[6] where authors propose an approximate algorithm for subgraph matching for biological purposes. The algorithm is designed for large graph database search rather than individual peers matching, thus, it uses indexing to accelerate the search process. The basic idea is to divide each graph in the database into small structural units, those units are then indexed according to their nominal and topological properties.

Matches between query structural units and those of data are filtered out using the index, they are then assembled in bigger matches in a way that avoids conflicts. This is done by constructing compatibility graph that links each pair of non-conflicting small matches, larger matches are then induced by means of clique finding techniques. Finally, remaining candidates are filtered by performing a full fledged estimation of their distance to the data graph.

Chapter 3

Molecular Pattern Recognition

In this chapter, we address our approach to tackle the automatic molecular model conversion through molecular patterns recognition, applying graph matching techniques. We also demonstrate the analysis and design of the suggested algorithm, and its associated data structure.

3.1 General Overview

The sought algorithm takes as input a set of patterns that are expected to be found in a molecular system, along with a given data graph representing a nano-system, typically much larger than any individual pattern. Patterns are usually suggested by a certain molecular model. The algorithm tries to predict the occurrences of those patterns in the data graph, taking into account noise and data imperfection that may deform the graph. The final output of the algorithm is the data graph, now annotated with mutually exclusive matches of patterns.

Given the exact mapping between each particle (atom or assembly of atoms) in the system (a molecule or a complex) and its respective match in the corresponding pattern, the assignment of forcefield parameter becomes straightforward. Instead of calculating those parameter for each individual particle in a system, parameters are precomputed and stored for each pattern, then once the mapping is done, particles inherit those properties (the forcefield parameters) form their respective matches.

The work was achieved in three distinguishable phases, along which both the algorithm and the data structure maturated. The first phase, was the development of the core matching engine, which was aimed to find a pattern graph into another –typically larger– data graph. This work fell into the category of Approximate Graph Matching, and was highly influenced by existing literature which handled this problem nicely.

Next, the same engine had to be improved and adapted to our requirements, to give better results in a timely manner. Thus, particular domain knowledge had to be integrated into the algorithm to enhance the search, keeping it as general as possible for an arbitrary nanosystem. To this end, information about atoms position in the space were brought into account, creating a geometry-aware search engine, and dramatically reducing search time.

Finally, matches for all suggested patterns had to be put together and brought back to the user in a comprehensive layout. That meant that the data graph had to be decomposed into mutually exclusive patterns' matches suggested by the algorithm. In this phase many issues had to be dealt with, such as overlapping and priorities. Figure 3.1 sketches roughly our algorithm, summarizing the three phases.

3.2 SAGA, an Approximate Subgraph Matcher

The pattern recognition problem is frequently mapped to approximate subgraph matching, that is to find a subgraph in a given data graph that matches to a certain extent a pattern graph. In contrast to exact graph/subgraph matching, exact isomorphism is not sough in approximate matching, in stead, relaxations on structural differences is introduced, tolerating potential noise or error.

As mentioned earlier in Chapter 2 the problem of approximate subgraph matching is handled differently across the literature. One elegant approach that tackle this problem and has an application in biology is presented by Tian et al. in [6]. Author suggests a novel technique called SAGA to signal certain patterns in biological pathways. Considering the robustness of the model suggested by SAGA and its relevance to our problem, this model was adopted to build the approximate subgraph matching module of our algorithm. While implementation choices were kept open in Tian et al.'s work, various decisions had to be taken while implementing SAGA in terms of both algorithm and data structure. The technical details of our approach are presented later in Chapter 4. While in this section and for the sake of completeness, we briefly present the model suggested by SAGA, where details about the model can be found in the original paper [6].

3.2.1 Problem Modeling

In order to develop their approach, authors define a formal framework of the problem first. In the model a graph is a 3-tuple $G = (V, E, \lambda)$ where V is the set of all vertices (nodes), E is the set of edges $E \subseteq V \times V$, and λ is the labelling function $\lambda : V \to L$, where L is the set of all possible labels.

Given two graphs, $G_1 = (V_1, E_1, \lambda_1)$ and $G_2 = (V_2, E_2, \lambda_2)$, a match from a pattern G_1 to data graph G_2 is defind as a bijection $\phi : V_1' \to V_2'$ where

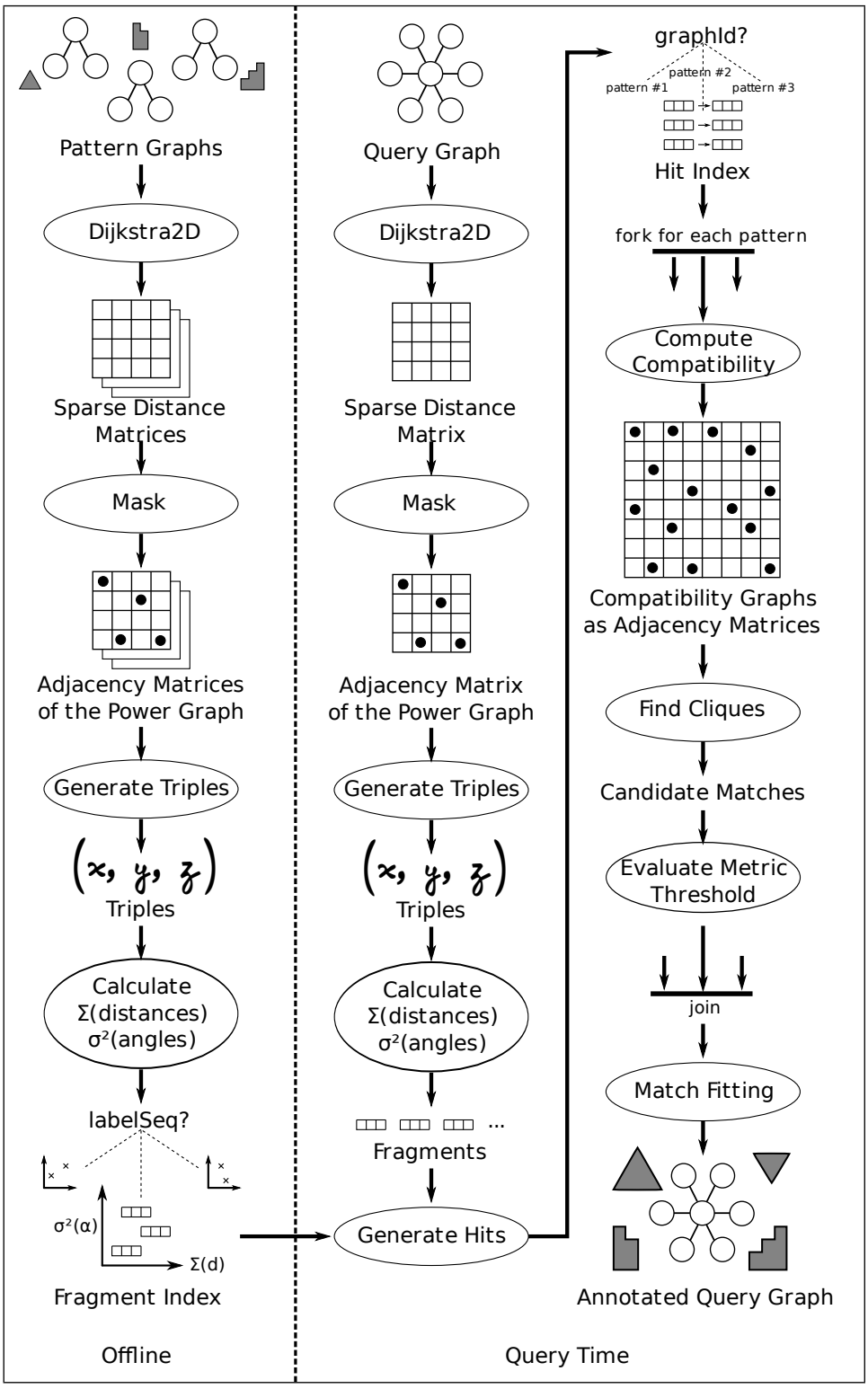


Figure 3.1: A diagram sketching the proposed algorithm.

$V_1' \subseteq V_1$ and $V_2' \subseteq V_2$. As a subgraph matching, not all the nodes in the data graph have to be mapped to respective matches, moreover, nodes in the pattern graph may or may not be mapped to nodes on the graph nodes, allowing for gap nodes as an approximate matching. A gap nodes g is a node in the pattern graph that doesn't match any node on the data graph according to a specific match, that is $g \in V_1 \setminus V_1'$. Originally, nodes with different labels are free to be matched.

An essential aspect of the model then, is to estimate a match's goodness, This is done by means of a metric function μ that evaluates the distance between the pattern and its matching subgraph.

$$\mu(G_1, G_2) = \omega_d \times \mu_d + \omega_\lambda \times \mu_\lambda + \omega_g \times \mu_g \tag{3.1}$$

The metric function is a sum of weighted metrics; the structural distance μ_d, the node mismatch distance μ_λ and the gap penalty μ_g, weighted by ω_d, ω_λ and ω_g, respectively.

$$\mu_d(G_1, G_2) = \sum_{v,w \in V_1',\ v<w} |d(v,w) - d(\phi(v), \phi(w))| \tag{3.2}$$

$$\mu_\lambda(G_1, G_2) = \sum_{v \in V_1'} labeld(\lambda_1(v), \lambda_2(\phi(v)) \tag{3.3}$$

$$\mu_g(G_1, G_2) = \sum_{v \in V_1 \setminus V_1'} peng(v) \tag{3.4}$$

The structural distance term measures the sum of the difference between the distance between each pair of mapped nodes in the patterns, and the distance between its corresponding pair in the data graph. Distances are defined by the function $d : V \times V \to \mathbb{N}_0 \cup \{\infty\}$ this is the length of the shortest path between two nodes in a graph when there is one, ∞ otherwise. Mismatched nodes are penalized based on a label distance function $labeld : L \times L \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ where $labeld(l, l) = 0 \ \forall \ l \in L$. The metric function also suggests that while unmatched data nodes are completely tolerated, gap nodes are penalized based on properties of the underlying node according to the function $peng : V \to \mathbb{R}_{\geq 0}$.

For the sake of simplicity, strict label comparison is applied; that is labels similarity is a dirac delta, and label distance function is given as follows.

$$labeld(l_1, l_2) = labelδ(l_1, l_2) = \begin{cases} 0 & \text{if } l_1 = l_2 \\ \infty & \text{otherwise} \end{cases} \tag{3.5}$$

3.2.2 Divide and Conquer Algorithm

The proposed solution is based on dividing pattern and data graphs into small structural units called *fragments*, fragments are then indexed based on

their properties of interest, allowing for efficient matching. Small matches, called *hits*, are then used to build up consistent, larger and more significant candidate matches, candidates are then evaluated according to the metric function, where a candidate relevance is inversely proportional to its distance from the data graph.

Fragments are all of fixed size `N`, representing sets of `N` nodes belonging to the same graph. To reduce the number of generated fragments, without affecting the soundness of the algorithms, a fragment should satisfy certain properties of forming a rational structural unit, otherwise it will be useless, while very time and space consuming, to generate all N-combinations of graph nodes. Thus, nodes of a single fragment are assumed to be topologically close, in other words, we expect every valid fragment $F = \{n_1, n_2, \dots n_N\}$ to satisfy

$$\forall\, n, m \in F,\, d(n, m) \leq d \tag{3.6}$$

where `d` is the maximum distance allowed between two nodes in one fragment. That means that each fragment forms a complete subgraph of order `N` of the `d`-th power of the original graph[1]. `N` is assumed to be a user-defined parameter.

Besides graph ID, nodes IDs, and nodes labels, relative distances between fragment's nodes are stored in the fragment itself, moreover, the sum of those distances are also associated, to enable faster search through indexing as we will see next. Table 3.2.2 shows fragment structure as suggested by SAGA.

Table 3.1: Fragment structure

Field	Size	Type
`graphId`	1	`int`
`nodeSeq`	`N`	`int`
`labelSeq`	`N`	`Label`
`distSeq`	$\binom{\texttt{N}}{2}$	`unsigned int`
`sumDist`	1	`unsigned int`

3.2.3 Indexing Patterns

To allow fast and efficient lookup for matches, fragments are indexes according to their properties of interest. Interesting properties of one node, thus, of its fragment, are those nominal and topological. Nominal properties are the label and the graph ID, while topological ones are respective distances

[1]The n-th power of a graph $G(V, E)$ is a graph $G^n(V, E')$ where $(v, u) \in E' \iff d(v, u) \leq n$ where $d(u, v)$ is the distance between two nodes v and u in G.

between different nodes. As matching –in this phase– is only done based on nominal and topological attributes of a graph, only those properties matter while indexing, where identifying properties of a node, such as its ID, are irrelevant to the fragment matching process.

SAGA suggests a concise, yet efficient, index structure, in which fragments are indexed according to their `labelSeq`; that is the sequence of the label of their nodes, and `sumDist`; that is the sum of intra-fragment distances. Only exact matches are considered for `labelSeq`, while range search is applied when matching `sumDist`. This simple two level index, called *FragmentIndex* can easily be implemented using existing techniques such as binary self-balancing trees, B-trees or B+-trees.

SAGA also suggests another index structure, *DistanceIndex*, in which relative distances between all nodes within a graph are stored, this index is particularly useful when evaluating the metric value for each candidate match.

For each pattern graph, valid fragments are generated and indexed, building the FragmentIndex, distances between nodes are also calculated generating the DistanceIndex, this work can be done offline, where built index can be kept for later processing, SAGA uses a database management system for this purpose.

3.2.4 Matching a Query

When the user provides a query, the same process that is previously run on patter graphs to generate fragments will be run on the query graph. However, fragments set for the data graph is extended to include all possible mutation for a fragment where more than one node share the same label. This is to solve for the problem of matching all relevant fragments without the need to exhaust FragmentIndex with redundant entries. Next, matches are queried for each fragment in the data fragments set. Matching is done through a two-level filtering, at the first level FragmentIndex is probed to retrieve candidate matches, candidate matches satisfy the following.

$$labelSeq_q = labelSeq_p \tag{3.7}$$

$$|sumDist_q - sumDist_p| \leq \binom{N}{2} \times MaxPairDist \tag{3.8}$$

Where $labelSeq_q$ and $labelSeq_p$ are query's and patern's label sequences respectively, and $sumDist_q$ and $sumDist_p$ are the sum of pair-wise distance of query's and patern's nodes, in respective order. `MaxPairDist` defines the maximum difference between the distance of pair nodes in a data fragment, and the distance between their respective matches in the pattern graph, this is a user defined value that the algorithm takes as a parameter.

As mentioned before, the structure of the two level FragmentIndex allow for such conditions to be efficiently checked. Those conditions are necessary,

though not sufficient, thus, results of the first level filtering are passed to the second level, where candidates are checked one by one to satisfy the maximum pair distance condition, eliminating false positives.

Pairs of fragments that survived the two level filtering are called *hits*. A hit can be defined as a bijection $h : F_p \rightarrow F_q$ where F_p and F_q are a pattern and a query fragment, respectively. Hits are then grouped according to their pattern graph ID. Next, they are assembled to build bigger and more significant matches. A candidate match is a maximal set of pair-wise non-conflicting hits, it is maximal in the sense that it contains no subset that consists of pair-wise non-conflicting hits. To this end, a hit compatible graph is built for each pattern graph, this is done by connecting all pairs of compatible (non-conflicting) hits in each group. Compatibility between two hits is the property of having no shared node in the pattern graph that maps to two different nodes according to the two hits, it also implies that no two different nodes in the pattern graph map to the same node in the query graph. Figures 3.2 and 3.3 show different examples of compatible and incompatible hits.

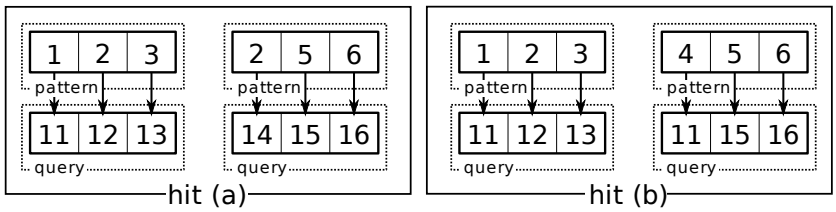


Figure 3.2: Incompatible Hits. (a) Shared node in the pattern graph that maps to different nodes in the query graph. (b) Two different nodes in the pattern graph map to the same node in the query graph.

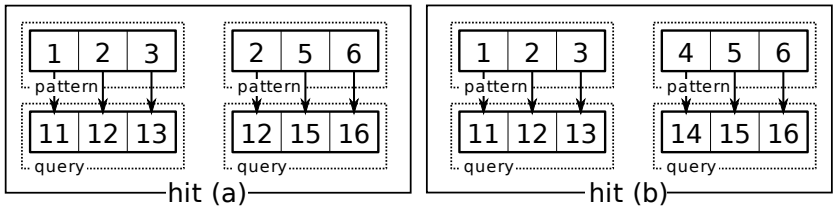


Figure 3.3: Compatible Hits. (a) One shared node in the pattern graph that maps to the same node in the query graph according to both hits. (b) No shared nodes in the pattern graph, neither in the query graph.

This can be expressed as follows. Given two hits h_1 and h_2, $h1$ and $h2$

are compatible if and only if the two following conditions hold.

$$h_1(p) = q1 \wedge h_2(p) = q2 \implies q_1 = q_2 \tag{3.9}$$

$$h_1(p_1) = q \wedge h_2(p_2) = q \implies p_1 = p_2 \tag{3.10}$$

Now given the hit-compatibility graph for each pattern, finding candidate matches boils down to the clique detection problem, as candidate matches are the maximal fully-connected subgraphs (cliques) of the compatibility graph. A clique finding algorithm, such as Bron & Kerbosch algorithm[8] is then applied to generate candidates. which are then checked to verify that the ratio of number of gap nodes to the order of the pattern graph doesn't exceed the allowed limit, which is a user defined parameter `PGaps`. This parameter is used for the early determination of weak matches, saving the time needed to evaluate their metric values.

Candidates that meet this condition are then passed to the metric function defined in equations 3.1, 3.2, 3.3 and 3.4, to evaluate their goodness. In this step, DistanceIndex is probed.

3.3 Geometry-Aware Pattern Finder

Our implementation of SAGA gave accurate results for small-size graphs, even when a margin of error or noise was introduced to the data graph. However, execution time for graphs of order larger than few tens became prohibitive. That was a direct consequence of the exponential nature of the clique finding problem. As the work of Moon & Moser[9] shows, the number of maximal cliques in a graph of order n is bound to $3^{\frac{n}{3}}$, this leads to an exponential complexity in the general case even when output-sensitive algorithms[2] are applied. The Bron & Kerbosh algorithm[8] used in our implementation is worst-case-optimal, and reported to be more efficient in the general case than other alternatives[10].

One work around to alleviate this complexity is to reduce the number of generated hits, this can be achieved by applying stricter constraints when fragments are matched. Domain knowledge provides more information about the matched patterns and the data graphs rather than solely the topological layout of particles along with their labels. In practice, this information may be of great benefit to produce more rigid, yet valid, similarity measures, reducing the number of hits that are passed to the maximal clique detector algorithm.

[2] Algorithms that guarantee a polynomial execution time with respect to the number of generated cliques do exist.

3.3.1 Introducing Geometry

In the domain of nano-technology, one property that makes sense in this context, and does count when matching patterns against nano-systems is the relative displacement of particles in the Euclidean space. In order for such a geometric measure to be valid, it has to be both scale and rotational invariant, as different models may represent distances in different units and scales, unit heterogeneity may even occur within the same model, furthermore, nano-systems are often observed from different angles, thus, represented in different respective rotations.

Once such a geometric measure is defined, it can be integrated in the fragment matching criteria, radically reducing the number of hits, by eliminating those that pass the nominal and topological check, but are still too irrelevant to form any significant match later when candidate matches are assembled and evaluated.

Euclidean distances between pair-wise particle in one fragment may seem to be an intuitive candidate to estimate fragments relevance more precisely. Given particle coordinates, Euclidean distances can easily be computed and stored. An upper limit can then be enforced on the difference between distances of two particle in a fragment and their respective matches in a hit in a similar manner to the way it is done for topological distances. However, one essential drawback of this approach is its tight dependency on the scale, a different choice of metric units leads to a complete failure to recognize the simplest patterns.

Angles formed between segments connecting each node with another two in the same fragment can easily be computed as well given particles coordinates, and they are indeed scale and rotational invariant, that makes them a perfect candidate for the sought property. However, indexing fragments according to all $N \times (N-1)$ angles would exhaust the index, and slow down the retrieval process. A similar approach than the one used for distances can be thought about, that is to use a statistical descriptor to index fragments, then the index is used as first level filter, eliminating most of the negatives, while individual values are checked in the second level filtering to eliminate false positive. Clearly, a correlation between an upper bound on differences between individual values and a bound on the difference between their statistical descriptors should exist.

Where the sum (a scaled mean in the case of constant finite sample space cardinality) was used as descriptor for distances, such measure becomes obsolete when it comes to angles in a planar geometry, as it is constant for an n-gon (n-angle polygon), subsequently, it is of minor interest even when non-planar geometry is considered, as particles of a fragment can often be coplanar (always for a 3-particle fragment).

3.3.2 Internal Angles and their Variance as Guard

In our approach we suggest using internal angles formed by a polygon fragment as a geometric measure, and the variance of their values as a statistical descriptor that enables the two level filtering. When speaking about polygons we limit ourselves to planar geometry, such restriction is necessary to develop our approach. To insure planar fragments, N in this phase was restricted to the value of 3.

The variance suggests itself as the second central moment when the first moment fails to provide a meaningful quantitative measure, as it is constant. Next, we prove that an upper bound on the difference between the individual internal angles of two matching nodes in two fragments yields an upper bound on the difference between the two variances of the internal angles values of the two fragments.

Theorem 1. *Given two random variables* $X_1 : \Omega \to \mathbb{R}_{\geq 0}$, $X_2 : \Omega \to \mathbb{R}_{\geq 0}$ *that satisfy the property*

$$E(X_1) = E(X_2) = \mu$$

where $E(X)$ *is the expected value of the random variable* X*, then the following statement holds,*

$$(\forall \omega \in \Omega,\ |X_1(\omega) - X_2(\omega)| \leq \varepsilon) \implies |\sigma^2(X_1) - \sigma^2(X_2)| \leq 2 \times \mu \times \varepsilon.$$

Proof. We have for all $\omega \in \Omega$:

$$\begin{aligned} |X_1(\omega) - X_2(\omega)| &\leq \varepsilon \\ |X_1(\omega) - X_2(\omega)| \times (X_1(\omega) + X_2(\omega)) &\leq \varepsilon \times (X_1(\omega) + X_2(\omega)) \\ |X_1^2(\omega) - X_2^2(\omega)| &\leq \varepsilon \times (X_1(\omega) + X_2(\omega)) \end{aligned}$$

This is true because $X_1(\omega) + X_2(\omega) \geq 0$ as both of the variable are positive by assumption, applying the inequality property of expected value we get:

$$\begin{aligned} E(|X_1^2 - X_2^2|) &\leq E(\varepsilon \times (X_1 + X_2)) \\ E(|X_1^2 - X_2^2|) &\leq \varepsilon \times (E(X_1) + E(X_2)) \\ E(|X_1^2 - X_2^2|) &\leq 2 \times \mu \times \varepsilon \end{aligned}$$

We have

$$|E(A)| \leq E(|A|)$$

then we can wirte:

$$\left|E(X_1^2 - X_2^2)\right| \leq 2 \times \mu \times \varepsilon \tag{3.11}$$

We also have:

$$\begin{aligned} E(X_1^2 - X_2^2) &= E(X_1^2) - E(X_2^2) \\ &= (E(X_1^2) - \mu^2) - (E(X_2^2) - \mu^2) \\ &= (E(X_1^2) - E^2(X_1)) - (E(X_2^2) - E^2(X_2)) \\ &= \sigma^2(X_1) - \sigma^2(X_2) \end{aligned} \tag{3.12}$$

Now given 3.11 and 3.12, we deduce:

$$|\sigma^2(X_1) - \sigma^2(X_2)| \leq 2 \times \mu \times \varepsilon$$

□

In our application, $X : \{0, 1, 2\} \rightarrow [0, \pi[$, a discrete random variable, we also have $\mu = E(X) = \left(\sum_{i=0}^{2} X(i)\right) / 3 = \frac{\pi}{3}$, that implies according to theorem 1, that enforcing a limit ε to the difference between internal angles of two matching nodes leads to an upper limit of the variance difference of $\frac{2}{3}\pi \times \varepsilon$.

To incorporate geometric measure into the fragment matching process, angles are computed and stored into each fragment while they are generated, moreover, another attribute is added to the fragment which is the shifted variance, to enable angles indexing. Table 3.3.2 shows the new structure of the fragment at this stage.

Table 3.2: Fragment structure

Field	Size	Type
`graphId`	1	`int`
`nodeSeq`	3	`int`
`labelSeq`	3	`Label`
`distSeq`	3	`unsigned int`
`angleSeq`	3	`unsigned int`
`sumDist`	1	`unsigned int`
`varAngle`	1	`unsigned int`

The index is then altered to account for the new criterion, while the first level of the indexed remains intact, to select fragments that strictly match according to their label sequence, the second level is adopted to perform an orthogonal range search instead or one dimensional range search. Details about the implementation are presented in Chapter 4.

The first level filtering now retrieves fragments that satisfy the loose index conditions that concern their label sequence, distance sum and angle

variance. Equation 3.13 shows the additional condition that a pattern and a data fragment must meet in addition to those shown in equation 3.7, 3.8 before they pass to the second level filtering, where more rigid check is performed against fragments to insure that they meet the matching criteria.

$$|varAngle_q - varAngle_p| \leq \frac{2}{3}\pi \times \varepsilon \tag{3.13}$$

where $varAngle_q$ is the angle variance of the query fragment, and $varAngle_p$ is that of the pattern fragment.

3.3.3 Geometrical Tolerance

In this phase, the geometric measure was introduced to enhance matching process in term of space and time cost. However, this measure was not integrated in the metric function. Observations showed that this introduction led to an algorithm faster than the original by orders of magnitude, where results remained to a large extent intact. In a matter of fact, the accuracy and support of the results are now a function of the newly introduced parameter `MaxPairDiff`.

The bigger the value is, the looser is the geometric check, the slower is the algorithm. For example, for all values higher that π hits passed to the clique finder are the same as if no further checks have been introduced, making the algorithm a bit slower than the original, because of the unnecessary overhead of checks still performed, though with a positive results constantly.

On the other hand, when the variable goes too small, the geometric test becomes too strict to allow for the slightest differences, leading to inconvenient results. A suitable value of this parameter is usually relative to the domain at hand, and how much similar graphs are expected to differ geometrically.

3.4 Graph Decomposition

The ultimate goal of our algorithm is to recognize predefined patterns expected to occur in a given data graph. That is to decompose the graph into subgraphs, each annotated with a pattern name, along with the map from the pattern's nodes to the subgraph's nodes, to enable the transmission of pattern's particles properties to the data graph nodes. This decomposition is not meant to be collectively exhaustive[3], though it is expected to be mutually exclusive[4].

[3] In a collectively exhaustive decomposition of set S, the union of all subsets equals to S.

[4] In a mutually exclusive decomposition, all pair-wise intersection of all subsets equals to the empty set $\emptyset$.

Until now, matches for a single pattern are searched within a graph, this operation can be repeated, or parallelized to generate matches for all the patterns. In an ideal world, those results could have been used to annotate the data graph, then the annotated graph is returned to the user in a friendly manner. However, it's not the realistic case!

In a matter of fact, results do often overlap, that is mainly the consequence of the fact that patterns themselves are similar to each others and may overlap, one pattern may be totally or partially included in another one. Moreover, results come with different scores (or inversely proportional metric values), although metrics fairly order result for one pattern according to its decreasing similarity, the order is not at all fair when applied to results coming from different patterns.

3.4.1 Score Function Normalization

The first step in this phase was to define means of normalization of the metric function amongst different patterns. Observations showed that the average acceptable metric[5] for a pattern is proportional to the size of the weighted size of the pattern graph, where the weighted size of a graph $G(V, E)$ is defined as follow;

$$||G|| = |V| - \sum_{v \in V} peng(v) \tag{3.14}$$

where $peng : V \rightarrow [0, 1]$ is a restricted version of the gap penalty function mentioned in equation 3.4.

Such observation is quite intuitive, as the larger the graph grows, the more likely structural differences and node gaps are to happen. The introduction of the weighted size instead of the graph order is necessary, as nodes with less or no gap penalty are less penalized, thus they should count less when normalizing. Normalizing is then done by simply dividing the metric function by the weighted graph size.

3.4.2 Thresholding

Next, a threshold that distinguishes the good from the bad is to be defined, this is done experimentally. Such threshold should be consistent along all patterns, widely variant thresholds indicate bad normalization of the metric function. The experimental nature of the guess of this threshold makes it closely related to the domain at hand, precisely the type of nano-systems we are dealing with.

When estimating the threshold value, a match soundness is judged based only on the nominal, topological, and geometrical similarity, though the result itself may not be agreeable. For instance, a small pattern that is

[5] An acceptable metric is the minimum metric value for which a match is considered to be correct.

included in larger ones will match its expected images in the data graph plus parts of the images of the bigger patterns in which it is included with similar metric values. This is admittedly expected from the algorithm so far, where the inclusion problem will be handled soon.

3.4.3 Competition on Graph's Nodes

To address overlapping and inclusion, we make the following assumption; For good matches (those with a metric value that doesn't exceed the threshold) a bigger match is prior to a smaller one, regardless their metric value. This assumption fits perfectly the case of total inclusion of small patterns by larger ones, which is a common phenomenon in most of the applications of nano-technology. Such assumption allows us to order the matches that survived the threshold test according to their priority, that is in lexicographical order, first according to their length, and second (if the lengths are equal) according to their decreasing metric values. Figure 3.4 shows an example of how matches for different pattern may overlap, and how this issue is handles.

The previous ranking then allows for an easy method to position pattern matches over the data graph, starting with the most prior match, node in the data graph are assigned to their respective images according to the match. Whenever a conflict occurs, the whole match is ignored, as conflicts indicate that a more prior match competes with the current one on some data graph nodes.

The outcome of this phase is the output of the algorithm, which is the data graph itself now annotated with detected patterns, and for each prediction, the map between the pattern's node and the respective matches in the data graph nodes.

3.5 Molecular Data Representation

Complementary to our algorithm, a general scheme is to be defined to represent molecular systems in a uniform manner. The scheme should be able to express topological, geometrical, and nominal attributes of a nano-system represented as graph (either a pattern or a data graph). Moreover, it has also to be as unified and general as the algorithm is, that is, all expected models and forcefield should be expressible using this representation.

This data structure is specially useful to represent patterns for expected model and forcefield. For a specific model, it stores for all patterns along with their particles represented as graph nodes, and the bonds between particles represented as graphs edges. Nodes are labeled, and they have a weight associated with each, denoting its significance. Significance will be used to estimate gap penalty when such a node is missing in a match. This node-level approach of weighting allows for a general gap penalty function.

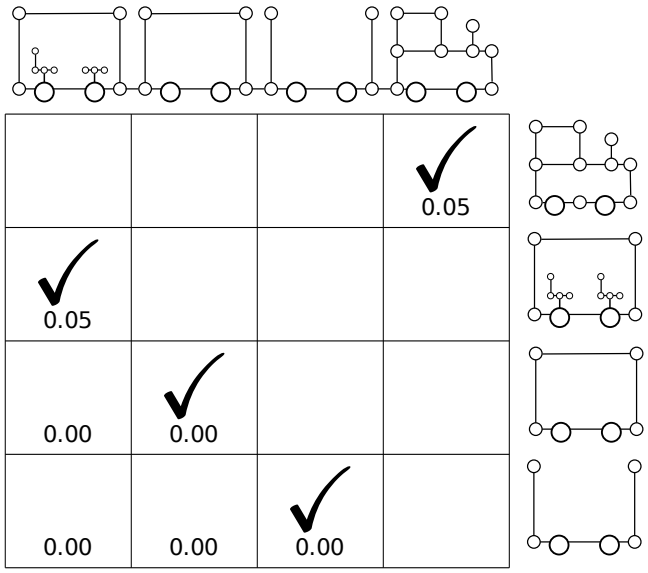


Figure 3.4: An example of a imaginary nano-train data graph that is matched against different patterns: the railway engine, the coal trailer, the carriage wagon, and the passengers wagon. It shows that though the coal trailer matches carriage wagon and passengers wagon perfectly, as it is an exact subgraph of both, the passenger wagon and carriage wagon patterns are prior to that of coal trailer, as they are bigger, thus they had the chance to choose their matches first, leaving only one choice to the coal trailer pattern. The same thing happens when the passengers wagon pattern takes the first chance to match the passengers wagon image, despite its higher metric than the carriage wagon pattern, caused by the error in the data graph (missing node), though the image –correctly– goes to the former.

A model may suggest zero-significance for particular nodes, though the nodes still have to be present. This is necessary to enable the guess of matches of those nodes if such matches exist, while keeping the score intact in case of their absence.

To avoid the burden of assigning individual values of significance to each and every particle in a molecular system, classes of particles can be defined. A particle inherits automatically the label and the significance of its class, yet, this can be overridden at the node-level.

Chapter 4

Implementation and Data Structures

In last chapter, a brief description of the algorithm was presented, the approach was developed incrementally. The basic idea inspired by the work of Tian et al.[6] was explained first, then an enhancement to this approach that dramatically decreased execution time was presented, and finally, a method to apply the results of such an approach to the problem of automatic conversion of molecular models was shown. In this chapter, we will dive deeper into implementation details and data structures, in an attempt to justify out technical choices.

In section 3.2 we briefly presented SAGA as suggested by Tian et al.[6]. However, in their publication only few implementation details were given. Next we will address –among other issues– our realization of their approach. Unless stated otherwise, all presented implementation choices in the rest of this chapter are ours.

4.1 Fragments Generation

A central issue to indexing is the generation of the index units, which are fragments in our case, a naive approach to generate fragments is to list all possible 3-combinations of nodes of the graph in hand, and for each triple, check whether or not it complies to the validity condition present in equation 3.6. However, such an approach becomes prohibitively expensive for modestly large graphs, thus, a more efficient approach had to be figured out.

We assume that both V and L are totally ordered sets, hence the strict total order $<_V$ and $<_L$ on each set, respectively.

Recall from section 3.2.1 that a valid fragment forms a complete (fully connected) subgraph of order $N = 3$ of the $\mathtt{d}$-th power of the original graph.

Thus, to generate fragments this power graph must be computed first, connecting all nodes that are at most `d` apart with a pseudo edge.

To this end, a variant of Dijkstra algorithm[7] is used, the algorithm builds a sparse distance matrix of the original graph, where only values less than a specified parameter are present, and all other bigger values are considered to be infinite. Dijkstra algorithm is used to calculate distances from one graph node at a time, iterating over all graph nodes to build up the distance matrix, all edges are assumed to have an equal weight of 1. The only difference from original Dijkstra –apart iterating all nodes, to compute the whole matrix– is that calculation of distances for one nodes is aborted once the maximum sought distance is reached. The incremental nature of the algorithm itself allows for such an assumption, as at the end of each iteration inside Dijkstra, all distances less than or equal to iteration number –assuming uniform edge weight of 1– are guaranteed to be decidedly evaluated.

A sparse distance matrix is then built using the above mentioned algorithm, with `d` as parameter. This matrix is in fact the adjacency matrix of the `d`-th power graph, considering all finite values as *true* and infinite values as *false*. Next, all `N`-combinations of the power graph are generated, where each combination corresponds to a fragment. A special data structure of the distance matrix is introduced, minimizing storage cost, and allowing for an efficient combination generation method. Distances is represented as a sparse triangular matrix; it is sparse because we are only concerned about a restricted set of distances (those that do not exceed `d`), it is triangular because our graph is undirected. The same node order $\leq$ is respected across rows and columns, each row, representing a node, stores the distances to nodes represented by rows that follow, avoiding redundancy, this can be expressed as follows.

$$n_i \in \mathcal{L}(n_j) \iff \begin{cases} n_j \neq n_i & \wedge \\ n_j \leq n_i & \wedge \\ (n_i, n_j) \in E_{G^d} & \end{cases}$$

Where $\mathcal{L}(n_j)$ is the line representing the node n_j, and E_{G^d} is the set of edges of G^d. Moreover, distances are presented as a linked-list, rather than dense array, where infinite distances (or those that are considered to be) are dropped. Figure 4.1 shows an example nano-system, along with its sparse triangular distance matrix.

Now given the above mentioned structure, 3-combinations can be generated efficiently by generating all ordered triples (3-tuples) of nodes that are pair-wise connected according to the power graph G^d. Matrix' rows are iterated in respective order, for each row, items of the linked-list are also traversed, at each item, both node corresponding to the current row, say n_1, and that corresponding to the current item of the linked list, say n_2, are

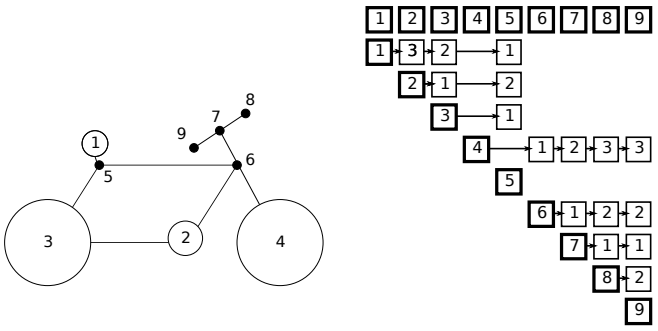


Figure 4.1: An imaginary nano-bike represented as a graph, along with the sparse triangular distance matrix with `d` set to 3.

connected according to the power graph, and form a good candidate for the first two elements of a 3-combination, as they are also ordered according to $\leq$. The third element is of bigger row number than both elements, that is than n_2, besides, it is connected to n_1 according to G^d, thus, if any third item exists, it is to be found in the rest of linked-list elements, subsequently, those elements are traversed, and for each element, the connection between the corresponding node, say n_3, and n_2 is checked, if such a connection exists, a new combination (n_1, n_2, n_3) is signaled.

For each signaled `N`-tuple, a new fragment is generated, relative distances are copied from the sparse distance matrix, and then summed for each fragment, furthermore, internal angles of the triangle are computed, based on vertices coordinates, then the shifted variance $\sigma^2 - \mu^2$ is computed and stored. The reason why the shifted variance is stored instead of the original one is that in the end we are only interested at the difference between variances, rather that the variances themselves, thus, and to simplify computations, the calculation of the variance shifted by a constant value μ^2 suffices, without affecting the theoretical model. Recall that the mean μ is constant in our case, thus, its square is as well.

To allow for efficient matching between fragments, nodes within a fragment should be sequenced in a meaningful order. A good order when considering strict label distance function $label\delta$ shown in equation 3.5 is the relation $<_L$, which defines –as mentioned earlier– a strict total ordering over the set of all possible labels L. However, nodes are only partially order according to $<_L$, since two or more nodes may share the same label in one fragment. Therefor, another relation had to be used to order those nominally identical nodes within in a sequenced fragment, in our implementation, we use $<_V$ for this purpose. This total ordering is useful for later stages of this phase, when fragments will have to be compared. The problem of matching all possible variations that one fragment may represent when

two or more nodes share the same label will be addressed later, when talking about fragments matching. According to this sequencing, each fragment now maps to an `N`-tuple of nodes in the same graph $F = (n1, n2, \ldots n_N)$. A fragment of size `N` will have `N` angle values, each angle is made by each previous vertex, current vertex, and next vertex, according to the ordering, it will also have $\binom{N}{2}$ different intra-fragment distances. Angle values and distances are sequenced as well based on nodes order in the fragment.

4.2 The Fragment Index

Next, the fragment index is built, as mentioned in the last chapter and suggested by Tian el al.[6], it is a two level index, where filtering is done in two phases. Fragments with different label sequence will be filtered out at the first level, thus, fragments are ordered according to their label sequence, a self-balancing binary tree is used for this purpose. At the leaves of the binary tree, fragments have to be searched according to their sum of internal pair-wise distances, and their angle values shifted variance, with certain margin of tolerance; that is, an orthogonal range search is to be performed. To this end, a 2-dimensional kd-tree data structure is used at the second level of indexing.

4.3 Hit Compatibility

It was also suggested by SAGA[6] that a query graph is provided, the sames fragment generation method is applied, with the slight difference that query fragments are mutated after the generation. Resulting fragment are then matched against the fragment index, before running a full fledged text to insure their similarity. Matching couples called hits are then grouped according to the pattern graph, compatibility between hits are then evaluating, generating a hit compatibility graph represented as an adjacency matrix that is passed to the clique finder algorithm to produce candidate matches. In our implementation, Bron & Kerbosh algorithm[8] is applied for maximal clique finding.

An important issue to consider in this context is the generation of the hit-compatibility graphs in an efficient manner. Hits are checked one against another to verify they meet conditions shown in equation 3.9 and 3.10. To this end, for each two different hits, intersection between the two pattern fragments is computed, this is done efficiently knowing that both fragments respect the same order (a condition that is not guaranteed for query fragments, because of mutations), for each element in the intersection, it is verified whether or not it maps to the same node according to both hits, the check is done during the intersecting process, so that if the mapping condition doesn't hold, the couple is considered to be incompatible and the

operation is aborted. If all elements of the intersection satisfy the mapping condition, then implication 3.9 holds. Now it suffice to prove that the intersection of the images of two pattern fragments equals to the intersection of the two query fragment to prove implication 3.10. However 3.9 implies that the former is contained in the later, then we only have to prove that the cardinalities of two sets are equal, that is the cardinalities of two intersections, as both functions are injective[1]. This is done by computing the cardinality of the intersection of query fragments, again applying early determination techniques to eliminate negatives as soon as possible.

The above mentioned technique utilizes the total ordering of pattern nodes, accelerating their intersection, in the same time, it avoids the intersection of query fragments, which can be expensive knowing that query fragments do not respect the same order. More over, the hit-compatibility check applies early detection of negatives as soon as possible to avoid any unnecessary calculations.

4.4 Evaluating Metrics

SAGA suggests a distance index to be used when evaluating the metric function for each candidate match resulting from the clique finding algorithm, this index stores for distances between all pair vertices in a graph. In our implementation, this index is no more than the sparse triangular distance matrix already computed. However, as mentioned earlier, and for the sake of efficiency, computed distances are restricted to a certain upper bound, so far this bound was set to `d`, which is the maximal intra-node distance in a fragment.

To enable the reuse of the matrix to compute metrics, all distances below certain value should also be present in the matrix, this value is in fact the order of the largest pattern graph, where all larger values can be considered infinite, bearing in mind that the difference between two infinite distances is infinite itself. Therefore, the value passed to the distance matrix will be the maximum between `d` and all pattern graphs orders. That means that distance matrix for pattern graphs is not spare anymore, and that the assumption made about mapping this distance matrix to an adjacency matrix of the `d`-th power graph is invalidated.

The density of the pattern graphs distance matrix is an inevitable compromise, however, patterns are usually of moderate sizes compared to query graphs, which keeps the complexity of such data structure within acceptable boundaries. To solve for the invalidity of the adjacency matrix, masking is applied to the distance matrix after it is populated, so that only values smaller than or equal to the mask are considered finite. Once fragment generation is done, the matrix is unmasked again.

[1] $f : A \to B$ is injective $\implies \forall S \in A, |f(S)| = |S|$

4.5 Matches Prioritization

Now the unmasked distance matrix is used to compute metrics for candidate matches, using the gap node percentage as a guard to eliminate weak candidates, as mentioned in last chapter. A threshold cut is then used to filter out bad candidates, while surviving matches are descendingly ordered according to the subtraction of their metric (shown in equation 3.1) from their weighted size (shown in equation 3.14). Next, the intuitive algorithm explained in section 3.4 is used to diffuse matched patterns on the query graph nodes.

4.6 Molecular System File Format

As mentioned in previous chapter, the algorithm comes with a supporting molecular data structure, that allows for a general representation of molecular systems, regardless the forcefield used to define the it. As was shown, the scheme is meant to be general In compliance with the algorithm universality.

However, the data have to be written to a permanent storage at the end of the day. A storage mechanism had to be defined to this end, it has to accommodate the schematic properties of the data structure. A desirable quality is not to be only machine interpretable, but also human readable.

Plain-text flat files suggest themselves in this context. However, hierarchy and structure are hard to express in plain text, even when they are present, it is at readability's expense. Alternatively, a relational database management system can be used to store for our graph data, this allows for sophisticated data structure arrangements, nevertheless, the use of database management systems gives raise to the problem of portability amongst different platforms, and makes data far from being readable by the mere eye.

One choice that is simple, portable, human readable, yet allows advanced schema and hierarchies is the use of XML standards. XML documents inherit simplicity and portability from plain text, and formalism and coherence from relational databases. In our implementation, an XML schema was defined to store for nano-systems (patterns and data graphs) confirming to the molecular data structure proposed in section 3.5. We used XMLIO (`http://xmlio.sourceforge.net/`) as a high level library to provide the necessary functionality to read and write XML documents.

Appendix A shows an example snippet extracted from a CHARMM19 patterns file.

Chapter 5

Experiment and Results

In this chapter we will address the validity of our approach, presenting our experiment framework first, and then detailing obtained results.

5.1 Polypeptides as Nano-Systems

A good application of nano-technology, and thus, of our algorithm, would be proteins' simulation. Because of their importance to biology, proteins are intensively studied molecular systems. Models exist to simulate their behavior and estimate their properties, those models are referred to as forcefields, as shown earlier in Chapter 1.

Proteins (also referred to as polypeptides) are organic compounds formed as linear chains of amino acids joined together by peptide bonds (special case of covalent bonds). The fact that proteins are built up of the repetition of predefined elements of a finite set of small patterns (that is amino acids) allows for an easier way to assign a models parameter to each particle (atom, or set of atoms, depending on the model). Rather than calculating those values for each particle by itself, parameters can be calculated for all patterns' particles, then, patterns are searched within a protein, and each particle in the chain is mapped to its respective match in a pattern, from with it will inherit models properties.

Our algorithm perfectly fits in this context to handle the matching issue. Amino acids, presented using one forcefield are passed to the algorithm as patterns, then a protein, presented possibly with another forcefield is passed as a query graph. The output will be the same protein with its particle marked with their respective matches. The tolerance towards gaps and structural differences of our algorithm makes it possible to use different representation across patterns and query graphs.

5.2 Testing Data Set

In order to evaluate the soundness of our algorithm, results had to be compared against trusted ground truth. To this end, a testing data set associated with truthful result had to be found, then our algorithm should be run on these data, comparing obtained results with those known to be true. We used the Protein Data Bank[23] for this purpose, to extract already annotated data, to which our results were compared. Then a testing framework was built, that reads the `.pdb` files generating query graphs, and running the algorithm on each, then results were compared with the annotation existing already in the `.pdb` file.

On the other hand, two pattern sets were prepared, the first, representing all 20 amino acids using CHARMM19 forcefield, the other representing the same amino acids using a CHARMM27-compliant representation. The main different difference between both is the fact that the former uses united atom, possibly eliminating hydrogen atoms, while the later uses all atom, explicitly showing protons, as mentioned earlier in Chapter 1. Again, amino acids were extracted from `.pdb` files, to generate CHARMM27 representations. tools such as AutoPSF were used. Then our patterns were transformed and stored in the general molecular system file format (section 3.5).

5.3 Experiment Framework

A testing framework was set up to undertake different experiment and to collect and compare the results. Our algorithm addresses general nano-system, however, while execution time grows moderately with the increasing number of graphs, it explodes exponentially when the number of vertices exceeds one hundred! Polypeptides are known to be huge molecular systems, with diversely big sizes. Thus, simply running the algorithm on one protein will take long time before getting any result, if at all possible.

Proteins on the other hand are easily divisible into groups of residues. We use this property to construct a naive approach to divide data graphs into smaller chunks, allowing acceptable execution times, a more elaborated approach to solve for exponential execution time will be discussed we addressing future works.

The applied approach is only used to enable estimating our methods accuracy against big test data. While when execution times are reported, experiments are performed without the use of any helping technique.

5.4 Results

In the first experiment, we apply the algorithm against Hepatitis-C virus NS3 protein, with PDB ID of 1A1V, extracted from the protein data bank,

after CHARMM19 patterns are loaded, with the geometric tolerance set to $\frac{\pi}{16}$. Next, We keep the same parameters, while changing the patterns to CHARMM27 pattern set, then we query the same protein. We repeat the same two experiments, but for $\frac{\pi}{32}$ as geometric tolerance this time. Tables 5.1 and 5.2 show the results of those experiments.

Patterns	Resisues					Particles		
	Good	Bad	Mix	Void	Total	Hit	Miss	Total
CHARMM19	384	39	0	9	432	3446	446	3892
CHARMM27	402	21	1	8	432	3637	255	3892

Table 5.1: Results of querying protein 1A1V extracted from PDB against both pattern sets, with geometrical tolerance set to $\frac{\pi}{16}$.

Patterns	Resisues					Particles		
	Good	Bad	Mix	Void	Total	Hit	Miss	Total
CHARMM19	397	35	0	0	432	3535	357	3892
CHARMM27	392	28	1	11	432	3540	352	3892

Table 5.2: Results of querying protein 1A1V extracted from PDB against both pattern sets, with geometrical tolerance set to $\frac{\pi}{32}$.

Next, we rerun the experiment for the same protein, this time presented as CHARMM27. This is done with geometric tolerance set to $\frac{\pi}{32}$, once with CHARMM19 as pattern set, and another time with CHARMM27 as pattern set. Obtained results are shown in table 5.3.

Patterns	Resisues					Particles		
	Good	Bad	Mix	Void	Total	Hit	Miss	Total
CHARMM19	312	107	0	13	432	2969	3519	6488
CHARMM27	306	117	0	9	432	3919	2569	6488

Table 5.3: Results of querying protein 1A1V represented as CHARMM27 against both pattern sets, with geometrical tolerance set to $\frac{\pi}{32}$.

The algorithm is up to 95% accurate in term of correctly guessed residues, while accuracy reaches 93.4% in term of correctly matched nodes. The two figures differ because the algorithm may guess that two residues match, but fails to guess all particle correspondences in every residue. The large number of particle misses when the CHARMM27 represented protein is matched against CHARMM19 pattern set is completely justified. That's because CHARMM27 is an all-atom representation, while CHARMM19 is a united-

atom one, that means that patterns are short to represent some particles that exist in the query, though, this shortfall doesn't harm the scoring function, as missing particles (mainly hydrogen protons) are completely tolerated in the model. This is materialized as zero significance in the pattern data set for those particles.

To examine the error tolerance and execution time, we consider the pentapeptide methionine-enkephalin (figure 5.1) as our query graph. Represented as CHARMM19, the polypeptide is queried against CHARMM19 and CHARMM27 patterns sets, and all five protein were correctly guessed in both experiments. Next, the molecule was deformed, removing three atoms, the zeta-carbon form the phenylalanine residue, the carbonyl oxygen form the methionine, and the nitrogen atom form tyrosine, figure 5.2 compares the deformed molecules with the original one. Despite the deformation, and thanks to the algorithm's gap node tolerance, all five residues were correctly detected again. Tables 5.4 and 5.5 list obtained results.

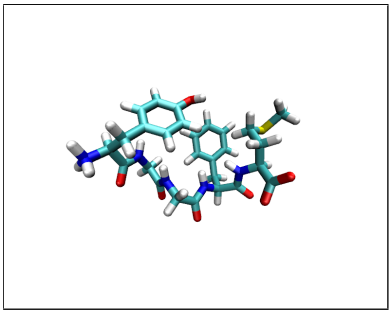
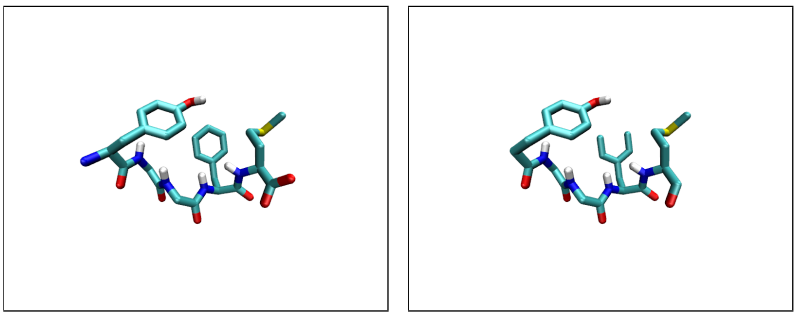


Figure 5.1: Met-Enkephalin



(a) (b)

Figure 5.2: On left, the original met-enkephalin in CHARMM19 format, while on right the deformed molecule used in the experiment.

Patterns	Residues			Particles			Time
	Hit	Miss	Total	Hit	Miss	Total	(seconds)
CHARMM19	5	0	5	44	1	45	13
CHARMM27	5	0	5	42	3	45	28

Table 5.4: Results of querying methionine-enkephalin represented as CHARMM19 against both pattern sets, with geometrical tolerance set to $\frac{\pi}{32}$.

Patterns	Residues			Particles			Time
	Hit	Miss	Total	Hit	Miss	Total	(seconds)
CHARMM19	5	0	5	42	0	42	4
CHARMM27	5	0	5	40	2	42	11

Table 5.5: Results of querying deformed methionine-enkephalin (after the removal of three particles) represented as CHARMM19 against both pattern sets, with geometrical tolerance set to $\frac{\pi}{32}$.

It is remarkable that the execution time more than doubles when querying using the CHARMM27 patterns set rather than CHARMM19 one, that's because CHARMM27 patterns are generally larger than those of CHARMM19, as they explicitly represent all atoms. This leads to more hits, thus more time finding the candidate matches (that is maximal cliques in the hit compatible graphs).

To demonstrate the value the geometric check adds, we run the same query, witch is methionine-enkephalin in CHARMM19 representation, against CHARMM19 pattern set, once with the geometric check enabled (geometric tolerance set to $\frac{\pi}{32}$), and again with the geometric check disabled (geometric tolerance $\geq \pi$). As illustrated in table 5.4, enabling the geometric filtering permit execution time of 13 seconds, while the execution time exceeds four hours on the same machine when the filtering is disabled. This radical drop in execution time is a direct result of the significant decrease of number of hits.

Chapter 6

Conclusions and Future Works

In this chapter we conclude to summarize what have been shown before, pointing out the contribution of this work, and the novelty it brings. Then we discuss in a glance future works addressing possible improvement to the algorithm.

6.1 Conclusions

In this work we presented a method to tackle the problem of automatic molecular models conversion, applying pattern recognition and graph matching techniques. This work builds on earlier achievements in similar scientific fields. Nevertheless, it differs from related works in two major aspects. First, it suggests an approach to incorporate scale and rotational invariant geometric measures in an effort to enhance and speed up the matching of two molecular systems. We construct such an approach on a well-established theoretical model. This improvement is shown to accelerate the process of matching orders of magnitude, without affecting the soundness of the obtained results.

Second, the work demonstrates a simple, yet efficient method to combine results of matching of several patterns in a sane manner, accounting for partial mapping, overlapping, and inclusion.

The proposed method shows impressive results in term of accuracy and error tolerance. However, when large molecular systems are queried execution time becomes an issue that's still to be considered. In the following section we will address potential solutions to this problem that are not yet materialized in our implementation by the time of writing this document.

6.2 Future Works

The algorithm was designed to be parallelizable, that is, to allow the execution on more than one processor simultaneously, and then to join results coming from different threads without affecting the final outcome. That means that the execution time can theoretically be divided by the number of available engines, with minimal communication overhead. In a matter of fact that is only true to a certain extent, as the number of utilized engines will be always limited to the number of patterns (as well as the number of available engines obviously). Moreover, some parts of the algorithm still have to be executed sequentially, the execution time of sequential code, however, is almost negligible compared to that of parallelizable one. Though concurrency is totally plausible, it's not yet implemented by the time being.

To alleviate the explosive growth of execution time for bigger molecular system, we note that it grows exponentially with the number of nodes in the query graphs, while it only grows linearly with respect to the number of graphs themselves. This observation suggests the decomposition of one big graph into smaller ones, then to query the small parts each at a time. However, care must be taken when partitioning the graph, as potential matches at partitions borders will not be recognized.

One possibility then is to decompose the query graph into smaller overlapping graphs, in such a way that each pattern can fit in at least one subgraph. This way, we avoid the problem of unrecognized matches at partitioning borders.

To insure this previous property, we first decompose the graph into its connected components, then for each connected component, and if the component still need to be divided, we assume an acyclic topology, that is a tree graph. We then traverse the tree depth first, and at each cut point, that is the node where one subgraph that will be reported to the algorithm ends, at least the closest n neighbors of the cut node (including the node itself) should be incorporated when constructing the next subgraph, where n is the order of the largest pattern graph.

To satisfy the assumption of tree topology, simple cycles in a connected component can be folded and treated as single nodes when partitioning, that is, partitioning cannot take place inside a cycle. More complicated cyclic topologies are harder to cope with. However, general application of nano-systems rarely contains such topologies.

One more issue to consider, is the size of one partition, if this size is too small, then the partitioning is pointless, as one pattern may cover the whole partition. Fortunately, in our application patterns comes in small sizes, allowing for reasonable partitioning.

Bibliography

[1] Gibson, D.G., Glass, J.I., Lartigue, C., Noskov, V.N., Chuang, R.-Y., Algire, M.A., Benders, G.A., Montague, M.G., Ma, L., Moodie, M.M., et al. (2010). "Creation of a bacterial cell controlled by a chemically synthesized genome". *Science*. 329: 52—56.

[2] Rappe A. K.,Casewit C. J.,Colwell K. S.,Goddard W. A.,Skiff W. M. (1992). "UFF, a full periodic table force field for molecular mechanics and molecular dynamics simulations". *Journal of the American Chemical Society* Vol. 114, No. 25, pp. 10024—10035.

[3] Allen B. Richon (1994). "An Introduction to Molecular Modeling". *Mathematech*. 1, 83.

[4] Brooks B.R., Bruccoleri R.E., Olafson B.D., States D.J., Swaminathan S., Karplus M. (1983). "CHARMM: A program for macromolecular energy, minimization, and dynamics calculations". *J. Comp. Chem.* 4: 187—217.

[5] Korkut A., Hendrickson W.A. (2009). "A force field for virtual atom molecular mechanics of proteins". *Proc. Natl. Acad. Sci. USA* 106: 15667—15672.

[6] Tian Y, et al. (2007). "SAGA: a subgraph matching tool for biological graphs". *Bioinformatics* 23: 232—239.

[7] Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". *Numerische Mathematik* 1: 269—271.

[8] Bron C., Kerbosch J. (1973). "Algorithm 457: finding all cliques of an undirected graph". *Commun. ACM* (ACM) 16 (9): 575—577.

[9] Moon J. W., Moser L. (1965). "On cliques in graphs", *Israel Journal of Mathematics 3: 23—28.*

[10] Cazals F., Karande C. (2008). "A note on the problem of reporting maximal cliques". *Theoretical Computer Science* 407 (1): 564—568.

[11] Bomze I.M., Budinich M., Paradalos P.M., Pelillo M. (1999). "The Maximum Clique Problem". In: Du D.Z., Pardalos P.M., (Eds.), *Handbook of Combinatorial Optimization - Supplement*, vol. A. Kluwer Academic Publisher, Dordrecht, p. 1—74.

[12] Massaro A., Pelillo M. (2003). "Matching Graphs by Pivoting". *Pattern Recognition Letters*. v.24 n.8, p.1099—1106.

[13] Barrow H.G., Burstall R.M. (1976). "Subgraph Isomorphism, Matching Relational Structures and Maximal Cliques". *Inform. Process. Lett.* 4 (4): 83—84.

[14] Motzkin T.S., Straus E.G. (1965). "Maxima for graphs and a new proof of a theorem of Turn". *Canad. J. Math.* 17 (4), 533—540.

[15] Bunke H. (1997). "On a relation between graph edit distance and maximum common subgraph". *Pattern Recognition Letters* 18, pp. 689—694.

[16] Finch AM., Wilson R.C., Hancock ER., "Symbolic Graph Matching with the EM Algorithm". *Pattern Recognition*, vol. 31, no. 11, pp. 1777—1790, 1998.

[17] Cross A.D., Hancock E.R. (1998). "Graph matching with a dual-step EM algorithm". *IEEE Trans. Pattern Anal. Machine Intell.* 20 (11), 1236—1253.

[18] Auwatanamongkol S. (2007). "Inexact graph matching using a genetic algorithm for image recognition". *Pattern Recognition Letters*, 28, 12, 1428—1437.

[19] Cross A.D.J., Wilson R.C.,Hancock E.R. (1997). "Inexact Graph Matching Using Genetic Search". *Pattern Recognition*, vol. 30, pp. 953—970.

[20] Singh M.,Chatterjee A.,Chaudhuri S. (1997). "Matching structural shape descriptions using genetic algorithms". *Pattern Recognition*, vol. 30, no. 9. pp. 1451—1462.

[21] Bengoetxea E., Larrañaga P., Bloch I., Perchant A., Boeres C. (2001). "Inexact graph matching by means of estimation of distribution algorithms". *Pattern Recognition*, 35: 2867—2880.

[22] Irniger C., Bunke H. (2005). "Decision trees for error-tolerant graph database filtering". *Proc. 5th Workshop on Graph-based Representations in Pattern Recognition*, pp. 301—312.

[23] Berman H. M.,Westbrook J., Feng Z., Gilliland G., Bhat T. N., Weissig H., Shindyalov I. N., Bourne P. E. (2000). "The Protein Data Bank". *Nucleic Acids Res.*, 28, 235—242.

Appendix A

Molecular Data File

Here we show a snippet of a CHARMM19 patterns file as an example of molecular data files, only one amino acid is shown, while the original file contains all 20 residues.

```
<model name='charmm19'>
    <class name='N' label='N' significance='1.0' />
    <class name='C' label='C' significance='1.0' />
    <class name='O' label='O' significance='1.0' />
    <class name='S' label='S' significance='1.0' />
    <class name='H' label='H' significance='0.0' />
    <pattern id='1' name='ALA'>
        <node id='213' class ='N'>
            <position x='-8.163' y='39.049' z='23.454'/>
            <neighbor id='214'/>
            <neighbor id='215'/>
        </node>
        <node id='214' class ='H'>
            <position x='-7.547' y='38.313' z='23.671'/>
            <neighbor id='213'/>
        </node>
        <node id='215' class ='C'>
            <position x='-9.576' y='38.761' z='23.269'/>
            <neighbor id='213'/>
            <neighbor id='216'/>
            <neighbor id='217'/>
        </node>
        <node id='216' class ='C'>
            <position x='-9.82' y='38.338' z='21.832'/>
            <neighbor id='215'/>
        </node>
        <node id='217' class ='C'>
```

```
            <position x='-10.09' y='37.643' z='24.162'/>
            <neighbor id='215'/>
            <neighbor id='218'/>
        </node>
        <node id='218' class ='O'>
            <position x='-9.365' y='36.701' z='24.485'/>
            <neighbor id='217'/>
        </node>
    </pattern>
    <pattern id='2' name='ARG'>

...

    </pattern>

...

</model>
```