Albert-Ludwigs-Universität Freiburg im Breisgau
Technische Fakultät - Institut Für Informatik

Lehrstuhl Für Bioinformatik
Prof. Dr. Rolf Backofen

De Novo Molecular Design Using Graph Kernels

Masterarbeit

Dragoş Alexandru Sorescu

December 2011 - May 2012

Dekan
Prof. Dr. Bernd Becker

Referenten
Prof. Dr. Rolf Backofen
Prof. Dr. Martin Riedmiller

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäßaus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

_____ _____

Ort, Datum Unterschrift

Acknowledgements

First I would like to thank Dr. Fabrizio Costa who proposed the subject of this thesis and closely supervised me through out the process. He came up with a lot of ideas that literally made this project possible. I would also like to thank him for the code of the NSPDK kernel which plays a major role in *GraSCoM* implementation. In the end I want to thank him for offering me the opportunity of working on such an interesting project. I cannot imagine a topic that I could have enjoyed more working on for my Master Thesis.

I would also like to thank Prof. Dr. Rolf Backofen for having me part of the group (as a student assistant) over the years of my study. I have learned a lot and I definitely enjoyed this experience.

Another big thank you needs to go to the group of Dr. Hauke Busch from the ZBSA for allowing me to run some of the experiments on their cluster.

I want to also thank my parents for encouraging and supporting me over the years.

A final but very special thanks goes to my girlfriend Ilinca, who supported me and made my days working on the thesis a lot nicer.

Zusammenfassung

Die Synthese von kleinen Molekülen, die die Wirkstoffeigenschaften von vorhandenen Medikamenten verbessern oder die wirksam in der Behandlung bisher nicht therapierbarer Krankheiten sind, ist eine sehr schwierige Aufgabe, in welche Pharma-Unternehmen enorme Ressourcen investieren. Dennoch zeigen Studien, dass nur einer von 5000 erforschten Wirkstoffkandidaten letztlich den Markt erreicht. Aus diesem Grund sind Pharma-Unternehmen auf der Suche nach Verfahren, mit denen unwirksame Kandidaten frühzeitig, und damit billig, erkannt und aussortiert werden können.

In diesem Zusammenhang sind computergestützte Methoden eine interessante Alternative, wenn sie die teuren und zeitaufwendigen Entwicklungsphasen Design, Synthese und Prüfung ersetzen können. Von diesen Methoden sind computergestützte Ansätze zum de-novo-Design von Molekülen besonders interessant, da sie beginnend von Null schrittweise neue molekularen Strukturen mit den gewünschten pharmakologischen Eigenschaften generieren können. Eine der größten Herausforderungen solcher Systeme ist dabei die Untersuchung des praktisch unendlichen chemischen Raumes.

In dieser Arbeit untersuchen wir eine neuartige Methode zur Navigation im Suchraum unter Nutzung einer parametrischen Familie von Teilgraphen genannt "Rooted neighborhood subgraphs"(RNS). Wir definieren die Struktur der molekularen Umgebung in Form von RNS-Substitutionen, wobei Ersetzungen auf das Vorhandensein von identischem Kontext beschränkt sind. Wir stellen einen neuartigen Ansatz namens "Graph Substitution Constructive Model (GraSCoM) vor, welcher eine lokale Suchstrategie verwendet. Die Bewertungsfunktion basierend auf einem schnellen Graph-Kernel-basierten diskriminierenden Modell, welche von Costa et al. in 2010 (F. Costa and K. D. Grave. Fast neighborhood subgraph pairwise distance kernel) eingeführt wurde. Die vorgestellte Methode wurde auf einem Datensatz mit 4337 Molekülen, welche als mutagen oder nicht mutagen klassifiziert sind, getestet. Die Evaluation zeigte, dass GraSCoM erfolgreich zum de-novo-Design von Molekülen genutzt werden kann.

Summary

Synthesis of small molecules that improve on the curative properties of existing drugs or that are effective in curing previously untreatable illnesses is a very hard task on which pharmaceutical companies are investing enormous amounts of resources. Despite this, studies show that only one out of 5000 screened drug candidates reaches the market and therefore the pharmaceutical companies are looking for *fail fast, fail cheap* solutions.

In this context, computational methods become an interesting alternative if they manage to replace the expensive and time consuming phases of design, synthesis and test. Among such methods, the computer-aided *de novo* molecular design approaches are particularly interesting as they produce from scratch novel molecular structures with desired pharmacological properties in an incremental fashion. One of the biggest challenges such systems have to face is the exploration of a practically infinite chemical space.

In this thesis we investigate a novel way of navigating the search space using a parametric family of subgraphs called rooted neighborhood subgraphs (RNS). We define the molecular neighborhood structure in terms of RNS substitutions, where the replacement is constrained on the presence of identical contexts. We introduce a novel approach that we call Graph Substitution Constructive Model (GraSCoM). GraSCoM proposes a local search strategy that uses as a scoring function a fast graph kernel-based discriminative model introduced by Costa et. al. 2010 (F. Costa and K. D. Grave. Fast neighborhood subgraph pairwise distance kernel). We tested the proposed approach on a dataset of 4337 molecules evaluated for their toxic properties and showed that GraSCoM is capable of performing successful *de novo* molecular design.

Contents

1 Introduction

Synthesis of small molecules that improve on the curative properties of existing drugs or that are effective in curing previously untreatable illnesses is a very hard task, a task on which pharmaceutical companies are investing enormous amounts of resources. Despite all the advances in technology and understanding biological systems, novel pharmaceutical discovery is still an expensive, difficult, and inefficient process, primarily due to the lack of models that accurately present the appropriate condition or that reflect the appropriate response[1].

On average, the development of a new drug takes 10 to 15 years and recent studies show that the research and development costs of a *new molecular entity* are of approximately US$1.8 billion, a number that is rising rapidly[21].

1.1 Motivation

During the drug discovery process, most of the effort is spent on investigating compounds that in the end turn out to be unsuitable because of bad ADMET[1] properties. As it turns out, only one in 5000 screened drugs reaches the market and because of this the pharmaceutical industry is looking for *"fail fast, fail cheap"* solutions (i.e. having fast, cheap methods for determining whether the drug candidate does or does not have suitable properties to become a drug).

In this context, computational methods become an interesting alternative since they can replace the time consuming and expensive drug discovery phases. Among such computational methods, those capable to perform *de novo* molecular design are particularly interesting[28]. These approaches produce in an incremental fashion, novel molecular structures with desired pharmacological properties from scratch.

1.2 Goal of this Thesis

Since *de novo* molecule design systems have to explore a virtually infinite search space, exhaustive searching is infeasible and usually the systems resort to local op-

[1] ADMET or ADME-Tox is the acronym for "Absorption, Distribution, Metabolism, Excretion, Toxicity", i.e. the set of properties used to describe the performance and activity of the pharmacological compound in the organism

timization strategies.

The goal of this thesis is to investigate novel molecular space search strategies by focusing on the fast graph kernel models based on the use of neighborhood subgraphs introduced in Costa et al. 2010[7]. Here, the subgraphs rooted in each vertex of a molecular graph and induced by all the vertices at bounded small distance from the root have been proved to yield information reach features.

1.3 Method Overview

The *de novo* molecule design system implemented for the purpose of this thesis, named *GraSCoM*, can perform two main tasks:

- Task 1: based on an initial molecule database (given in GSPAN format), *GraSCoM* computes its own databases of well-defined structures (cores and interfaces); these structures will be defined in sec. 4.1.
- Task 2: given a molecule seed set, a molecule train set (with the associated target file containing ± 1 values), the tool learns a model that can then try and predict if a new molecule is good or bad; using the databases produced via *Task 1* and the seed set, *GraSCoM* produces novel molecular structures which it then scores using the learned model.

1.4 Structure of this Thesis

For the beginning, in chapter 2 we will have an in depth look into *de novo* molecular design, discussing what are the challenges in this field and analyzing some of the state of the art methods. In chapter 3 the mathematical background of graph kernels is presented from the perspective of machine learning algorithms. Then, in chapter 4 the proposed method of this thesis will be explained. Finally, in chapter 5 we will evaluate the *GraSCoM* system, whereas chapter 6 contains the conclusion and some future work proposals.

2 Computer-Based *De Novo* Molecular Design

As discussed in chapter 1, the pharmaceutical industry has to face a lot of resource-related challenges when it comes to releasing an new drug to the market and despite all the efforts and research invested in improving the research and development productivity, the trends over the past years show a movement in the opposite direction[21]. One way for tackling this issue was introduced from the late '80s and early '90s, when the first automated *de novo* design techniques were conceived[18] and since then these computer-based techniques have became a solid complementary approach to high-throughput screening.

Although the *de novo* molecule design systems still have to face a lot of challenges, their most important asset is the ability of generating novel molecular structures in a time- and cost-efficient manner. Even though there is research going into the design of peptides and other polymeric structures, the focus of this study is on designing small, drug-like molecules and thus we will focus on this area for the rest of the thesis.

There are three main main phases when it comes to drug development: design, synthesis and test. Any *de novo* design system tries to effectively replace the first two phases (Fig. 2.1), and to do this, it needs to decide on strategies for three problems[28]:

- assembly: how to construct candidate compounds?
- scoring: how to evaluate their potential quality?
- search: how to efficiently sample the search space?

2.1 Constructing Candidate Compounds

There are two approaches when it comes to constructing new candidate structures, that relate to the building blocks used for the "artificial" design phase: the atom- or the fragment-based approach.

While the atom-based approach is superior to the fragment-based approach because of the structural variety that it can produce, it comes with a drawback: the

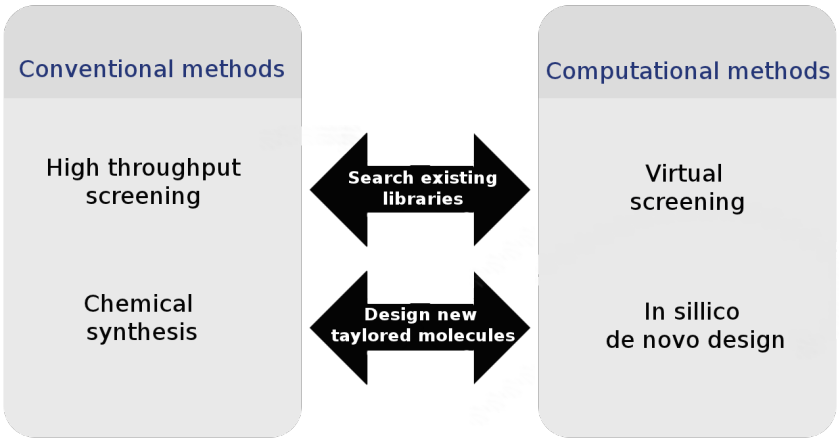

Figure 2.1: Computational counterparts for conventional drug discovery methods[11]

search space becomes infinite (as theoretically any part could be reached) and thus the biggest challenge is selecting the best/suitable candidates.

On the other hand, the fragment-based approach has a smaller search space to deal with as it performs a "meaningful reduction" by focusing on relevant areas (i.e. parts that regularly appear in compounds and/or are of interest for the particular study). Another advantage that this second approach has is that it's the designer's choice of how to define these fragments and it usually happens that a fragment can consist of only one atom, making the first approach a subset of the second.

The pioneer *de novo* systems used an atom-based approach, but as it comes hand-in-hand with a combinatorial explosion problem, most of the systems today use fragments as basic building blocks.

There are several approaches for performing candidate structure sampling (some of which are presented in Fig. 2.2):

- linking: key interaction sites of the receptor are saturated with preferred building blocks that once in place, are automatically connected by suitable linker fragments
- growing: starting with a single block positioned at one of the key interaction sites, new blocks are sequentially added to yield the final compound
- lattice-based sampling: using a grid representation of potential positions of ligand atoms within the binding pocket, ligand candidates are formed using the points that lie along the shortest path through the lattice connecting interaction sites
- random structure mutation

- transitions driven by molecular dynamics (MD) simulations: an active site is filled with atoms, which adopt low energy configurations during a MD simulation; bonds between atoms of randomly chosen building blocks can be formed and subsequently broken based on geometric and energetic considerations[23]
- graph-based sampling: evolutionary algorithms are applied to optimize topological molecular graphs

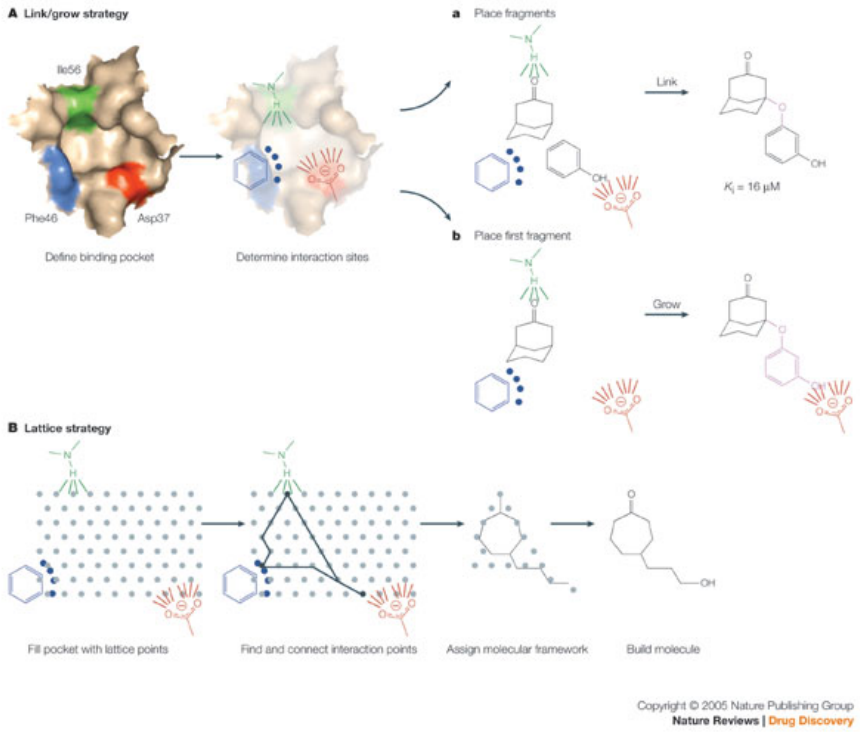

Figure 2.2: Structure sampling: linking, growing and lattice-based assembly[28]

2.2 Evaluating Candidate Compounds

While deciding on how to construct the candidate compounds is more of a design/approach issue, but with direct impact on the final results, one critical aspect is how these compounds are then scored. Thus, the scoring function has a central role in the system, as it not only judges the quality of the new molecules, but implicitly assigns fitness values to them and guides the design process through the search space. The molecules used in the evaluation function's induction phase are usually (very) different from the "new generation" of molecules that the system wants to score, and thus the reliability of such a function could be debatable.

These evaluation functions are mainly based on the so-called *primary target constraints* which are generated from all the information related to the ligand-receptor interaction that the function can collect. Based on the source used for collecting these constraints, the design strategies can be divided in two:

- receptor-based: constraints are gathered from the 3D receptor structure which makes these approaches limited to target proteins with known 3D structure; however, this is not always the case for relevant pharmaceutical targets
- ligand-based: constraints are gathered from known ligands of the particular target

When it comes to the *receptor-based* approaches all of them attempt to approximate the binding free energy and can be subdivided into three different types of scoring functions[28]:

- explicit force-field functions (most costly approach)
- empirical scoring functions: fast methods using a weighted sum of individual ligand-receptor interaction types approach(supplemented by penalty terms)
- knowledge-based scoring functions: not so popular among *de novo* design system, the approach scores molecules based on statistical analysis of ligand-receptor complex structures where the frequencies of each possible pair of atoms in contact with each other are determined; interactions occurring more frequently than randomly expected are considered attractive and the other way around.

However, generally the goal of *de novo* molecular design goes beyond generating molecules with high binding affinities, as there are many other important properties to be analyzed in a molecule (i.e. the AMDET property set). Because of this, usually so-called *secondary target constraints* (constraints other than the binding affinity) are introduced, making the scoring function a weighted sum over the primary and secondary constraints.

2.3 Sampling the Search Space

As mentioned in the previous subsection, the fact that the evaluation function decides on which molecules are good to keep for the next iteration of the algorithm and which should be discarded, makes it act as a "virtual chemist" that decides which parts of the search space should be explored next. This brings us to another major aspect of computer-based *de novo* molecular design: how to search in the (virtually infinite) chemical space ?

The size of the search space for small organic molecules has been estimated to be in the order of 10^{60}[4]. Even though this number probably over-estimates the real size of the search space relevant for drug discovery (as it does not account for chemical

stability and drug-likeness)[11], a brute-force searching remains unfeasible. Instead, local optimization strategies are used to sample the search space, making the solution converge to a "practical" optimum (which isn't necessarily the global optimum).

Even though deterministic approaches exist, most of the algorithms used in this area are non-deterministic and rely on some sort of stochastic structure optimization[28]. This does not mean that the exploration of the search space is done randomly, but that instead these algorithms have a way of narrowing it towards a goal state by including as much chemical knowledge as possible about the structure of this space. Details of how this navigation through the space search is done will be presented in the next subsection (sec. 2.4).

Any algorithm based on the local search approach works on the idea of moving from a solution to the next in the space of candidate solutions (by performing local changes to the current set) until an optimal solution (set) has been found or a threshold was reached. In our case of infinite search space, moving from a current solution to the next cannot be done arbitrarily, but within the local neighborhood of the current solution.

It is important to mention that the success of such artificially evolving systems requires a fundamental characteristic of the underlying search space, namely the *Principle of Strong Causality*. Defined in the context of *Evolution Strategies*, this principle states that the variation operators (mutation and recombination) should perform search steps in such manner that small search steps result in small fitness changes and vice versa[27]. The principle has been reformulated for the field of drug design as the *Chemical Similarity Principle*[17] and it requires for a systematic compound optimization algorithm to have a *neighborhood behavior*.[1]

There are two types of designs one can use when implementing such a local (chemical) space search algorithm:

- positive design: restricts the local search space to the areas that it considers to be more likely for producing drug-like molecules
- negative design: defines and avoid areas of the search space containing adverse properties and unwanted structures

2.4 Navigating the Search Space

Regardless of how a program tries to generate the molecules, almost all the time they try to mimic the iterative process of drug discovery and consist of the following steps:

[1]The property of a *structural space* to map onto the *activity space* in a way ensuring that neighboring points in the former are likely to correspond to neighboring points in the latter[13].

1. select the initial seed set of molecules (real molecules, with good ADMET properties)
2. apply changes to the seed set in order to produce new molecules
3. score the new set of molecules
4. apply a fitness function; the fittest molecules become the new seed set
5. if the new set of molecules is "optimal" or (time/iteration/etc) threshold was reached, terminate; otherwise go back to step 1

One big problem of *de novo* design when trying to generate new compounds (step 2) is the combinatorial explosion. Because of this, the algorithms use combinatorial search strategies that reduce the search space and have an efficient way of navigating over it. As depicted in Fig. 2.3, the search space for the 2D structure of new compounds can be represented by a tree, with the root being the initial state, the leaves representing the end states (containing the candidate compounds) and all other nodes corresponding to intermediate states. The goal of the search strategy is to find a path from the initial to the end state (in the context of this figure a depth-first search strategy is used).

Several *de novo* design programs use breadth-first search (BFS) or depth-first search strategies (DFS). The DFS technique does not guarantee finding the best solution, as it advances trough the levels of the search graph by expanding the "best" node from the current level (expansion that could be guided by the score of the node, by chance or by a combination of both), significantly pruning the search space. On the other hand, the BFS technique examines all nodes from each level and as a result does guarantee to find the optimal solution. In order to be able to perform such an exhaustive search, most of these programs reduce the search space by using the linking method for structure assembly. Others perform an "altered" BFS, like RASSE[19] (an approach that allows no more than 100 such fragments to become the templates for further growing), or SPROUT[9] which uses the A^* algorithm.

While LEGEND[20] uses pure random sampling (also known as Monte Carlo search), CONCEPTS[22] was the first to use the Monte Carlo search with a Metropolis criterion. This combined approach works as follows: every newly generated compound gets evaluated; if it has a better score then the original compound, it gets through to the next round; otherwise, it can still get accepted with a probability based on the difference between the score of the new molecule and the seed molecule and a random number.

Finally, the last big class of algorithms used in *de novo* design that we will briefly discuss are the *evolutionary algorithms* (EA). These approaches use mechanisms inspired by biological evolution: reproduction, mutation, recombination and selection. There exist a number of different techniques used for this type of algorithms, out of which three are relevant with respect to our field of study:

- genetic algorithms: populations of fixed-length strings (binary or real values) are used for encoding the phenotype (molecular structures); in the context of EA, these strings are referred to as chromosomes

- genetic programing: the phenotype is encoded as a tree, facilitating the extension and contraction of the chromosome

- evolution strategies: instead of using a chromosome to encode the phenotypic features when performing structure generation, the phenotype itself becomes a molecular graph

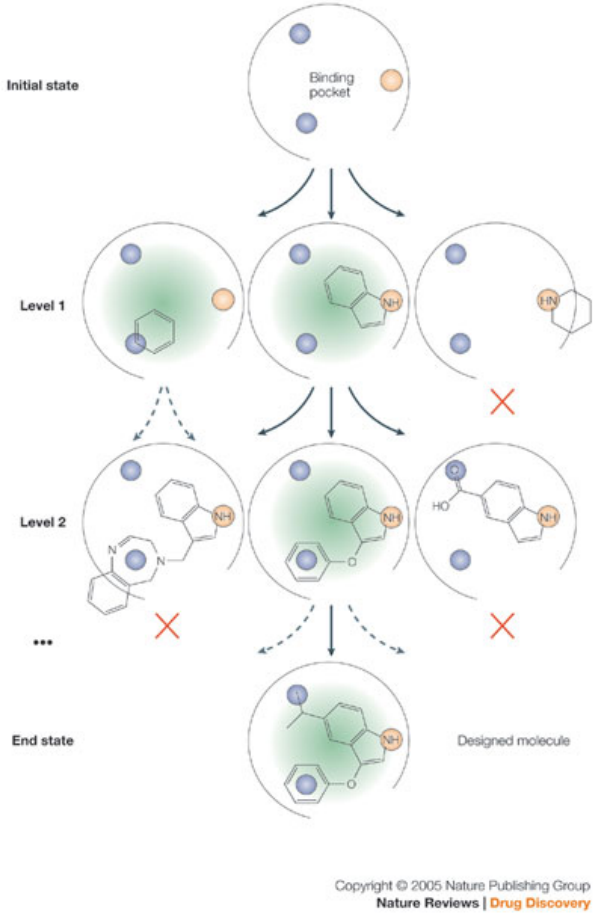

Figure 2.3: Input (initial state):binding pocket+predicted key interaction sites; Yellow circles indicate sites for a ligand hydrogen-bond donor and blue circles represent regions for lipophilic ligand parts. Partial structures are partial solutions (light green) if they satisfy the primary target constraints. Some of the partial structures are rejected due to boundary violation (L1&L2) or due to mismatching interaction types (L2).[28]

2.5 Successful *de novo* Design Approaches

There have been several successful approaches to *de novo* molecular design and in this section we will cover some of them in order to emphasize the potential of such techniques.

Introduced 20 years ago, LUDI[3] is an early linking approach that showed promising results. First the interaction sites are defined either by a set of rules or as discrete positions in space suitable to form hydrogen bonds or to fill a hydrophobic pocket[3]. Then fragments from a user-inputted database are placed inside the binding cavity. In the final step LUDI applies an empirical scoring function and connects the best fragments to form the new molecule. Using this approach, LUDI generated compounds that with some manual optimization to ease synthesis, proved to be inhibitors of HIV-RT[2].

The *evolutionary algorithm* SYNOPSYS[30] was able to produce 10 (synthesized and tested) new inhibitors of the HIV-1 protein reverse transcriptase. The method's success was achieved by starting with a database of existing molecules and enforcing synthesizability to the new generation of compounds by using only chemical reactions.

Based on a fragment-based approach, BREED[24] was the first approach to use the structural information emerging from the field of high-throughput structural biology. With a fast, automated approach for generating new inhibitors by joining fragments of known ligands, BREED produced over 100 novel, chemically viable, protease and kinase inhibitors for the HIV-1 virus.

We won't cover all *de novo* successful programs (like LEGEND[20], TOPAS[29], BOMB[31] and others) even if they all contain very interesting approaches with promising results. We should however mention that there are numerous other examples that maybe didn't produce immediate results, but did introduce interesting new ideas for approaching the problem.

2.6 Expectations and Limitations

Throughout the previous sections (sec. 2.1 - sec. 2.5) we gave an overview on how an "*in silico* laboratory" for performing *de novo* molecular design tries to replace the traditional methods of drug design. Given the properties of such a system, there are a couple of points worth mentioning related to the expectations and limitations of such a setup:

[2]Human Immunodeficiency Virus Reverse Transcriptase

- there is no guarantee that the produced molecules will be chemically valid (synthetically feasible with drug- or lead-like properties)
- as the approach tries to generate new molecules from existing knowledge (used to guide the design process through the search space), it could be that interesting candidates don't pass the threshold of the fitness function or don't even get discovered at all during the iterative process; a way of preventing this is performing stepwise knowledge extrapolation
- the system will rarely yield novel lead structures with the desired pharmacological behavior in the first instance; however they could represent an inspiring example that medical chemists can further improve
- if the compounds produced by such a system represent a reasonable suggestion for investigating new bioisoteres, or it performed a *scaffold-hop*[3], we can consider that the system fulfilled its purpose

[3]The process of identifying isofunctional structures with different backbone architectures.

3 Graph Kernels

This chapter presents the theoretical background of graph kernels, an approach allowing efficient graph comparison that finds its application in a variety of problems where the item of study can be represented as a graph, including *de novo* molecular design. It is important to mention that even if in the literature sometimes the term of *graph kernels* is used to describe kernels between two nodes in one graph, in this thesis we will use *graph kernels* to denote functions that compare two graphs to each other (i.e. defined on $\mathcal{G} \times \mathcal{G}$).

3.1 Using Kernels for Nonlinear Classification

Generally speaking, kernels are used in algebra to measure the degree to which a homomorphism fails to be injective[8]. In mathematical terms, we define a kernel as a continuous and symmetric function k, that given two arguments x and x', it maps them to a real value (that measures the similarity between them):

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$
$$k(x, x') = k(x', x), \qquad \forall x, x' \in \mathcal{X}$$

In the machine learning (ML) field, the main interest is to design algorithms that based on empirical data, are able learn and make successful predictions about new, unseen data. The linear case has been studied in depth from the early days of ML and is now very well developed. However, most of the time the (real world) data that we would like to learn from, requires the use of nonlinear methods for inducing the dependencies needed to make successful predictions.

Using kernel methods, it is possible to map the nonlinear input data into a different feature space where linear classifiers can be used. As this mapping can substantially increase the number of features to consider, one would like to still be able to evaluate the data efficiently. A common way to do this is by using the *kernel trick*.

3.1.1 The Kernel Trick

First we will introduce a series of definitions that will help understanding how this technique works.

Definition 3.1.1 (Cauchy Sequence)**.** Given a metric space $X = (x, \rho)$, a sequence $\{x_n\}_{n=1}^{\infty}$ is called a *Cauchy sequence* if

$$\forall \varepsilon > 0, \text{ there } \exists n_0 = n_0(\varepsilon) \text{ such that for } \rho(x_n, x_m) < \varepsilon, \forall n, m > n_0 \tag{3.1}$$

In other words, a *Cauchy sequence* is a sequence whose elements become arbitrarly close to each other as the sequence progresses.

Definition 3.1.2 (Complete metric space)**.** A metric space $X = (x, \rho)$ is called *complete* if every sequence in X converges in X.

Definition 3.1.3 (Hilbert space)**.** A *Hilbert space* is a real or complex inner product space that is also a complete metric space with respect to the distance function induced by the inner product.

Usually, the empirical data that a ML algorithm uses in order to learn a model consists of a vector of *inputs* x_i, alongside with the *targets* vector y_i. Mathematically, we specify this as $(x_1, y_1), ..., (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$. Once a model was learned using this data, we would like to be able to make predictions for unseen data $x \in \mathcal{X}$, i.e. to choose $y \in \mathcal{Y}$ such that (x, y) is *similar* to the training examples.

To do this we first define a function Φ that maps our data into a Hilbert space $\mathcal{H}$ (also known as the *feature space*). A good choice of this function (also known as the *feature map*) would make the input data in $\mathcal{X}$ be linearly separable in $\mathcal{H}$, as shown in the simple example in Fig. 3.1.
The question rising now is how do we still keep our ML algorithm efficient if we now have to process data in a higher-dimensional space? The answer lies in *Mercer's Theorem*, but in order to understand how it is connected to kernel functions, we first need to make a couple of definitions.

Definition 3.1.4 (Kernel matrix)**.** Given a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and a set of patterns $x_1, ..., x_n \in \mathcal{X}$, the $n \times n$ matrix K with elements

$$K_{i,j} = k(x_j, x_i), \quad i, j = [1, n]$$

is called the *kernel matrix*(or Gram matrix) of k with respect to $x_1, ..., x_n$.

Definition 3.1.5 (Positive definite matrix)**.** A real $n \times n$ symmetric matrix $K_{i,j}$ is called *positive definite* if

$$\sum_{i,j} c_i c_j K_{i,j} \geq 0, \quad \forall c_i \in \mathbb{R}$$

Definition 3.1.6 (Positive definite kernel)**.** Given $\mathcal{X} \neq \emptyset$, we call $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a *positive definite kernel* if for $\forall n \in \mathbb{N}$ and $\forall x_i \in \mathcal{X}, i = [1, n]$ gives rise to a positive definite Gram matrix.

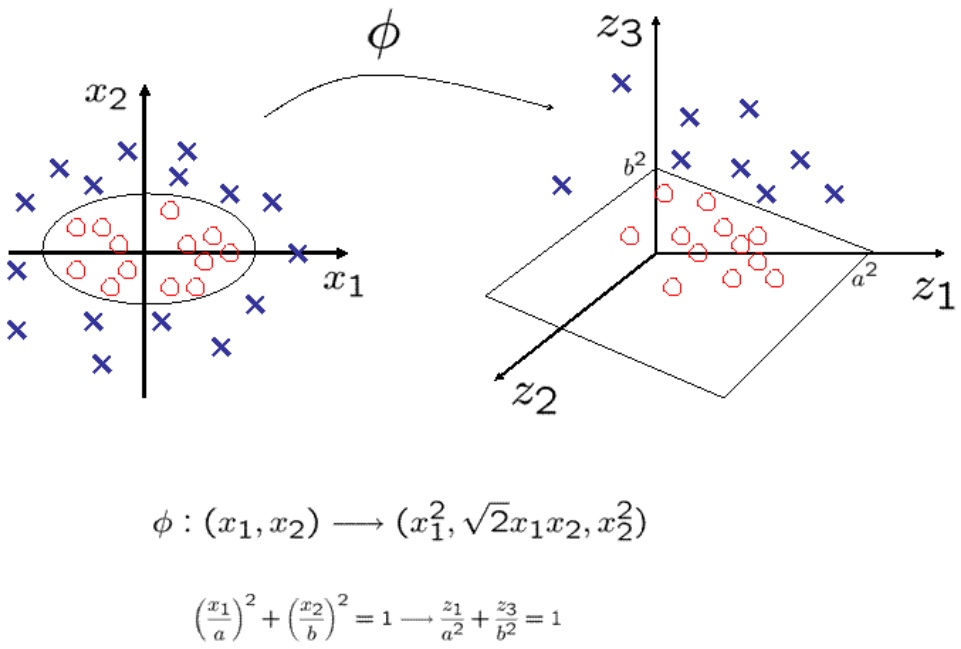

Figure 3.1: Mapping data from $\mathbb{R}^2$ into $\mathbb{R}^3$ (Carlos C. Rodriguez, The Kernel Trick)

Theorem 1 (Mercer's Theorem)**.** A continuous, symmetric function k can be expressed as an inner product for some feature map Φ, if and only if k is positive semidefinite, i.e

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \quad \Phi : \mathcal{X} \to \mathcal{H}_K$$
$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle \Leftrightarrow \int k(x, x')g(x)g(x')dxdx' \geq 0, \forall g$$

In other words, for any positive definite kernel there exists a Hilbert space $\mathcal{H}_K$ of real valued functions defined on $\mathcal{X}$ and a feature map Φ, such that the value of the kernel function is equal to the inner product of the mappings $\Phi(x), \Phi(x')$ in $\mathcal{H}_K$.

With *Mercer's Theorem* we finally arrived to the essence of the *kernel trick*, which suggests that instead of having to explicitly map the data with a Φ function and take the dot product, one can just use any positive definite kernel, without having to care at all about the feature map.

Once introduced to the field of learning, kernel methods became more and more popular due to the nice properties they have, like efficiency in computation or flexibility with respect to the input data (doesn't have to be linearly separable, doesn't have to be represented as a vector). This flexibility property is something that all

ML algorithms can benefit from, as for example one would like to have multiple, heterogeneous data sources. For this purpose *multiple kernel learning* approaches have been developed, allowing intelligent ways of recombining results from a large number of kernels[2].

3.2 Kernels for Graphs

When it comes to defining kernels for measuring the similarity between highly structured data, graphs have become of more and more interest as they can represent a lot of real world data like biological sequences, phylogenetic trees, RNA structures, natural language texts, semi-structured data (such as HTML and XML) and much more[15]. Of particular interest for the topic of this thesis is representing chemical compounds as undirected, labeled graphs where the atom types are encoded in the vertex labels and the bond types are encoded in the edge labels).

3.2.1 Graph Generalities

In the following we will introduce some general graph definitions (that will be used in to coming parts of the chapter) while following the notations in *Handbook of Graph Theory*[10]. We define a graph as an ordered pair $G = (V, E)$ comprising of a set of vertices V and a set of edges E and the following graph-related notations:

$V(G)$−the set of vertices of graph G
$E(G)$−the set of edges of graph G
$\mathcal{D}(u, v)$−the distance between $v \in V(G)$ and $u \in V(G)$
(defined as the length of the shortest path between them)
$\mathcal{N}_r(v)$−the neighborhood of radius $r \in \mathbb{N}$ of a vertex $v \in V(G)$
$=\{u|\mathcal{D}(u, v) \leq r, u \in V(G)\}$

Definition 3.2.1 (Induced subgraph)**.** Given a graph G, we define the *induced subgraph* on a set of vertices $W = \{w_1, ..., w_k\}$, the graph G' for which $V(G') = W$ and $E(G') = \{(u, v)|u, v \in W, (u, v) \in E(G)\}$

Definition 3.2.2 (Neighborhood subgraph)**.** We define the *neighborhood subgraph* of radius r of vertex v denoted by $\mathcal{N}_r^v$, as the subgraph induced by $\mathcal{N}_r(v)$.

Definition 3.2.3 (Labeled Graph)**.** Given a function $\mathcal{L}$ that maps a vertex/edge to its corresponding label, $G = (V, E, \mathcal{L})$ is a *labeled graph* (i.e. a graph whose vertices and/or edges are labeled, possibly with repetitions, with symbols from a finite alphabet).

Definition 3.2.4 (Isomorphic graphs)**.** Two graphs $G = (V, E)$ and $G' = (V', E')$ are called *isomorphic* if there exists a bijection $\Phi : V' \to V'$ such that

$$\text{for } \forall u, v \in V(G) \text{ there } \exists (u, v) \in E(G) \Leftrightarrow (\Phi(v), \Phi(u)) \in E(G')$$

We denote the isomorphism between G and G' by $G \simeq G'$. Two labeled graphs are isomorphic if there is an isomorphism that also preserves the label information, i.e. $\mathcal{L}(\Phi(v)) = \mathcal{L}(v)$.

3.2.2 Graph Kernel Types

While there are many different classes of graph kernels, of particular interest for this thesis are the *R-convolution Kernels*. Thus, there will be a more detailed presentation of the latter and just a brief talk about the rest.

Random Walk Kernels These type of kernels measure the similarity of two graphs by performing random walks[1] on both and measuring the similarity between the labels of the visited nodes. While they are indeed a good way to measure graph similarity (by taking into account the whole structure of the graph), they lack efficiency, usually being somewhere around the magnitude of $\mathcal{O}(n^6)$. If in some situations the complexity can be improved, there are two known problems that are not as easy to deal with as dealing with one, worsens the other and vice-versa:

- tottering: as the walks cannot restrict repetitions of nodes/edges and they are probability based, it could happen that due to cycles or undirected edges, the same vertices get considered too many times and artificially improve the similarity score
- halting: to avoid cycles, a series of decaying factors need to be used to down-weight the longer walks; however this usually causes longer walks to be (completely) ignored and thus less of the graphs structure is taken into account.

Subtree-Pattern Kernels By considering all pairs of vertices from both graphs and iteratively comparing their neighborhoods, these kernels count some *subtree patterns* in order to produce the similarity score. The patterns they look for are: rooted subgraphs such that there is a tree homomorphic to the subgraph, and the number of distinct children of both root nodes in the pattern and in the tree to be the same[26]. These kernels also face the tottering problem and as it turns out, the run-time complexity is even worse than the one of the random walk kernels.

[1] A random walk is a graph traversal where the starting node, the next neighbor to visit each step and the node that ends the walk are chosen based on some probability.

Cyclic-Pattern Kernels The idea is to decompose the graphs into cyclic patterns and count how many are shared between the two graphs. The approach is efficient when restricted to graph databases in which there exists a natural small upper bound on the number of simple cycles for almost every graph[14] but turns out to be NP-hard when applied to a general graph.

Fingerprint and Depth First Search Kernels A series of kernels specifically created to work on small molecular graphs. They take use of some generalized versions of molecular fingerprints[2] and depth first search exploration of depth d[25].

R-Convolution Kernels In 1999, D. Haussler introduced the *R-convolution kernels* as a way to construct kernels on sets whose elements are discrete (i.e. strings, graphs)[12] and till today they remain the most used approach when dealing with such instances. The core idea behind *R-Convolution kernels* is that composite objects can be expressed via their constituent parts and the relation R between them.

Assume $x \in \mathcal{X}$ is the composite structure and $x_1, ..., x_D$ are its parts (not necessarily disjoint), with $x_d \in \mathcal{X}_d$ for $d = [1, D]$, $D \in \mathbb{N}^+$ and each $\mathcal{X}_d$ is a countable set. Let R be a boolean predicate defined as

$$R : \mathcal{X}_1 \times ... \times \mathcal{X}_D \times \mathcal{X} \to \{true, false\},$$
$$R(x_1, ..., x_D, x) = \begin{cases} true, & \text{if } x_1, ..., x_D \text{ is a valid decomposition of } x \\ false, & \text{otherwise} \end{cases}$$

We can now define R^{-1} as the inverse relation that given a composite structure, it yields all valid decompositions: $R^{-1}(x) = \{x_1, ..., x_D | R(x_1, ..., x_D, x) = true\}$.

In the original paper[12] it is proven that if two instances $x, y \in \mathcal{X}$ admit a valid decomposition as $x_1, ..., x_D$ and $y_1, ..., y_D$ and

$$\text{there } \exists k_d : \mathcal{X}_d \times \mathcal{X}_d \to \mathbb{R}, \quad \forall d = [1, D],$$
$$\text{then } k(x, y) = \sum_{\substack{x_1,...,x_D \in R^{-1}(x) \\ y_1,...,y_D \in R^{-1}(y)}} \prod_{d=1}^{D} k_d(x_d, y_d)$$

is a valid kernel called a *convolution* or *decomposition* kernel. Thus, a convolution kernel is a sum (over all possible decompositions of a composite structure) of the product of valid kernels over the constituent parts of that structure. The power and flexibility of this approach comes from the vagueness surrounding the relation function R, allowing the definition of a multitude of kernels just by changing the decomposition.

[2] A molecular fingerprint is a bit-string representation for the structure of a molecule.

In this context, probably one of the first approaches that comes to mind is decomposing two graphs into all their subgraphs and compare them pairwise. This is exactly the idea behind the *all-subgraphs kernel*.

Definition 3.2.5 (All-Subgraphs Kernel)**.** Given two graphs G and G', the all-subgraphs kernel is defined as

$$k_{subgraph}(G, G') = \sum_{S \subseteq G} \sum_{S' \subseteq G'} k_{isomorphism}(S, S'),$$

$$\text{where } k_{isomorphism}(S, S') = \begin{cases} 1, & \text{if } S \simeq S' \\ 0, & \text{otherwise} \end{cases}$$

While the Neighborhood Subgraph Pairwise Distance Kernel (NSPDK) is also an R-convolution kernel, because it is the one we use in our design approach, we will present it in detail in the following section.

3.3 Neighborhood Subgraph Pairwise Distance Kernel

The NSPDK is the kernel used in this thesis for measuring the similarity between two graphs. The relation function R that it uses, decomposes the graph in all paris of neighborhood subgraphs of radius r that are at distance d from each other (as in the example from Fig. 3.2).

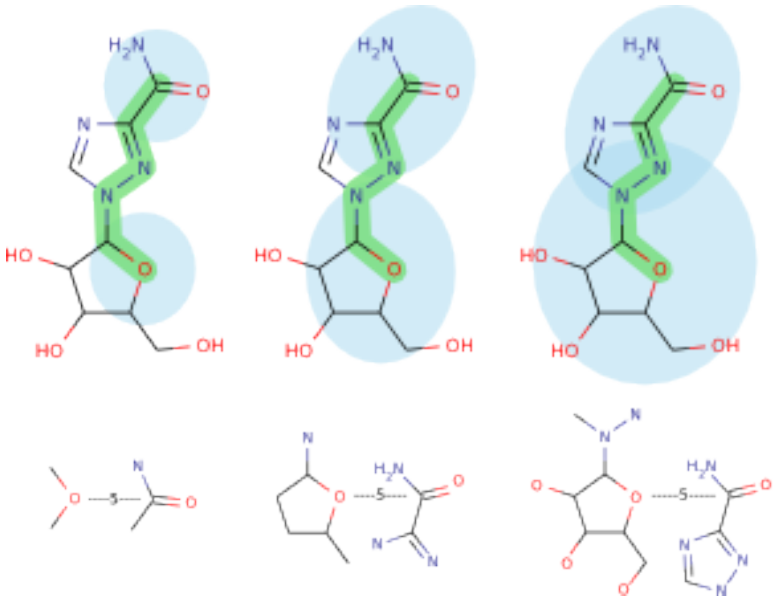

Figure 3.2: Pairs of neighborhood subgraphs for radius r=1,2,3 and distance d=1[7].

3.3.1 Definition

Formally, given a graph G and two rooted subgraphs A^v and B^v that are isomorphic to some $\mathcal{N}_r$, if $\mathcal{D}(u,v) = d$ the $R_{r,d}$ can be defined as

$$R_{r,d}(A^v, B^u, G) = \begin{cases} true, & \text{iff } A^v, B^u \subseteq \{\mathcal{N}_r^v : v \in V(G)\} \\ false, & \text{otherwise} \end{cases}$$

Definition 3.3.1 (Exact matching kernel)**.** The *exact matching kernel* $\delta(G, G')$ is defined as

$$\delta(G, G') = \begin{cases} 1, & \text{if } G \simeq G' \\ 0, & \text{otherwise} \end{cases}$$

The decomposition kernel $k_{r,d}$ on the relation $R_{r,d}$ that counts neighboring graphs of radius r at distance d between two graph G and G' can now be defined as

$$k_{r,d} : \mathcal{G} \times \mathcal{G} \to \mathbb{N}^+$$
$$k_{r,d}(G, G') = \sum_{\substack{A_v, B_u \in R_{r,d}^{-1}(G) \\ A'_{v'}, B'_{u'} \in R_{r,d}^{-1}(G')}} \delta(A_v, A'_{v'})\delta(B_u, B'_{u'})$$

Definition 3.3.2 (NSPDK)**.** Using $k_{r,d}$, the *Neighborhood Subgraph Pairwise Distance Kernel* is a sum over all possible radiuses and distances:

$$K(G, G') = \sum_r \sum_d k_{r,d}(G, G')$$

For efficiency in computation and relevant similarities scores, the approach in this thesis follows the suggestion of Costa et. al.[7] and uses a zero-extension of K by setting an upper limit for both r and d, as well as a normalized version of $k_{r,d}$ that ensures an equal weight to all relations, regardless of the size of the induced part sets. The final version of the NSPDK used here becomes

$$K(G, G') = \sum_{r=0}^{r^*} \sum_{d=0}^{d^*} \frac{k_{r,d}(G, G')}{\sqrt{k_{r,d}(G, G)k_{r,d}(G', G')}}$$

4 Method

This chapter will present in detail the approach for *de novo* molecular design used by the *GraSCoM* system. First, sec. 4.1 will explain the concepts that are fundamental for the novel molecular space search strategy used, strategy that will be detailed in sec. 4.2.4. The last part of the chapter presents the challenges that appeared when designing and implementing this system and how they were tackled.

4.1 Terms and Definitions

In order to understand the method, one needs to know the concepts it uses. Inspired by the notion of *substitutable* languages[6], we introduce two well-defined graph structures which we call *core* and *interface*, and the procedure called *core substitution*.

4.1.1 Interface

For a vertex v, we define as the *interface border* at radius r the set of vertices that are exactly at distance r from v, that is

$$\mathcal{I}_v^r(G) = \{u | \mathcal{D}(u, v) = r; u, v \in V(G); r \in \mathbb{N}^+\}$$

We now introduce the notion of the *interface* rooted in v of radius r and thickness t as the subgraph induced by all the nodes inside the neighborhood of radius t of the vertices on the interface border. Formally,

$$\mathcal{I}_v^{r,t}(G) = \{u | u \in \mathcal{N}_t(p); p \in \mathcal{I}_v^r(G); v, u, p \in V(G); r \in \mathbb{N}^+\}$$

Given the molecule in Fig. 4.4, an interface rooted in vertex 2 (highlighted with a red circle) of radius 5 and thickness 2 will look like the one in Fig. 4.1.

4.1.2 Core

Given a graph G, we rename the notion of neighborhood subgraph of radius $r \in \mathbb{N}$ induced by vertex $v \in V(G)$ (as introduced 3.2.2) as the *core* of radius r with root

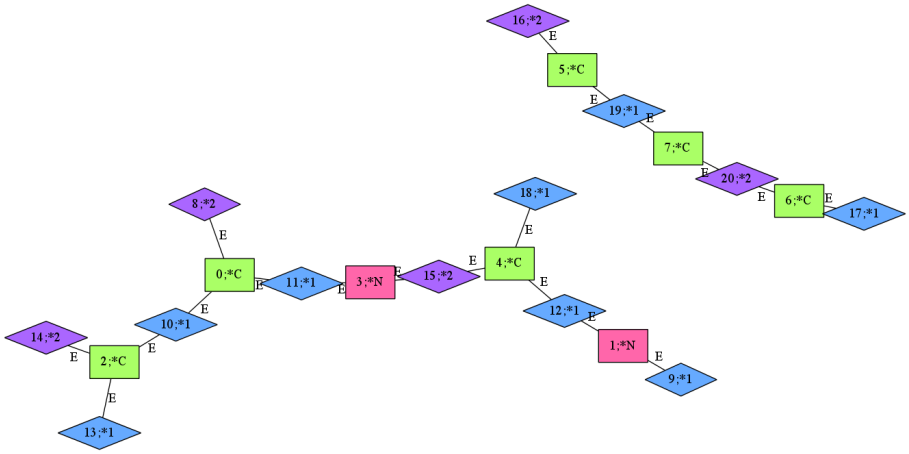

Figure 4.1: Interface of radius 5 and thickness 2 rooted in node 2 of the graph in Fig. 4.4.

v, denoted by $\mathcal{C}_v^r$. Thus, $\mathcal{C}_r^v$ is the subgraph induced by the set of vertices at distance less than or equal to r from v. We do this renaming because we believe the term *core* to be more suggestive with respect to the procedure we will introduce next. Moreover, in our approach a core is defined only in relation to its corresponding interface. Fig. 4.2 shows the core of radius 5 rooted in node 2 (highlighted with a red circle) of the graph presented in Fig. 4.4.

4.1.3 Core Substitution

The *core substitution* procedure is how we call the approach we use for generating new chemical compounds. In the context of languages, Clark et. al.[6] introduced the notion of substitutable words (and languages) in the sense that two words are substitutable if they have the same context. We generalize this idea to graphs and we say that if we have two graphs in which we are able to identify the same interface, then we can substitute their corresponding cores.

4.1.4 Hypergraphs

It is important to mention that if we want to use this core substitution procedure and just apply it as is on regular graphs, we lose any information that could be stored in the edges. This is of course undesired, especially in the case of chemical compounds where the equivalent graph representation contains edges labeled with the type of chemical bonds.

For performing the substitution without loss of information on any type of graph, internally we apply a simple transformation to a hypergraph in which the edges from the original graph become a node that stores the label of the edge it represents and

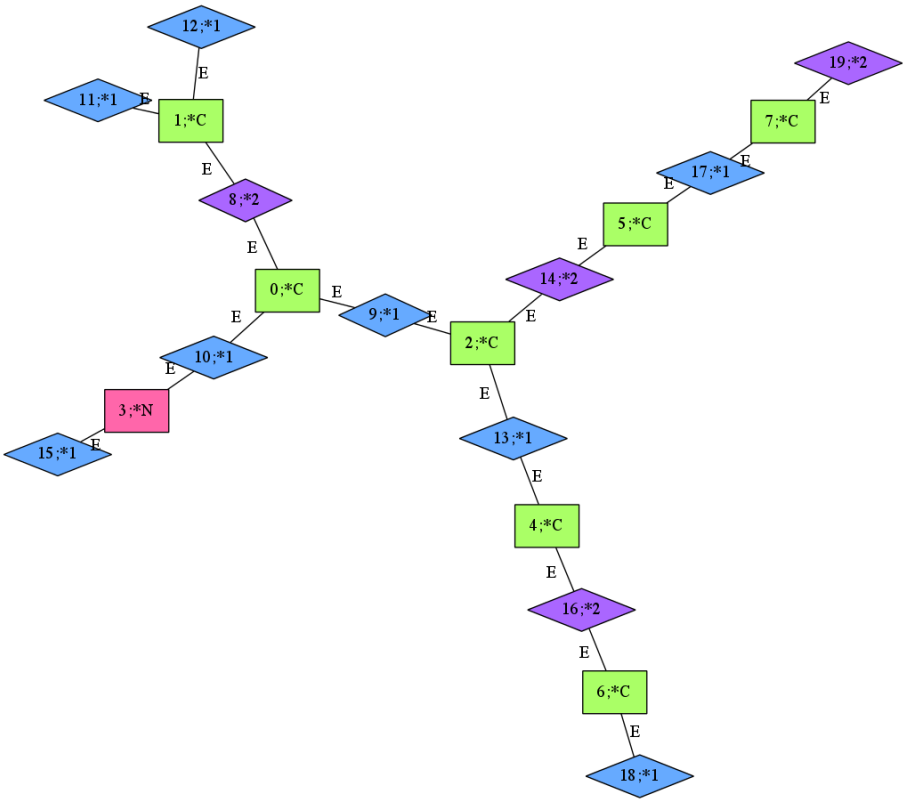

Figure 4.2: The core of radius 5 rooted in node 2 of the graph in Fig. 4.4.

is connected to both of its initial source and destination vertices. We denote the set of vertices in the hypergraph H corresponding to real vertices representing atoms in the original graph G by $V_{V(G)}(H)$ and the set of "edge" nodes by $V_{E(G)}(H)$. Thus, $V(H) = V_{V(G)}(H) \cup V_{E(G)}(H)$. Fig. 4.3 presents a molecule in it's graph representation form while Fig. 4.4 shows the equivalent hypergraph transformation.

We now need to make a couple of natural restrictions that ensure we benefit from this transformation:

1. we can root for defining cores/interfaces only in nodes in $v \in V_{V(G)}(H)$
2. $1 \leq r, r \in \{2k+1 | \forall k \in \mathbb{N}^+\}$: alongside with restriction 1, ensures that the nodes on the interface border (which are also the border of the core) are in $v \in V_{E(G)}(H)$
3. $0 \leq t < r, t \in \{2k | \forall k \in \mathbb{N}\}$: alongside with restriction 1, ensures that the end nodes of the interface are also in $v \in V_{E(G)}(H)$ and that the interface does not include the whole core (which would make core substitution useless)

Therefore, by applying the hypergraph transformation and satisfying the conditions above, we now ensure that when we extract any of the two structures, we also preserve the edge information from the original graph.

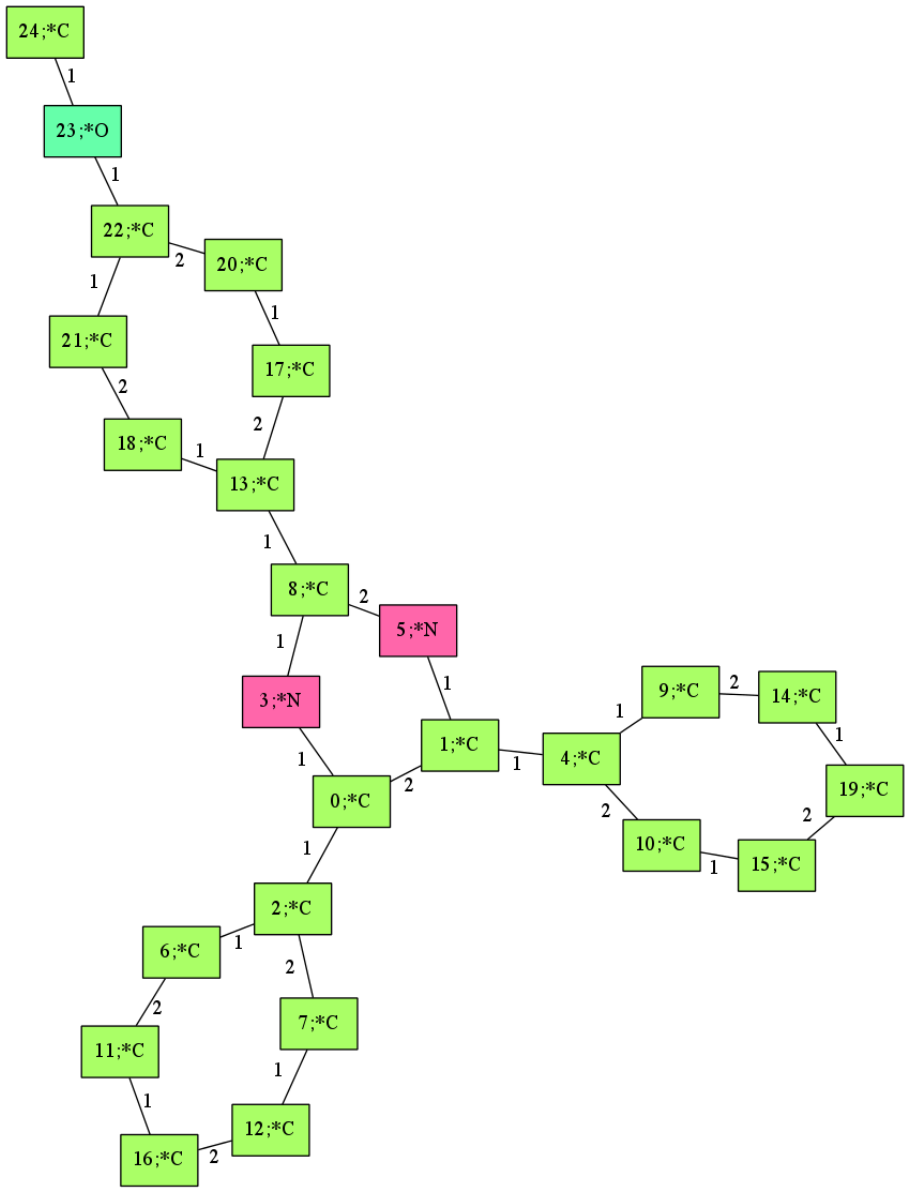

Figure 4.3: A molecule read in from the dataset as a graph.

4.2 Graph Substitution Constructive Model (GraSCoM)

As presented in the chapter 2, every de novo design system has to present solutions to three problems, all of crucial importance with respect to the final results: how does it search for new compounds, how does it assemble them and how does it score new ones. In the following we will discuss each of these issues from the perspective of *GraSCoM*, the system implemented for the purpose of this thesis.

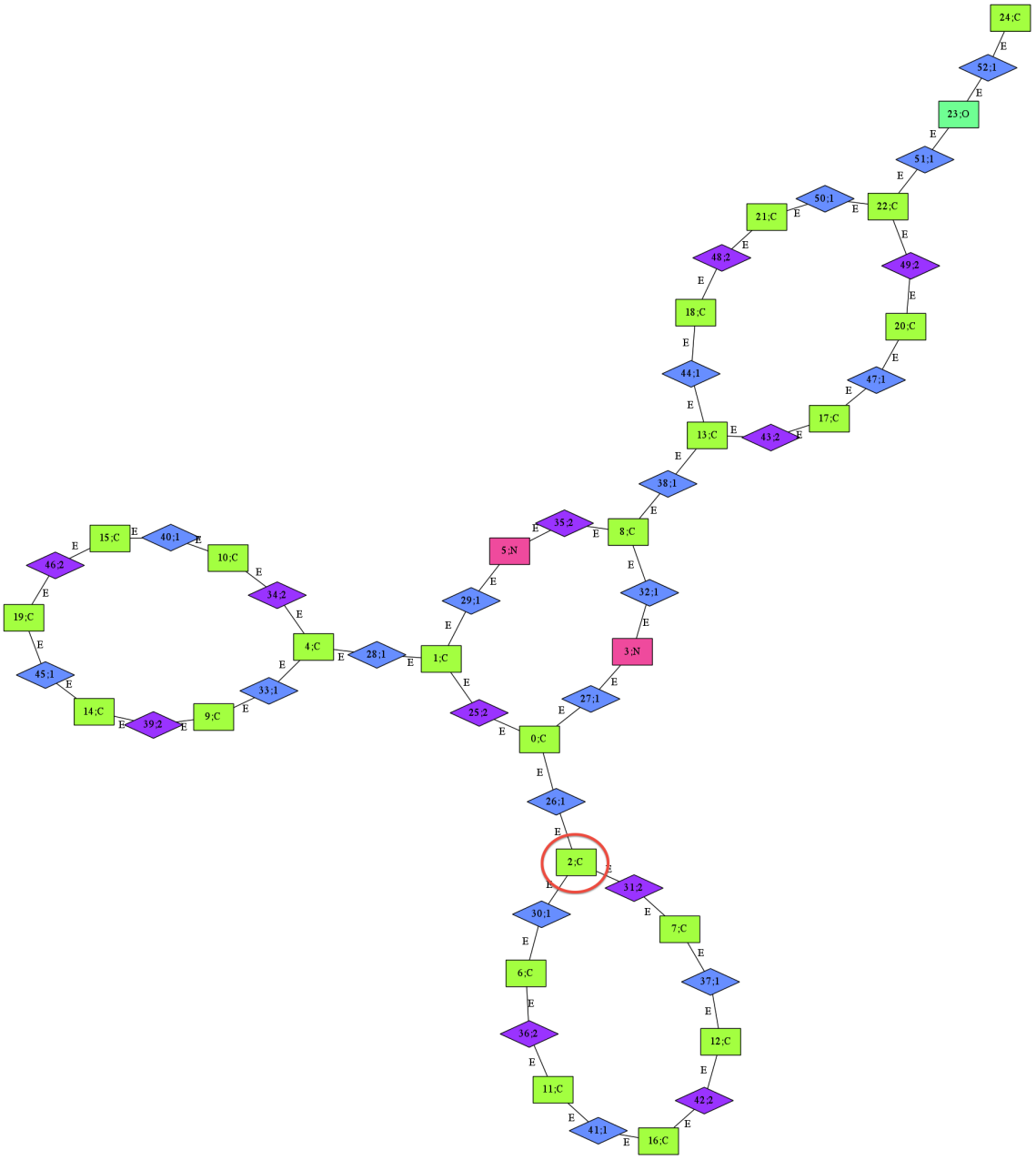

Figure 4.4: The equivalent hypergraph transformation to the molecule in Fig. 4.3

4.2.1 Searching

GraSCoM proposes a novel way for navigating over the chemical space. Given an input (train) set of valid molecules (internally represented as 2D hypergraphs as described in sec. 4.1.4), for each molecule we iterate over the complete set of nodes $v \in V_{V(G)}(H)$ and extract for different radiuses and thicknesses interfaces together with their corresponding cores, and store them in a meaningful way into a database. At the end of this database computation step, we will have stored interfaces along-

side with a list of cores that were found for each particular interface.

We then start with an initial seed set and again for every compound, we iterate over a set of vertices and using the same values for radiuses and thicknesses, we identify interfaces. If we find an interface that is stored into our database, we then get the previously computed list of valid cores for that interface and perform the core substitution procedure. When doing this we check that we are not substitution identical cores, and thus ensure that we are not copying molecules over to the next generation. Nevertheless, it could happen that a molecule from the seed set is generated by substituting cores in a different molecule or that the same molecule is generated in different iteration steps. For this type of situations, we consider a new compound only if we haven't done it before.

This type of navigating over the search space is very interesting. While being a local search technique, the "width" of the neighborhood considered varies depending on how frequently the interfaces identified in the current molecule were found in the list of molecules used to populate the database. Thus, it could be considered as a special type of the *adaptive* neighborhood width approach, where the area of the neighborhood to consider is not necessarily controlled by an internal parameter of the system, but by the similarity to other valid molecular structures.

There are however ways of influencing how the search space sampling is done. The straight forward way of doing it is by controlling the maximum allowed value for the radius (Max_r) and the minimum one for the thickness (Min_t). These are both very powerful as the difference $CF = Max_r - Min_t$ is directly correlated to what could be referred as the *creativity factor*. When a big CF is used, it makes the algorithm perform substitutions of big cores while having a thin interface, and thus make massive changes to the structure of the compound. On the other hand, a small CF would make the approach much more conservative in the sense that it would perform a substitution of a small core only when the structure of the thick interface is the same and thus requesting the current molecule to be structurally very similar to the molecule that was considered when extracting the corresponding interface and core.

Another way of sampling the search space the system allows is based on computing a score for each vertex that tells what influence does a particular node have with respect to the final score of that compound. It then considers only the nodes with a negative influence (or the top $n > 0$ such nodes) and tries to search for known interfaces that are rooted in these vertices. The way these scores are obtained will be discussed in sec. 4.2.3.

4.2.2 Assembling

New compounds are assembled by substituting cores (i.e. fragments) identified in valid chemical compounds and therefore the approach is fragment-based. However, with a radius $r = 1$ and thickness $t = 0$ it gets the capabilities of an atom-based technique as the cores will consist of an atom connected to its "edge" vertices and the interfaces will be only of thickness $t = 0$ and thus consist of "edge" nodes. Actually for any pair $\langle$radius, thickness$\rangle$ where the difference between the two parameters is 1, the method will perform an atom substitution. The difference will consist in how big the thickness is, and thus how much structure similarity do we require to perform the core substitution.

4.2.3 Scoring

The scoring problem is the last of the three problems a *de novo* design system has to deal with. After performing a local search and assembling the new molecules, the algorithm needs to score them and then retain the best ones as a seed for the next generation.

Support Vector Machine One of the most popular and best performing ways for doing classification analysis in ML is by using support vector machines (SVM). The general idea of this approach is that given a set of training examples each marked as belonging to one of two classes (usually denoted as positives and negatives), the SVM constructs a hyperplane in a higher dimensional space that separates the two classes of input data. The need for this mapping makes the use of kernels functions be the most efficient and natural choice.

Stochastic Gradient Descent To be able to make predictions, a ML algorithm first needs to learn a model. The question is what technique to apply for this purpose. One popular choice is the *gradient descent* where at each iteration the algorithm performs a step proportional to the negative gradient of the learning function in the current point. As this is done by looking at all the training data, serious computational issues can arise when dealing with big datasets. A similar, but much faster technique is the *stochastic gradient approach* where instead of looking at the whole dataset, at each step only one randomly picked example is taken into account. If intuitively this approach would suggest a worse performance and in fact convergence is much more noisier, the literature shows that when dealing with large scale datasets it actually performs better because more training examples can be processed in the allowed time[5].

For the purpose of scoring molecules' and vertices' contributions to the scores, *GraSCoM* uses an SVM with an NSPDK kernel and a stochastic gradient approach for

learning. As explained before in sec. 4.2.1, for each molecule in the seed set, a number of new candidate compounds are generated. Out of this number we keep only top k, where k is either a fixed number, or the number of new compounds that have a score greater than the highest score obtained by a molecule in the previous round. With the latter constraint, we restrict the algorithm to produce better molecules each round (with respect to the scoring function). For the first iteration we restrict the score to be greater than 0. Once the complete seed set was processed, all top k molecules are merged, sorted and then top m are kept as seed molecules for the next iteration, but top $s > m$ are saved to a file for computing performance measures later on. All three parameters k, m, s are given as an input to the system.

4.2.4 The Method in a Nutshell

Now that we've explained how *GraSCoM* addresses the three key problems of computer based *de novo* design, we will present a summarized pseudocode description to have a complete overview of the approach. Algorithm 1 and 2 present the procedure for identifying core-interface pairs (one inducing the features into the graph construction model while the other performs the core substitution procedure). Algorithms 3 presents the vertex selection policy and finally algorithm 4 introduces the main loop of the procedure.

Algorithm 1 Identify all cores and interfaces rooted in a vertex v and induce them into the graph construction model M_{GC}.

```
procedure InduceGCModel( G, v, Min_r, Max_r, Min_t, M_GC )
    for r ← Min_r; r ≤ Max_r; r ← r + 2 do
        C_r^v ← ExtractCore( G, v, r )
        C_ID ← ComputeGraphHashID( C_r^v(G) )
        SetGraphID( C_r^v, C_ID )
        for t ← Min_t; t < r; t ← t + 2 do
            I_v^{r,t} ← ExtractInterface( G, v, r, t )
            I_ID ← ComputeGraphHashID( I_v^{r,t}(G) )
            SetGraphID( I_v^{r,t}(G), I_ID )
            AddInterfaceCorePairToGCModel( I_v^{r,t}, C_r^v, M_GC )
        end for
    end for
end procedure
```

4.3 Identifying Graphs

As suggested by the $ComputeGraphHashID(G)$ function inside the pseudocode of algorithms 1 and 5, we need a way of producing unique and meaningful ids for each

Algorithm 2 Identify all cores and interfaces rooted in a vertex v and perform core substitutions when the current graph G contains an interface also found inside the graph construction model M_{GC}; return the rewired graphs.

procedure CONSTRUCTGSET(G, v, Min_r, Max_r, Min_t, M_{GC})
  for $r \leftarrow Min_r$; $r \leq Max_r$; $r \leftarrow r + 2$ **do**
    $\mathcal{C}_r^v \leftarrow$ ExtractCore(G, v, r)
    $\mathcal{C}_{ID} \leftarrow$ ComputeGraphHashID($\mathcal{C}_r^v(G)$)
    for $t \leftarrow Min_t$; $t < r$; $t \leftarrow t + 2$ **do**
      $\mathcal{I}_v^{r,t} \leftarrow$ ExtractInterface(G, v, r, t)
      $\mathcal{I}_{ID} \leftarrow$ ComputeGraphHashID($\mathcal{I}_v^{r,t}(G)$)
      $\mathcal{I}_M \leftarrow$ GetInterface($\mathcal{I}_{ID}$, M)
      $\mathcal{C} \leftarrow$ GetCores($\mathcal{I}_M$)
      for all $\mathcal{C}_i \in \mathcal{C}$ **do**
        if $\mathcal{C}_{ID} \neq$ GetGraphHashID($\mathcal{C}_i$) **then**
          $\bar{G} \leftarrow \bar{G} \cup$ SubstituteCore(G, $\mathcal{I}_v^{r,t}$, $\mathcal{C}_r^v$, $\mathcal{C}_i$)
        end if
      end for
    end for
  end for
  return $\bar{G}$
end procedure

core/interface we identify (meaningful with respect to the labels of the hypergraphs vertices and its structure). For this, we use the *exact matching kernel* introduced in Costa et. al.[7].

4.3.1 Hash Codes for Exact Matching

The purpose of assigning such meaningful ids is to later be able to efficiently check if two graphs (i.e. two interfaces) are isomorphic. The exact matching kernel efficiently checks for graph equality by using a two-step approach: first it computes a graph invariant encoding for both graphs using a label function, and then it generates a hash out of this encoding such that in the end the comparison between the two graphs is reduced to comparing two strings.

Encoding the graph as a string As mentioned above, the graph invariant encoding is done via a label function $\mathcal{L}^g : \mathcal{G}_h \rightarrow \sum^*$, where $\mathcal{G}_h$ is the set of rooted subgraphs and $\sum^*$ is the set of strings over the finite alphabet $\sum$. To understand how it works we need to introduce two more label functions, one for the vertices and one for the edges.

Algorithm 3 Given a graph G, a scoring function $f : G \times \{2^{|V(G)|}\} \mapsto \mathbb{R}$ and a *Policy*, return the appropriate vertices.

```
procedure VertexSelectionPolicy( G, f, Policy )
    if Policy == 0 then                                   ▷ Root in all vertices
        V̄ ← V(G)
    else
        for all v ∈ V(G) do
            idToScore[v] ← f(G, v)
        end for
        SortMapBySecondElement( idToScore )
        V' ←GetMapFirstElementList( idToScore )
        if Policy > 0 then      ▷ Root in top policy vertices with negative score
            iterator ← 1
            for all v_i ∈ V' do
                if iterator ≤ Policy then
                    V̄ ← V̄ ∪ {v_i}
                    iterator ← iterator + 1
                else
                    break
                end if
            end for
        else                                  ▷ Root in all vertices with negative score
            for all v_i ∈ V' do
                V̄ ← V̄ ∪ {v_i}
            end for
        end if
    end if
    return V̄
end procedure
```

Algorithm 4 The main GraSCoM loop first induces the graph construction model M_{GC} and learns a scoring function. This function is then used alongside with a *Policy* for selecting the vertex set in which the algorithm further roots to identify known interfaces. Each constructed $\bar{G}$ set is then filtered by getting the top $GSetBeamSize$ scoring graphs. At the end of each iteration the filtered reunion of the best scoring graphs, $\bar{G}_F$ is filtered similarly for the top $SetBeamSize$ to produce the *SeedSet* for the next iteration.

procedure ConstructGSet($GCModelInput$, Min_r, Max_r, Min_t, $MaxIterationNumber$, $SupervisedSample$, $SupervisedSampleTarget$, $InitSeedSet$, $GSetBeamSize$, $SetBeamSize$, M_{GC}, $Policy$)
  for all $G \in GCModelInput$ **do**
    for all $v \in G$ **do**
      InduceGCModel(G, v, Min_r, Max_r, Min_t, M_{GC})
    end for
  end for
  f $\leftarrow$ LearnScoreFunction($SupervisedSample$, $SupervisedSampleTarget$)
  $SeedSet \leftarrow InitSeedSet$
  for $i \leftarrow 1 \ldots MaxIterationNumber$ **do**
    for all $G_i \in seedSet$ **do**
      for all $v \in$VertexSelectionPolicy($G_i, f, Policy$) **do**
        $\bar{G} \leftarrow$ ConstructGSet(G, v, Min_r, Max_r, Min_t, M_{GC})
      end for
      $\bar{G}_F \leftarrow \bar{G}_F \cup$ FilterGCSet($\bar{G}, GSetBeamSize$)
    end for
    $SeedSet \leftarrow$ FilterGCSet($\bar{G}_F, SetBeamSize$)
  end for
  return $\bar{G}_F$
end procedure

The vertex labeling function $\mathcal{L}^n$ assigns to each vertex $v \in V(G_h)$ the concatenation of all lexicographically sorted distance-label pairs $\langle \mathcal{D}(u,v), \mathcal{L}(u) \rangle, \forall u \in G_h$, alongside with the distance to the root $\mathcal{D}(u,h)$ (given that G_h is a rooted graph). Using this, the edge labeling function $\mathcal{L}^e$ assigns to each edge $(u,v) \in E(G_h)$ the triplet $\langle \mathcal{L}^n(u), \mathcal{L}^n(v), \mathcal{L}(u,v) \rangle$. In the end, $\mathcal{L}^g$ assigns to the rooted graph G_h the lexicographically sorted list of $\mathcal{L}^e(u,v), \forall (u,v) \in E(G_h)$.

For efficient comparison, this graph encoding is then mapped to a 32-bit integer using a Merkle-Damgård hashing function $H : \sum^* \rightarrow \mathbb{N}$, and thus verifying if two graphs are isomorphic becomes a numerical equality test between $H(\mathcal{L}^g(G_h)$ and $H(\mathcal{L}^g({G'}_{h'})$.

The *graph isomorphism problem* is known as one of the few NP problems for which finding polynomial algorithms is still an open issue. Because of this, there are different techniques used that either work on specific types of graphs, or give approximate solutions. Using the hashing function is what makes the exact matching kernel give fast solutions, but at the same time it could be that different, non-isomorphic graphs get the same id. As mentioned in the paper, the error introduced by hash collisions is negligible[7], but nevertheless it needs to be accounted for.

4.3.2 Mapping Interface Vertices

Another issue that needs to be tackled regards the core substitution procedure (i.e. rewiring a database core to the current interface). When we store a ⟨interface, core⟩ pair into our database we also keep a mapping between the nodes that are on the interface border. This way we know how to rewire the core back.

However, in the second part of the *GraSCoM* algorithm (where we produce novel compounds), as explained before when we identify a interface in the current seed molecule we want to rewire all the corresponding cores we have in our database. The problem is that to do this we need to know the correspondence between the vertex ids of the current and the database interface.

So how do we know which node corresponds to which? Well, once again we take use of the labeling function $\mathcal{L}^n$ we introduced before. Given that these labels are produced only with respect to the features of the graph and not the ids, for each interface we can lexicographically sort the list of labels and then perform a *hash induced*[1] breadth-first view (BFV) on both graphs starting from the first node in each of the lists that hasn't been already visited. We do this until all nodes have been visited (i.e. we traversed all the connected components). Algorithms 5 and

[1]We call a *hash induced* BFV a visiting of the nodes in a breadth-first manner, where after expanding the current nodes' child instead of always choosing in a predefined (left-right) manner, we first visit the vertex that has the smallest hash id.

6explain this procedure in pseudocode.

Using this approach we obtain a structure-based vertex mapping between the two interfaces and thus know how to perform the rewiring.

Algorithm 5 Hash-induced connected component BFV
$StartId$: the id of the node where to start the BFV
$IdToHash$: id $\mapsto$ hashed label

```
procedure HashInducedBFView( StartId, IdToHash )
    Q.push( StartId )
    while ¬Q.empty() do
        v ← Q.pop()                          ▷ Get first element from queue
        visitedNodes ← visitedNodes ∪ {v}
        next ← NextNodesToVisit( v )          ▷ Get child nodes of v
        next.SortByHashId( IdToHash )         ▷ Get sorted by hash list of ids
        for all v ∈ next do
            Q.push( v )
        end for
    end while
    return visitedNodes
end procedure
```

4.3.3 Extended Interface

The last important procedure that needs to be introduced is what we called *extending* the interface and is performed before generating the id in order to encode more information inside this value. For every interface $\mathcal{I}_v^{r,t}(G)$ we do this simple but crucial two-step method: first instead of considering $\mathcal{I}_v^{r,t}(G)$ we consider $\mathcal{I}_v^{r,t+1}(G)$ (artificially increasing the thickness by 1) and then we perform a relabeling of all vertices.

The relabeling mean adding context information to the interface by following these sequential steps:

1. $\forall v \in \mathcal{I}_v^r(G)$ the original label is prepended with a "i_"
2. $\forall v \in \mathcal{I}_v^{r,t}(G) - \mathcal{I}_v^r(G) \wedge v \in \mathcal{C}_v^r$ the original label is prepended with a "c_"
3. $\forall v \in \mathcal{I}_v^{r,t+1}(G) - \mathcal{I}_v^{r,t}(G)$, if $\mathcal{C}_v^r$ relabel with "c_", otherwise with "e_".

As one can notice, while for all vertices that are part of the interface the relabeling adds extra information to the existing one (by marking if the node is inside the core, on the interface border, or between), for the artificially considered nodes we discard the existing label information and just mark them with respect to their appartenance to the core. By doing this, a lot of context information is added to the labels

Algorithm 6 Hash-induced dual interface breath-first view; returns $v_2 \mapsto v_1$ map
$\mathcal{I}_C/\mathcal{I}_{DB}$: current/database interface
$\mathcal{I}_C\_Hashes/\mathcal{I}_{DB}\_Hashes$: id $\mapsto$ hashed label

```
procedure DualBFView( I_C, I_DB, I_C_Hashes, I_DB_Hashes )
    for all v ∈ I_DB do
        alreadyExplored_1[v] ← false
    end for
    for all v ∈ I_C do
        alreadyExplored_2[v] ← false
    end for
    SortLexicographicallyByHashID( I_c_Hashes )
    SortLexicographicallyByHashID( I_DB_Hashes )
    n_1 ← NextUnvisitedNode( I_DB, alreadyExplored_1 )
    n_2 ← NextUnvisitedNode( I_C, alreadyExplored_2 )
    while n_1 ≠ ∅ ∧ n_2 ≠ ∅ do
        alreadyExplored_1[n_1] ← true
        alreadyExplored_2[n_2] ← true
        visited_1 ← HashInducedBFView( n_1, I_DB )
        visited_2 ← HashInducedBFView( n_2, I_C )
        for i ← 1 ... visited_1.size() do
            alreadyExplored_1[visited_1[i]] ← true
            alreadyExplored_2[visited_2[i]] ← true
            vertexMapping[visited_2[i]] ← visited_2[i]
        end for
        n_1 ← NextUnvisitedNode( I_DB, alreadyExplored_1 )
        n_2 ← NextUnvisitedNode( I_C, alreadyExplored_2 )
    end while
    return vertexMapping
end procedure
```

and implicitly to the final hash id.

This transformation ensures a successful hash-induced breath first view (and mapping of the vertices) as explained in sec. 4.3.2. At the same time it causes a stricter isomorphism between two interfaces and thus the whole core substitution procedure to be more rigorous with respect to the structure of the molecule.

5 Evaluation

In this chapter we will present and discuss various results obtained on *GraSCoM* under different parameter and experiment setups.

Dataset The experiments we have performed all used the Bursi[16] dataset that contains a list of 4337 molecular structures with corresponding Ames data (2401 mutagens and 1936 nonmutagens). The dataset focuses on the study of toxic properties which are usually related to subparts of the chemical structure.

5.1 Methodology

Limited, positive seeds For all the experiments we used as an input for computing the database a set of positive molecules. This is of course due to the fact that the algorithm is going to identify and extract cores and interfaces from these compounds and we want to populate our database with structures that come from valid sources. Due to computational expenses (time and memory) which are caused by a combinatorial explosion that happens every iteration, we only keep from one iteration to the next a limited subset of the newly generated molecules. We will refer in detail to this issue in sec. 5.3.

Training the model In order to score the molecules (or the contribution of each vertex) we need a model. For this purpose in every setup we learned a model from the complete Bursi dataset.

Cross-validation Generally we used the cross-validation technique for estimating the accuracy of the model. In this sense, we performed a 70/30 split of the dataset into train and test. We then learned a model from the complete train set and performed predictions on the test set. These predictions were then confronted with the known, true target values. The resulting measures represent the reference values.

Iterations The default parameter we used for the number of iterations *GraSCoM* should perform is 5. For every experiment setup we then performed 5 repetitions, every time generating different "random" train and test sets. While we do call a random number generator for selecting the molecules to consider every round, we also use a fixed seed every time (the current repetition number) in order to produce the same sets over different runs of the experiment. This makes the results over different setups comparable.

Performance measures We used the *perf*[1] software to compute the following three performance measures:

- Receiver Operating Characteristic(ROC). The ROC curve is generated by plotting the fraction of true positives out of the positives against the fraction of false positives out of the negatives at different thresholds.
- F-measure(PRF)= $\frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision + Recall}}$
- Accuracy (ACC) = $\frac{\text{True Positives + True Negatives}}{\text{True Positives + False Positives + True Negatives + False Negatives}}$
- Area under the precision recall curve (APR): Given that we plot the precision as a function of the recall (i.e. p(r)), APR=$\int_0^1 p(r)\,\mathrm{d}r$

For evaluating the quality of our newly generated molecules we used two approaches:

- merge the train negatives with the new molecules considered as positives (further refer to this as the *full train set merge*) or,
- merge the whole train set with the new molecules considered as positives (further referred to as the *negative train set merge*).

The purpose of the *full train set merge experiment* is to see what is the influence that the newly generated compounds have on the predictive performance of the model. The second experiment is meant to test how "powerful" the newly generated molecules are, by considering them the only positives in the set used to induce the model. Once we've done the merging, we learn a model that we then use to try and classify the test set. The predicted values and the true values are given as an input to *perf* which returns the measures mentioned above.

5.2 Experiments

With the types of experiments we've set up, we tried to answer different questions with regards to the behavior of the system under various parameters.

[1] http://osmot.cs.cornell.edu/kddcup/software.html

5.2.1 Size of the Input Dataset

For this type of experiment we made the only exception from the cross-validation scheme and instead of performing 5 repetitions of the experiment with 70/30 randomly split sets, we start from the same train/test sets every repetition, but we gradually increase the size of the train set proportional to the iteration number. This way, initially we will extract structures from one fifth of the train set in the first repetition, two fifths in the second and so on until the 5^{th} repetition uses the whole train set (which in this setup consists of 1696 positive molecules).

By doing this we are keeping the same test set every repetition while varying only the size of the train set from which we extract core/interface structures. Fig. 5.1-Fig. 5.8 present the measures collected under the two "merging" approaches for generating the new train set that induces the model. Apart from the APR and PRF in the *negative train set merge* evaluation, all other performance measures show what we would have expected, i.e. that the more knowledge out of which the system can learn, the better it can perform.

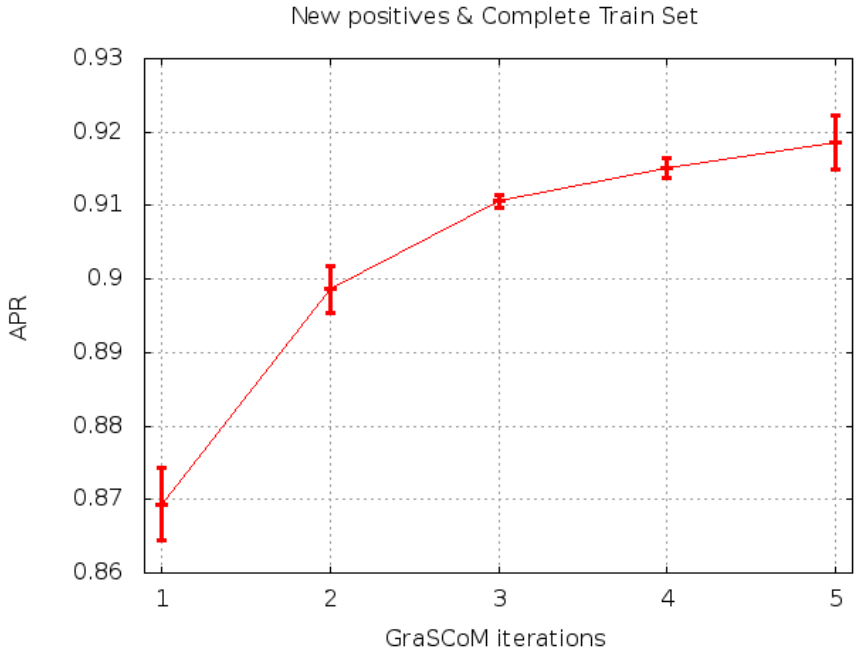

Figure 5.1: APR value measured for the "Size of the input dataset" experiment

Figure 5.2: APR value for the "Size of the input dataset" experiment

Figure 5.3: ACC value measured for the "Size of the input dataset" experiment

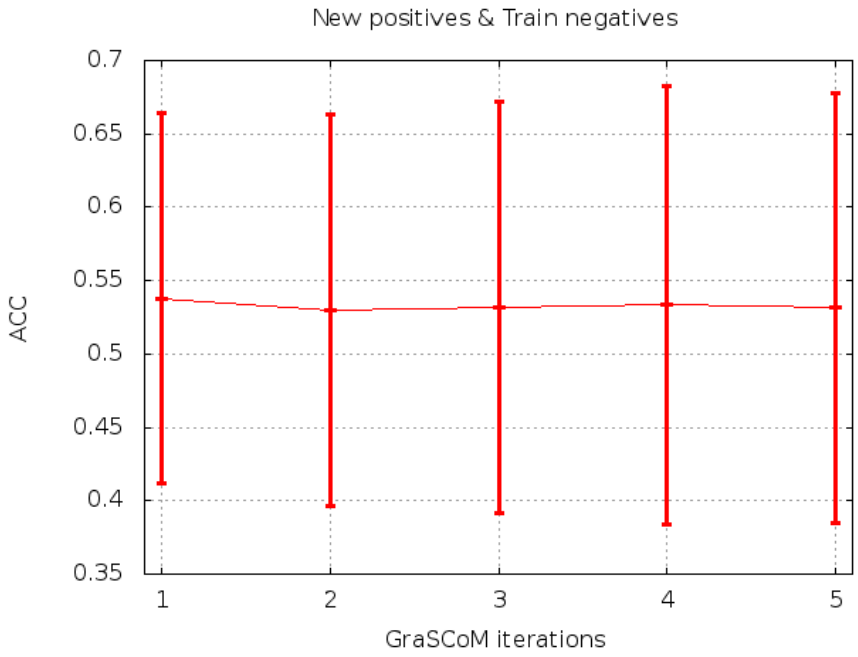

Figure 5.4: ACC value measured for the "Size of the input dataset" experiment

Figure 5.5: PRF value measured for the "Size of the input dataset" experiment

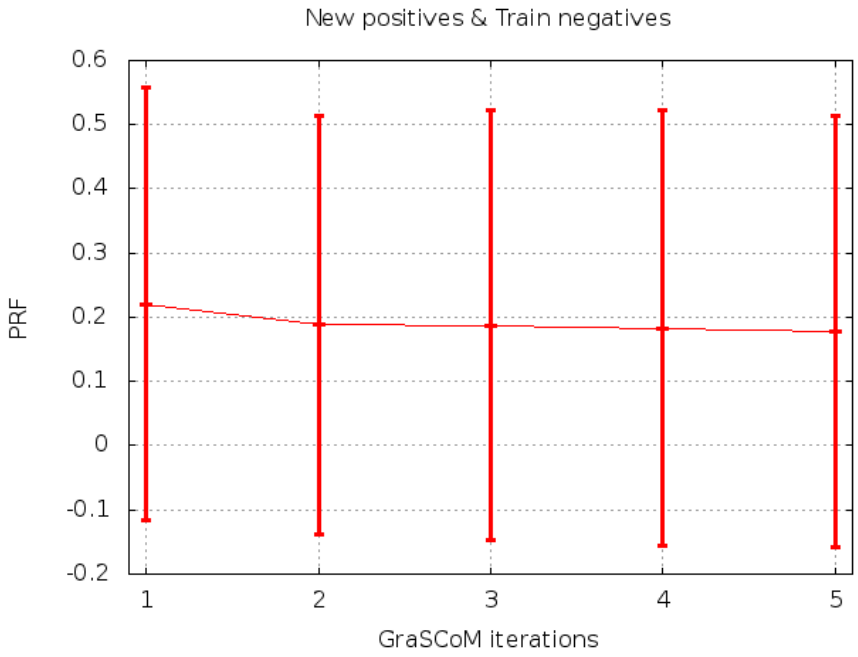

Figure 5.6: PRF value measured for the "Size of the input dataset" experiment

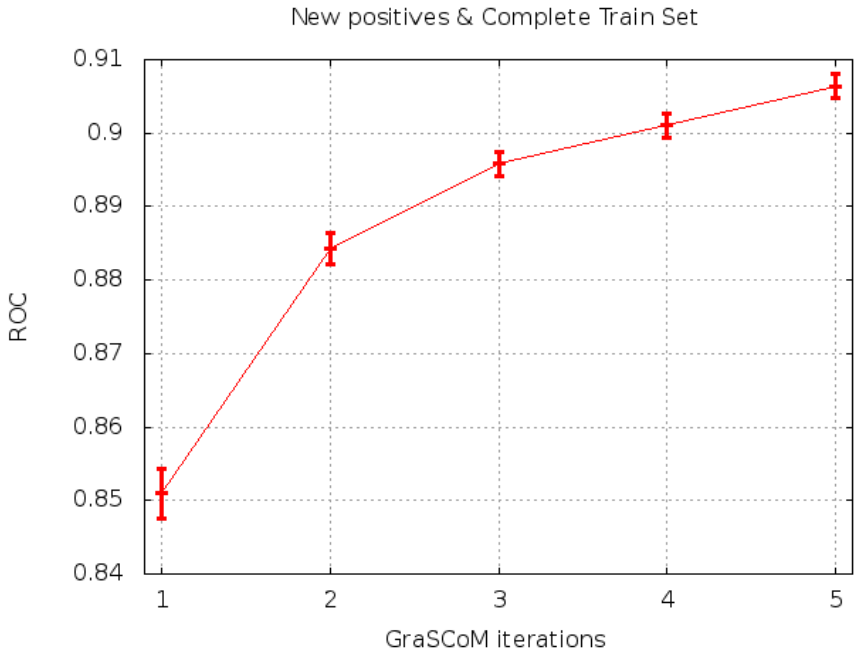

Figure 5.7: ROC value measured for the "Size of the input dataset" experiment

Figure 5.8: ROC value measured for the "Size of the input dataset" experiment

We can also analyze how does the number of extracted structures change under this linear growth of the input set. As one can notice from the the plots in Fig. 5.9-Fig. 5.10, a linear growth in the set used for inducing our graph construction model causes a similar linear growth in the size of cores and interfaces. This is a bit surprising, as one would expect to see a faster "saturation" w.r.t. the number of features that the system can extract given an increasing number of inputs.

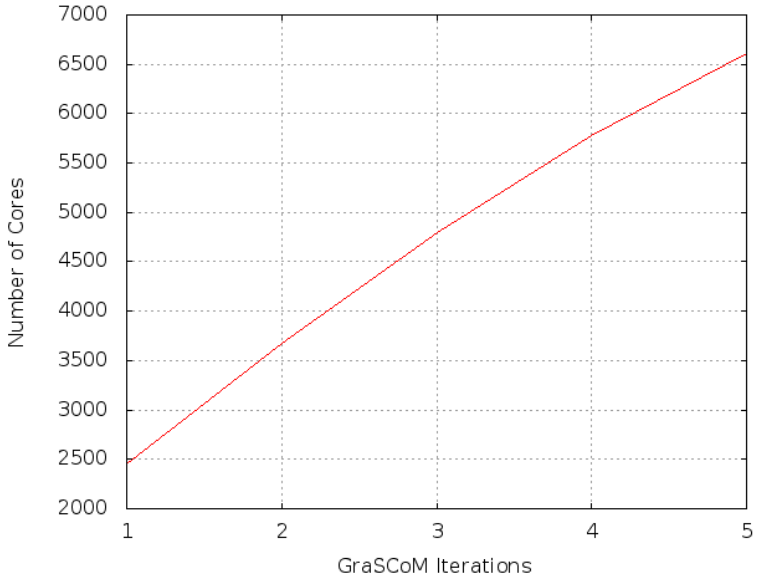

Figure 5.9: Linear growth of the input set causes linear growth in the number of cores

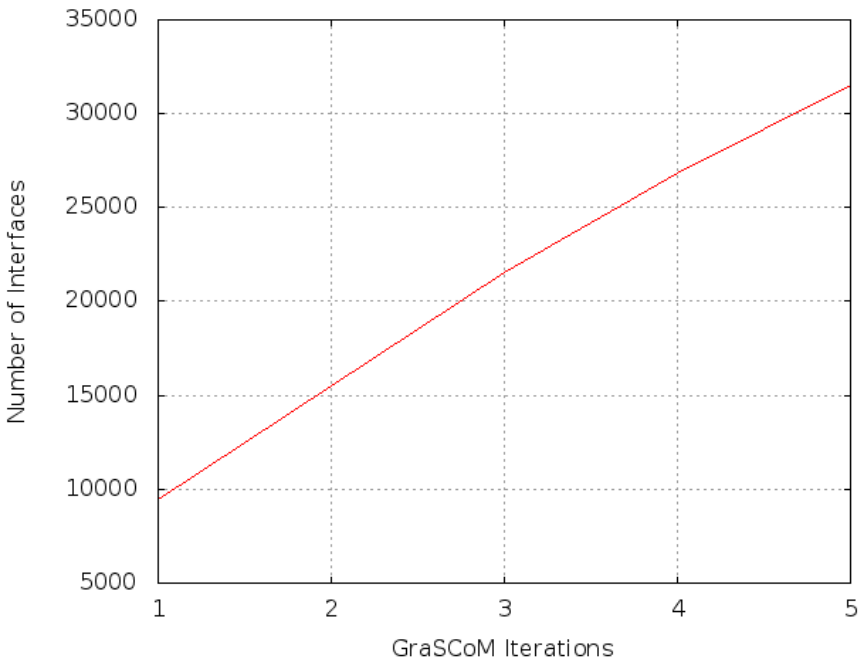

Figure 5.10: Linear growth of the input set causes linear growth in the number of interfaces

5.2.2 Probability of Finding Similar Molecules to the Test Set

The purpose of this experiment is to check how does the similarity between the newly generated molecules and the test set evolve over iterations and under different parameter constraints. For measuring this, we use the NSPDK to which we feed a list containing a merge between the 1000 top new compounds we generate after each iteration and the test set molecules (which lets assume are of size t). The NSPDK computes the similarity between all pairs of given molecules and returns a matrix of (1000+t)x(1000+t) containing scores between 0 and 1, where 1 is a perfect match. Obviously, the matrix will contain 1 on the diagonal.

What we have noticed in this experiment is that the more iterations we make, the less similar the newly generated molecules are to the test set. This is something to be expected, as the method tends to grow the size of the molecules. However, this can be controlled by the the *creativity factor* and in fact we actually managed to recreate compounds from the test set in some of our first iterations. The most impressive figures were always achieved under a *creativity factor* of 1 (i.e. $\langle Max_r = 5, Min_t = 4\rangle$ or $\langle Max_r = 3, Min_t = 2\rangle$). For these values we got up to 26 hits when comparing to the positives from the test set, which is a spectacular result given that the number of test positives was around 700.

The reason why these hits happened under these parameter values is because, as

explained in sec. 4.2 a thick interface with a radius that is as close as possible to it makes the system perform really small steps in the search space, be very conservative, and thus inspect the similar structures to the ones it gets as seeds. When the *creativity factor* has a value of one, it means that the system is performing only atom swapping, and depending on the thickness it expects more and more from the molecule in terms of structure similarity.

This is a very promising result which shows that the system is indeed capable to perform *de novo* molecular design and and the same time learn meaningful features from a dataset that contains molecules with similar (toxic) properties. Fig. 5.11 shows some examples of the valid positive molecules that the system generated.

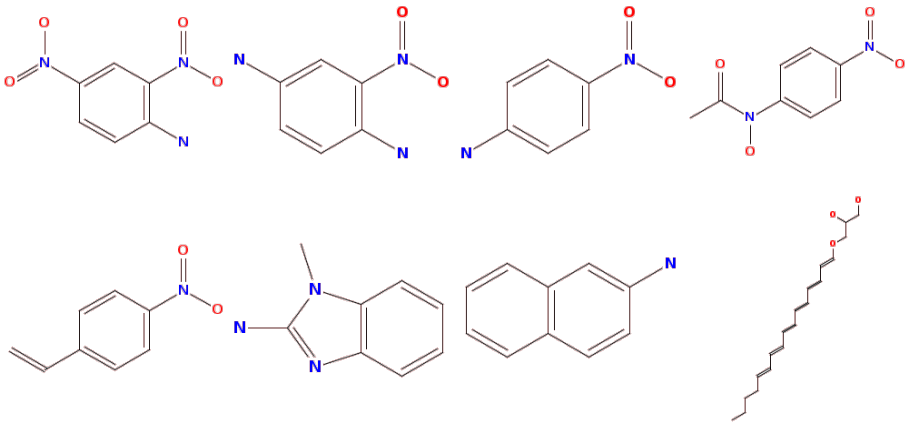

Figure 5.11: Valid de novo molecules created by *GraSCoM*

5.2.3 Radius Parameter

As we mentioned in the description of the method, the radius and the thickness are two parameters that influence a lot the way in which the search space is sampled. If we allow big radiuses, it means we allow substitutions of large cores and thus big changes with respect to the structure between the seed and the new generation. This causes the model to have actually worse performances as one can see in figures Fig. 5.12-Fig. 5.27 and this is because the learned model prefers molecules that are similar in structure to what it already knows.

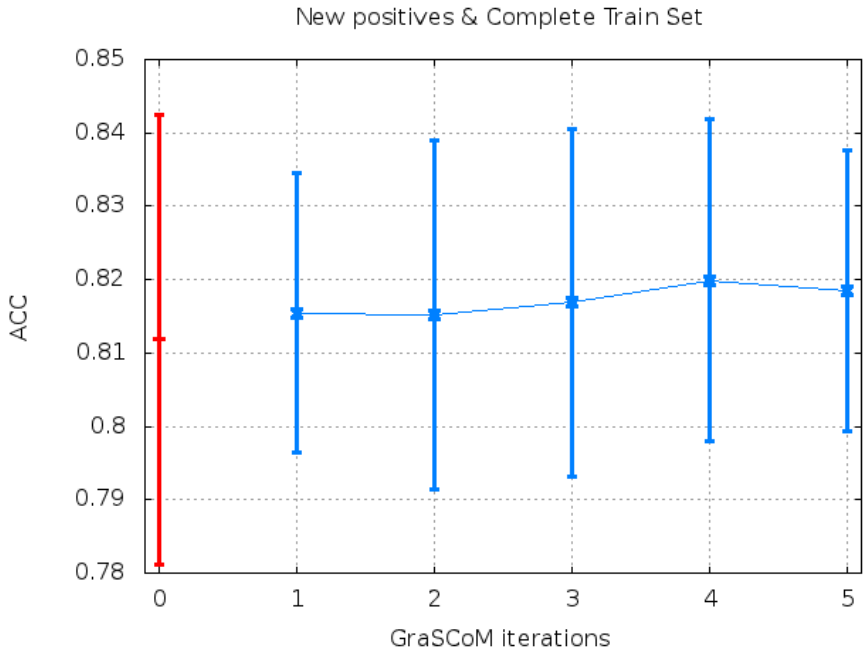

Figure 5.12: ACC value measured for 200 seeds, and radius 3 on *full train set merge*

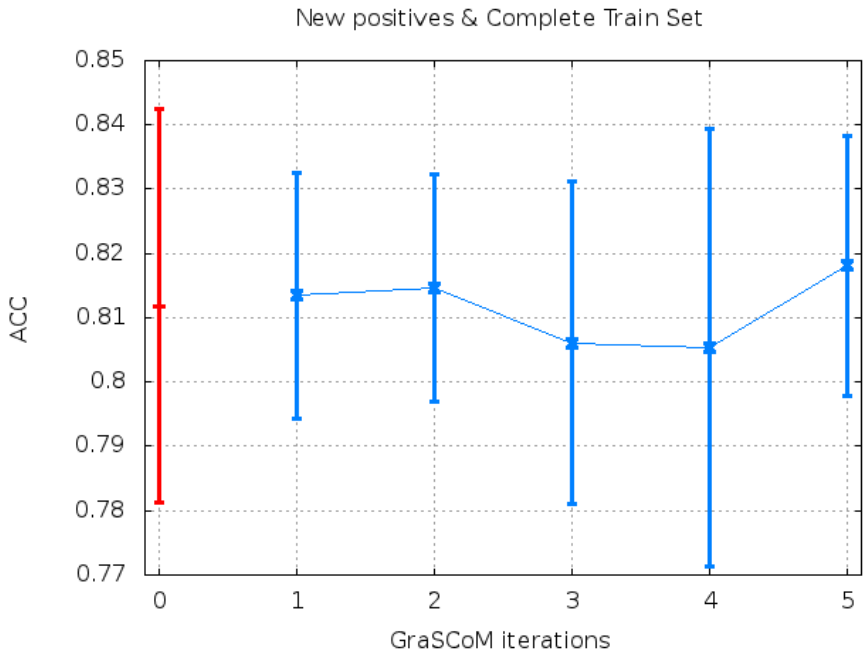

Figure 5.13: ACC value measured for 200 seeds, and radius 5 on *full train set merge*

Figure 5.14: ACC value measured for 200 seeds, and radius 3 on *negative train set merge*

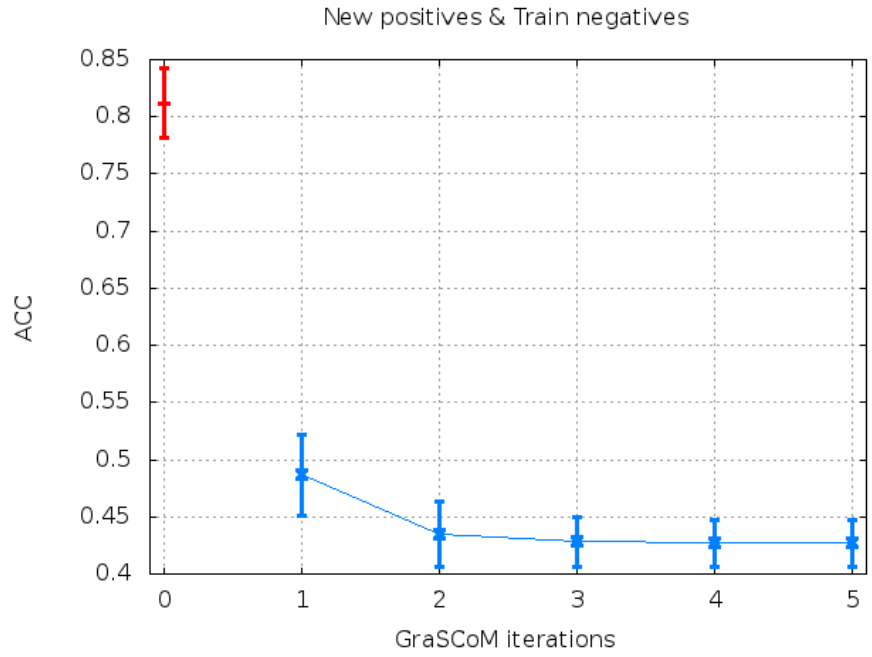

Figure 5.15: ACC value measured for 200 seeds, and radius 5 on *negative train set merge*

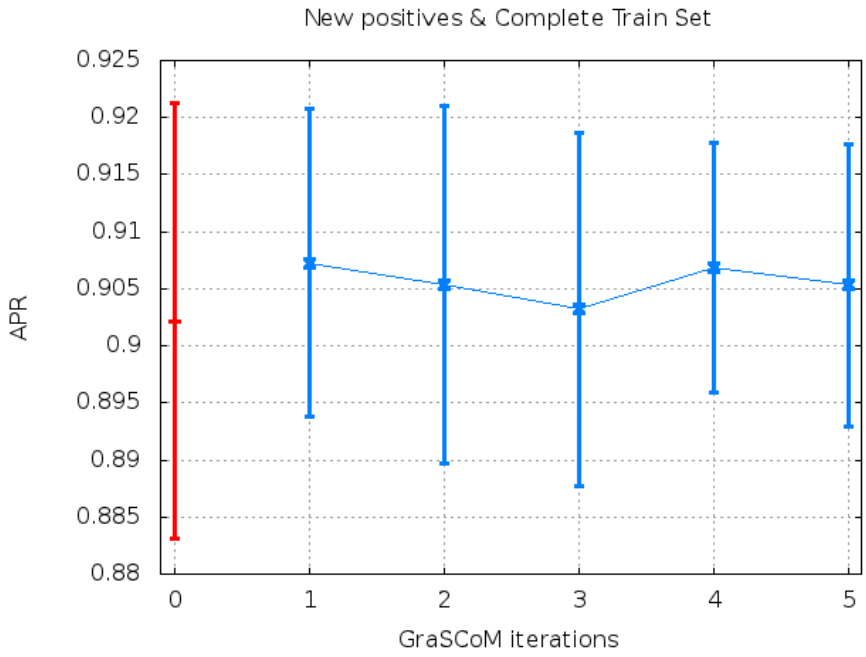

Figure 5.16: APR value measured for 200 seeds, and radius 3 on *full train set merge*

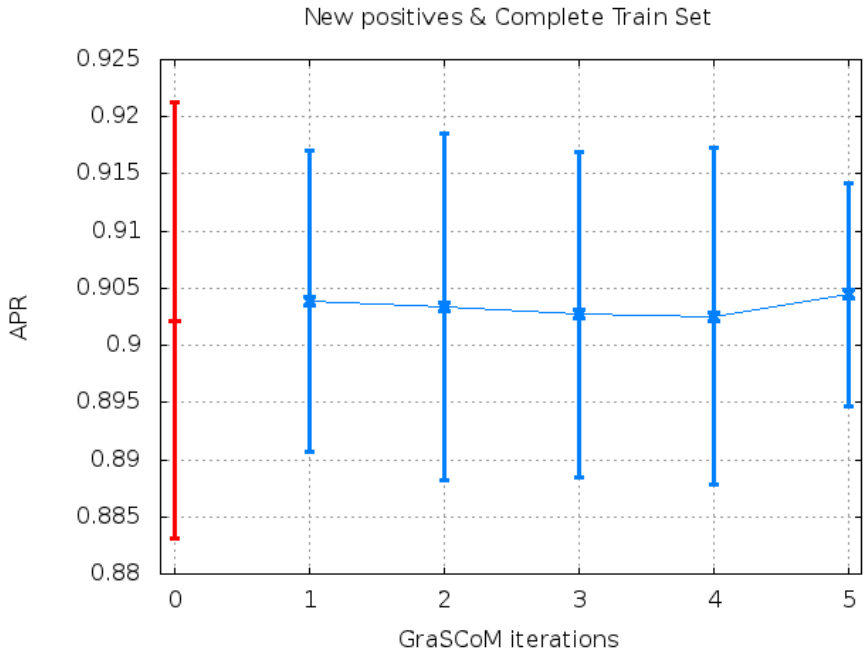

Figure 5.17: APR value measured for 200 seeds, and radius 5 on *full train set merge*

Figure 5.18: APR value measured for 200 seeds, and radius 3 on *negative train set merge*

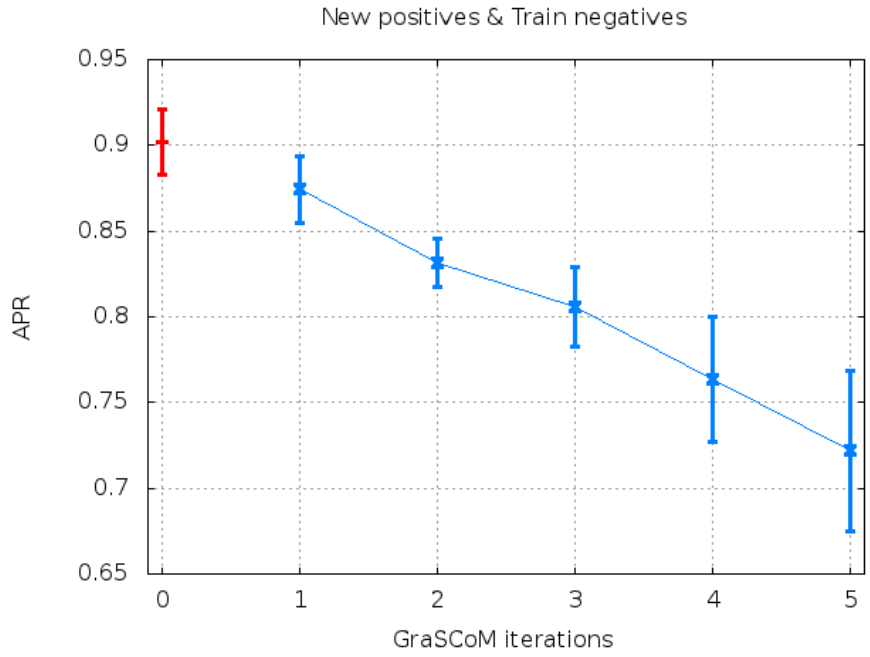

Figure 5.19: APR value measured for 200 seeds, and radius 5 on *negative train set merge*

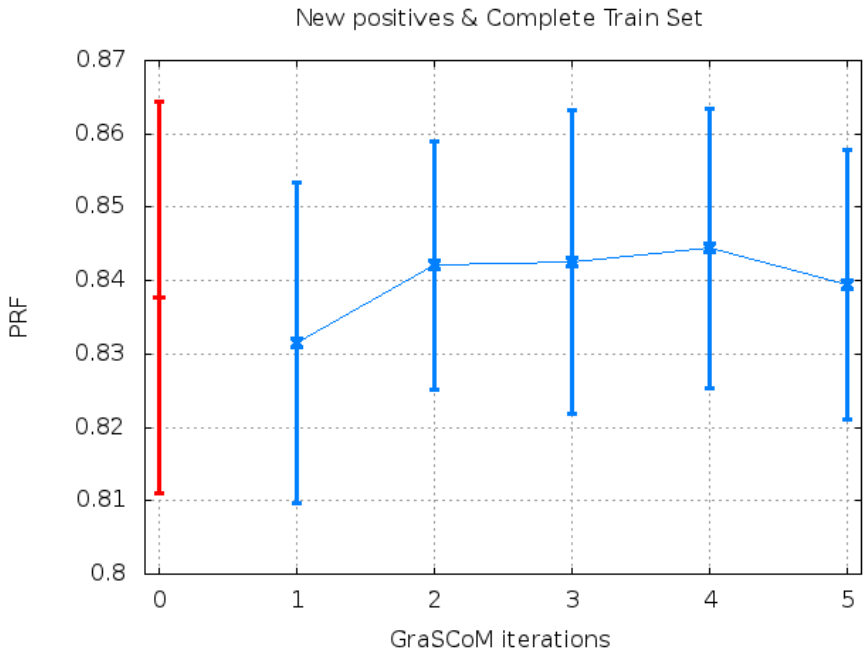

Figure 5.20: PRF value measured for 200 seeds, and radius 3 on *full train set merge*

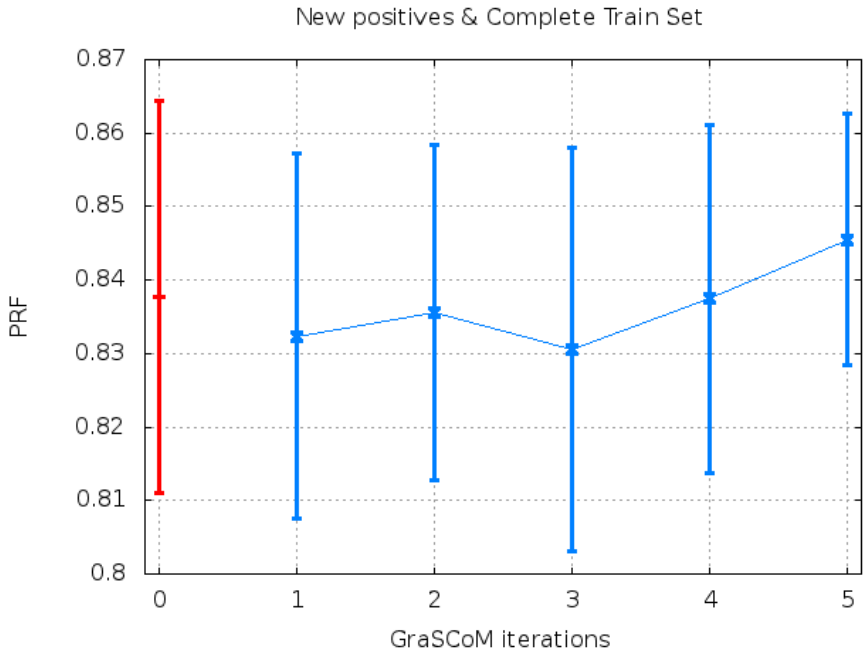

Figure 5.21: PRF value measured for 200 seeds, and radius 5 on *full train set merge*

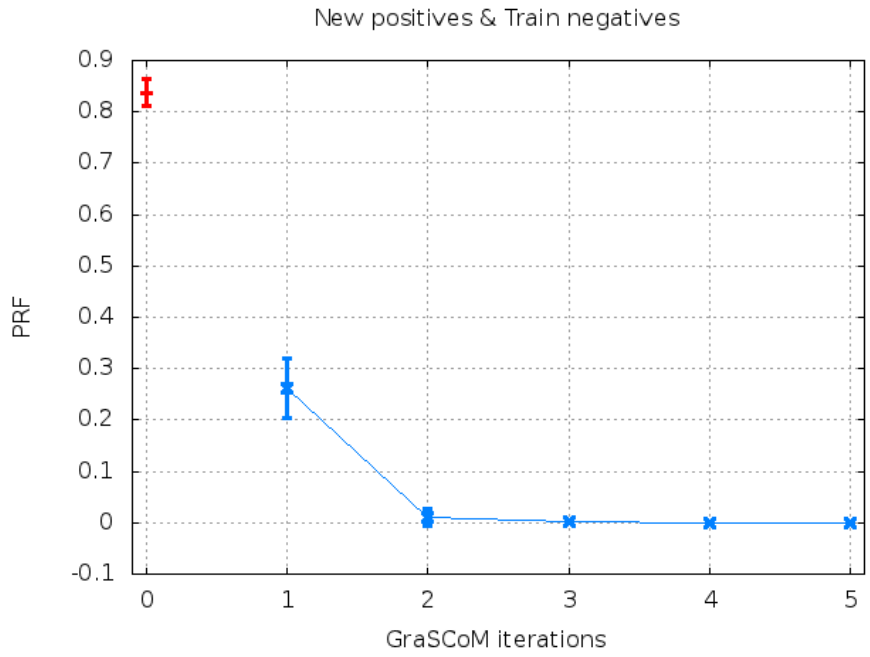

Figure 5.22: PRF value measured for 50 seeds, and radius 3 on *negative train set merge*

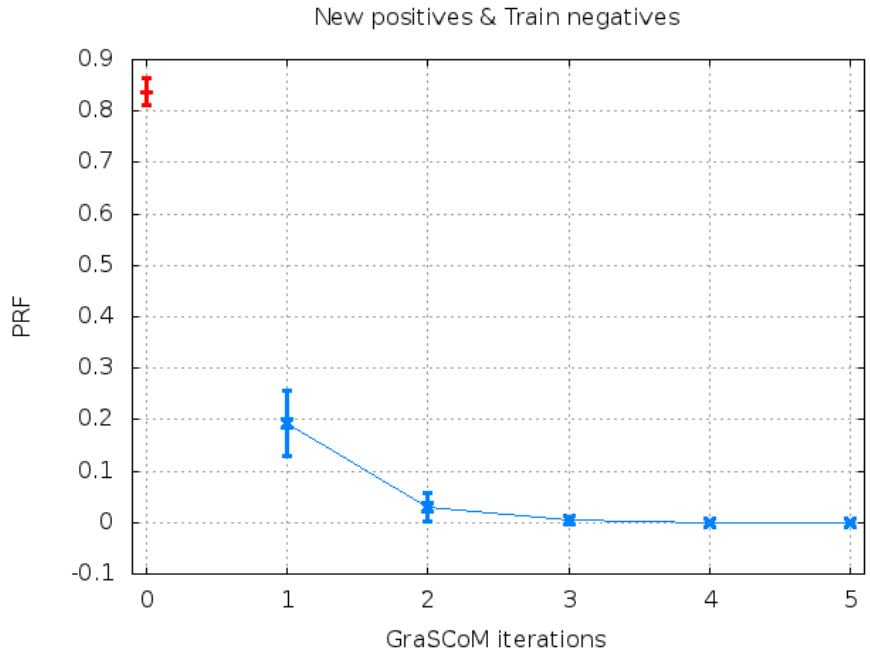

Figure 5.23: PRF value measured for 200 seeds, and radius 5 on *negative train set merge*

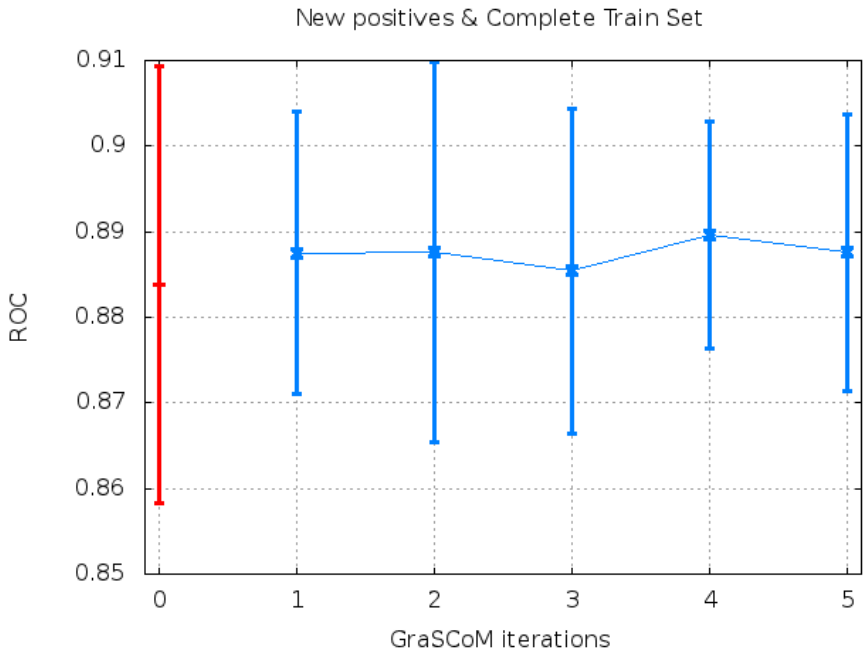

Figure 5.24: ROC value measured for 50 seeds, and radius 3 on *full train set merge*

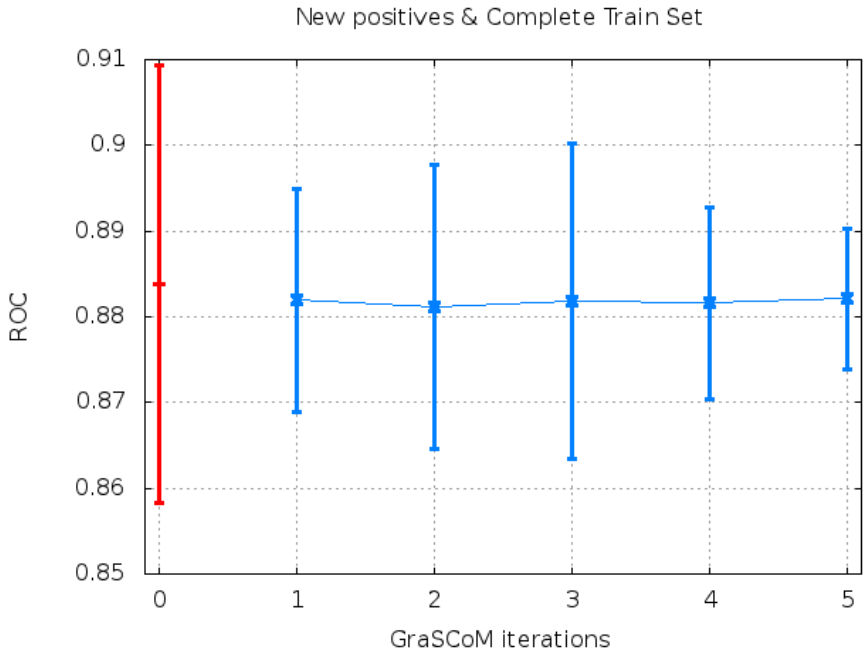

Figure 5.25: ROC value measured for 200 seeds, and radius 5 on *full train set merge*

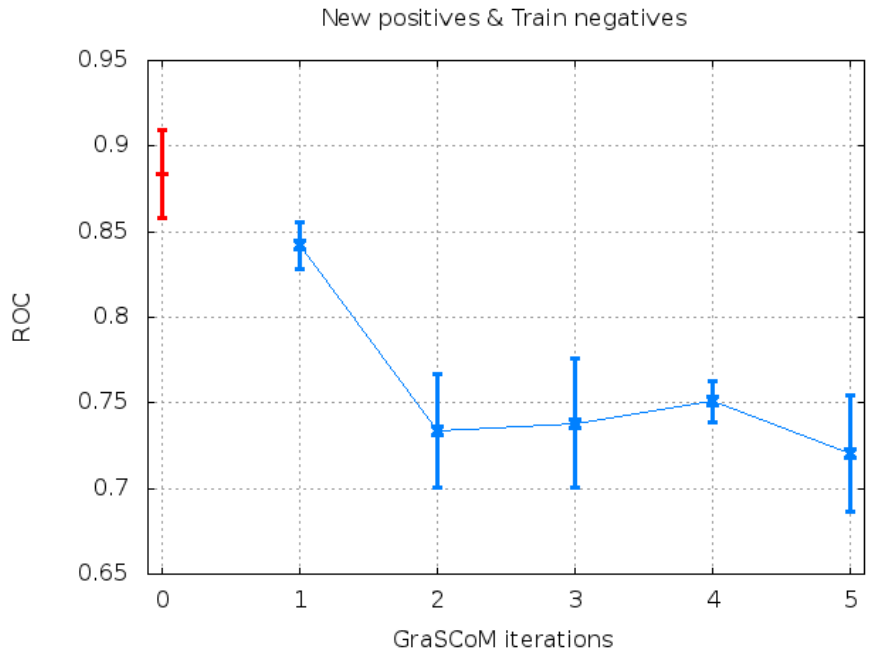

Figure 5.26: ROC value measured for 50 seeds, and radius 3 on *negative train set merge*

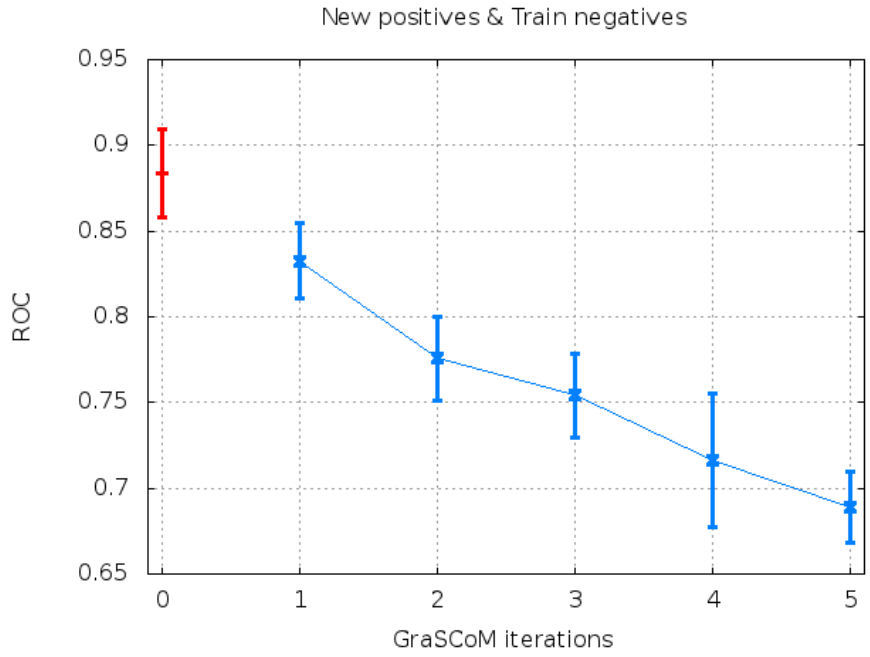

Figure 5.27: ROC value measured for 200 seeds, and radius 5 on *negative train set merge*

5.2.4 Number of Seeds Parameter

Here we analyzed how the system behaves under different sized seed sets through out complete runs. Interestingly, if intuitively one would expect a direct relationship between the seed set and the final performance, at least for the values we've considered (10,50,100,150) the difference in negligible. When one comes to think about it, it actually makes sense that there variation is so low, as the size of the database is the one that heavily influences the number of generated new molecules from each seed molecule.

In this sense, given a train set of around 1400 positives, the database can contain a number of over 7000 cores and more than 3200 interfaces. With these numbers the rewiring of every seed molecule generates from around 2500 new compounds to over 7000. One can imagine that the jumps performed in the search set are impressively big even if we take the lower bound of 2500 molecules.

Figure 5.28: ACC value measured for 50 seeds, and radius 5 on *full train set merge*

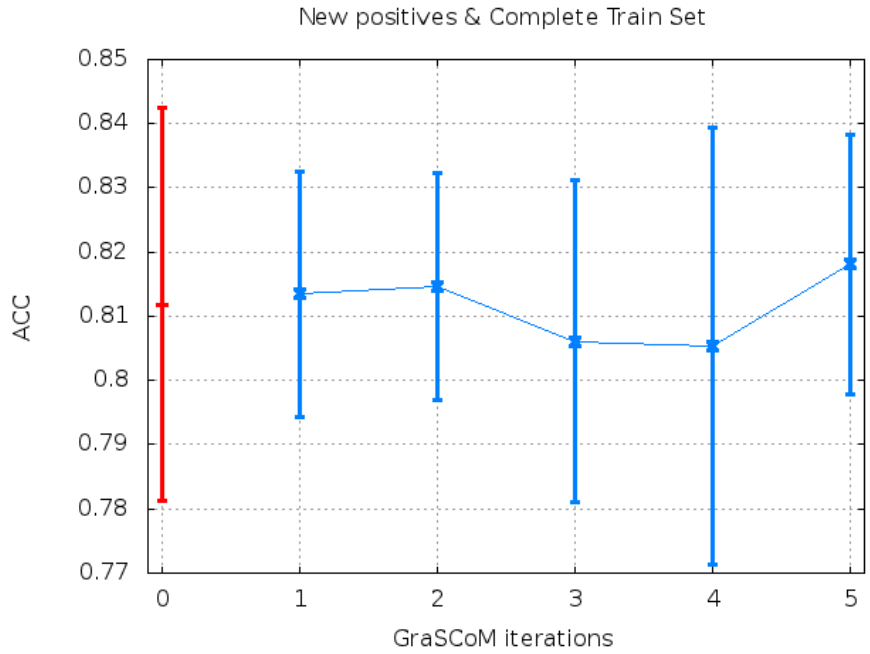

Figure 5.29: ACC value measured for 200 seeds, and radius 5 on *full train set merge*

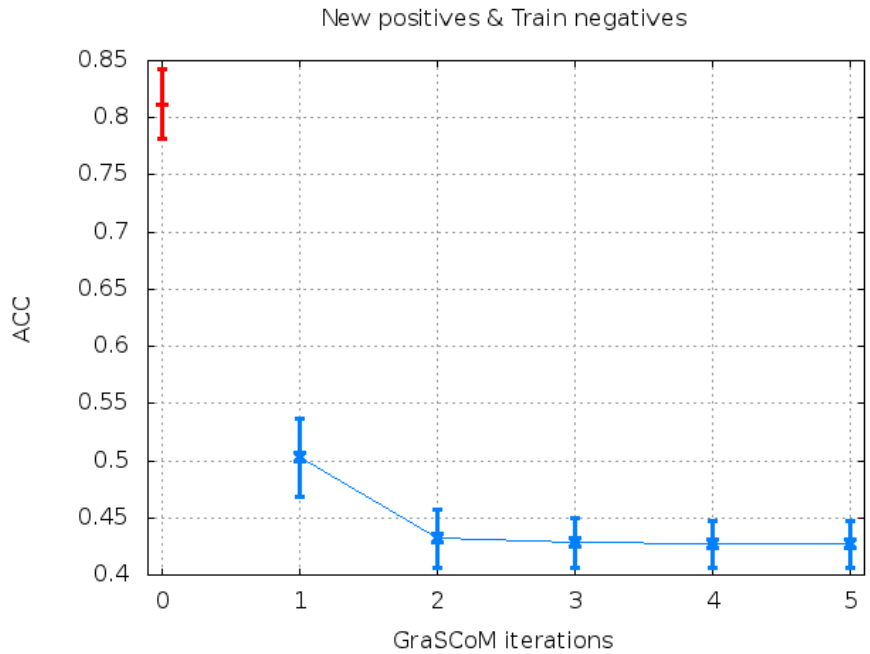

Figure 5.30: ACC value measured for 50 seeds, and radius 5 on *negative train set merge*

Figure 5.31: ACC value measured for 200 seeds, and radius 5 on *negative train set merge*

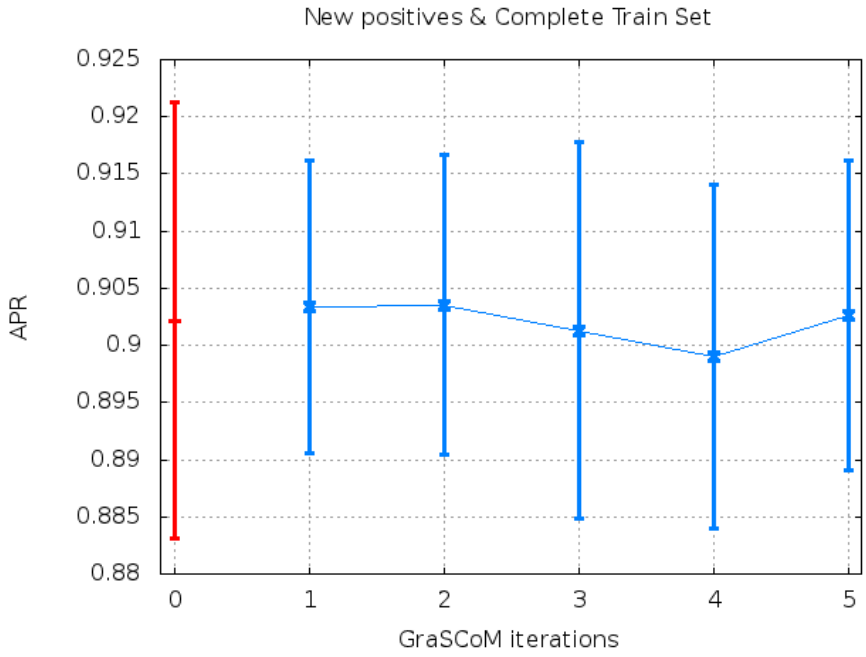

Figure 5.32: APR value measured for 50 seeds, and radius 5 on *full train set merge*

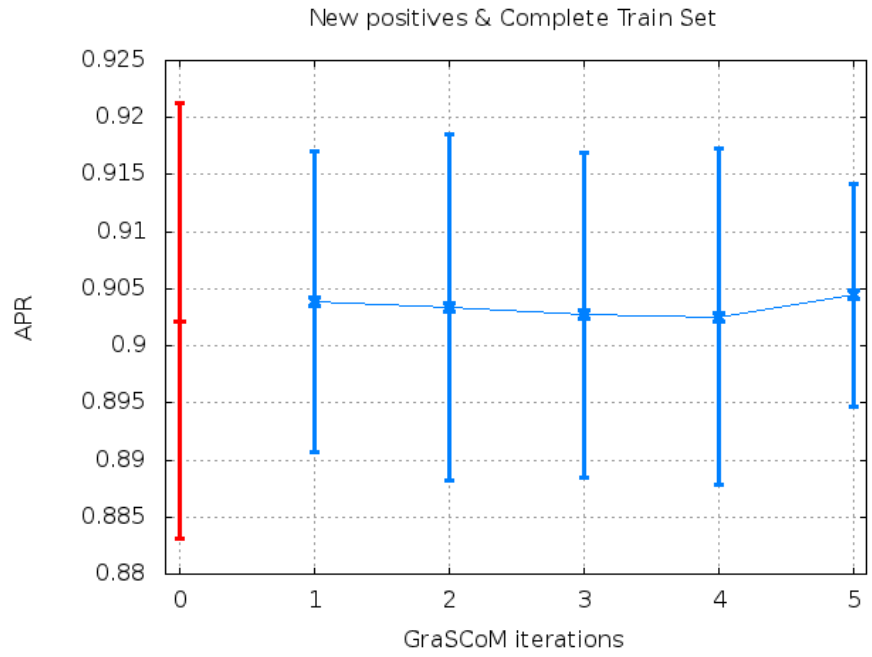

Figure 5.33: APR value measured for 200 seeds, and radius 5 on *full train set merge*

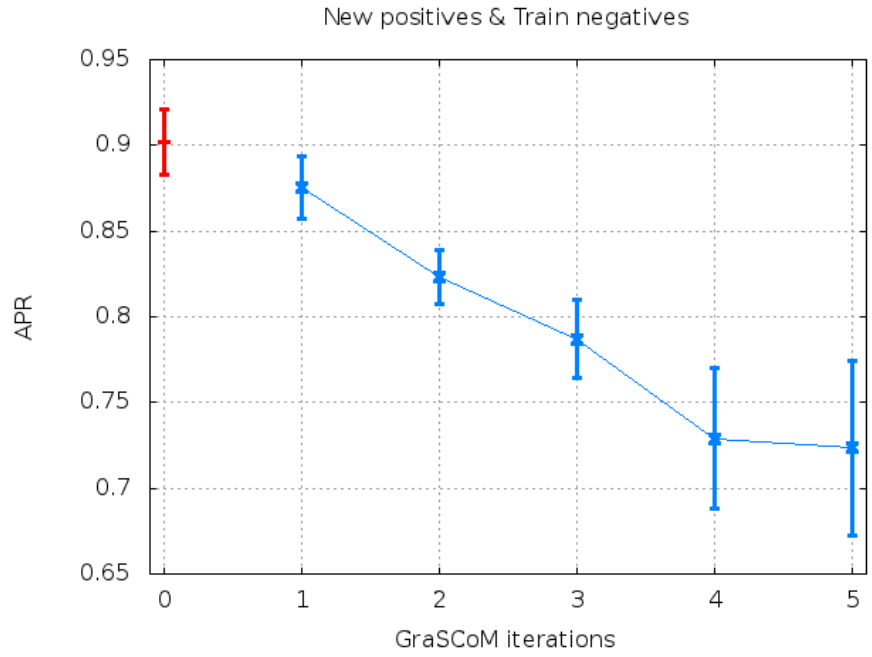

Figure 5.34: APR value measured for 50 seeds, and radius 5 on *negative train set merge*

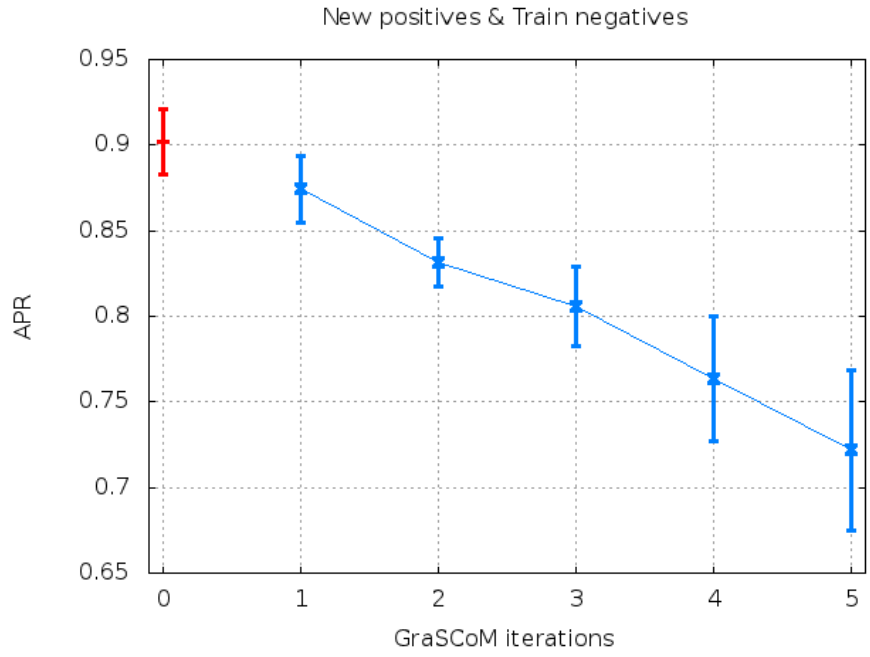

Figure 5.35: APR value measured for 200 seeds, and radius 5 on *negative train set merge*

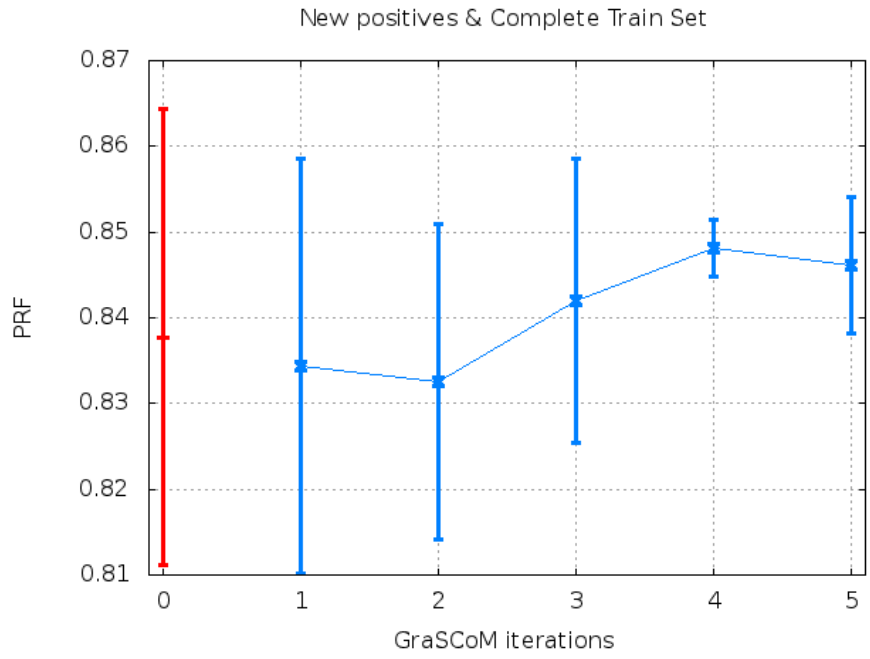

Figure 5.36: PRF value measured for 50 seeds, and radius 5 on *full train set merge*

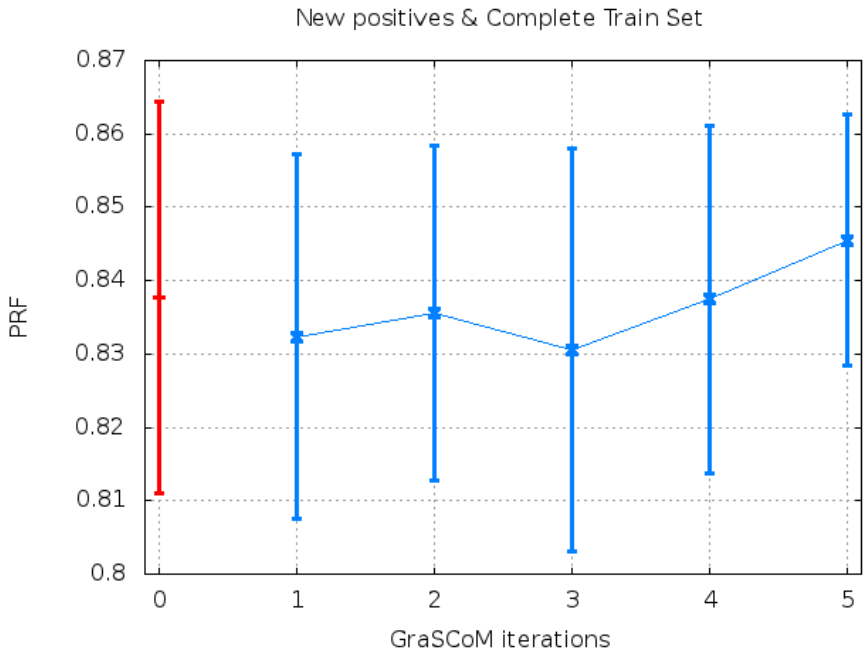

Figure 5.37: PRF value measured for 200 seeds, and radius 5 on *full train set merge*

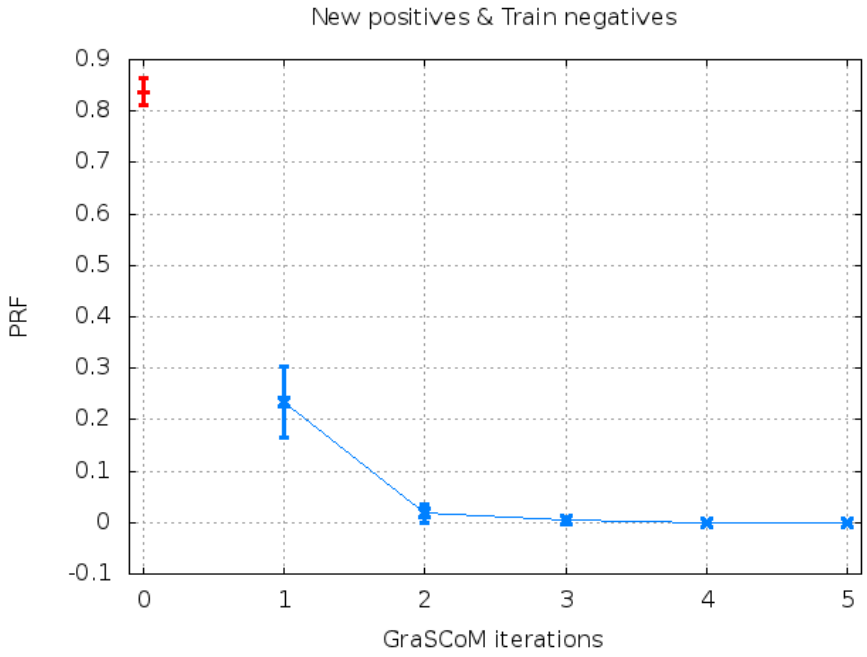

Figure 5.38: PRF value measured for 50 seeds, and radius 5 on *negative train set merge*

Figure 5.39: PRF value measured for 200 seeds, and radius 5 on *negative train set merge*

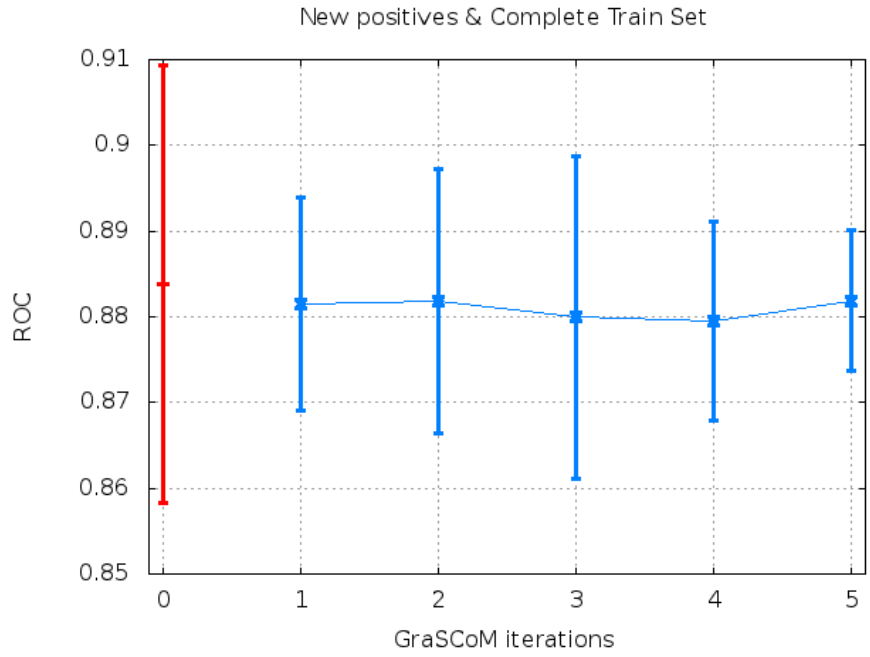

Figure 5.40: ROC value measured for 50 seeds, and radius 5 on *full train set merge*

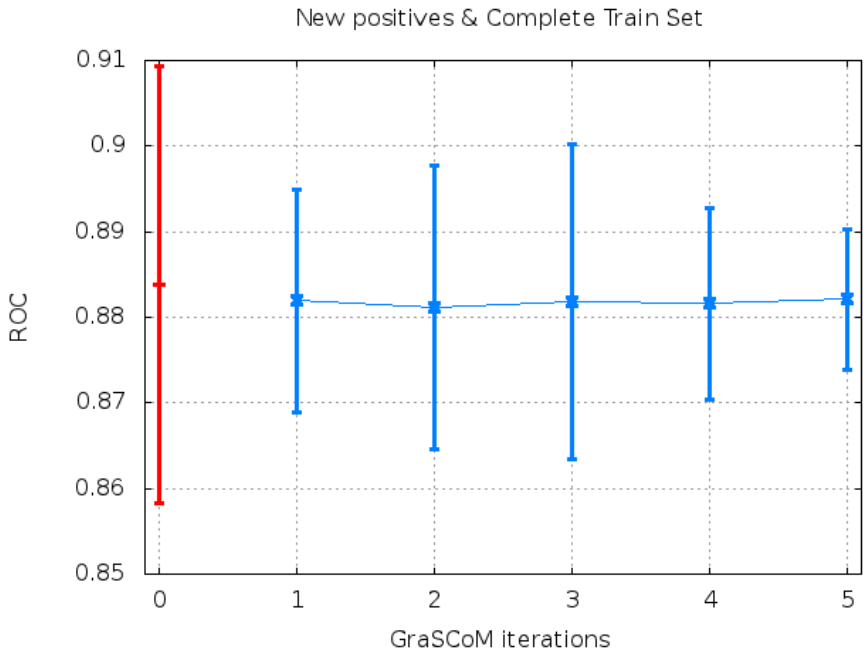

Figure 5.41: ROC value measured for 200 seeds, and radius 5 on *full train set merge*

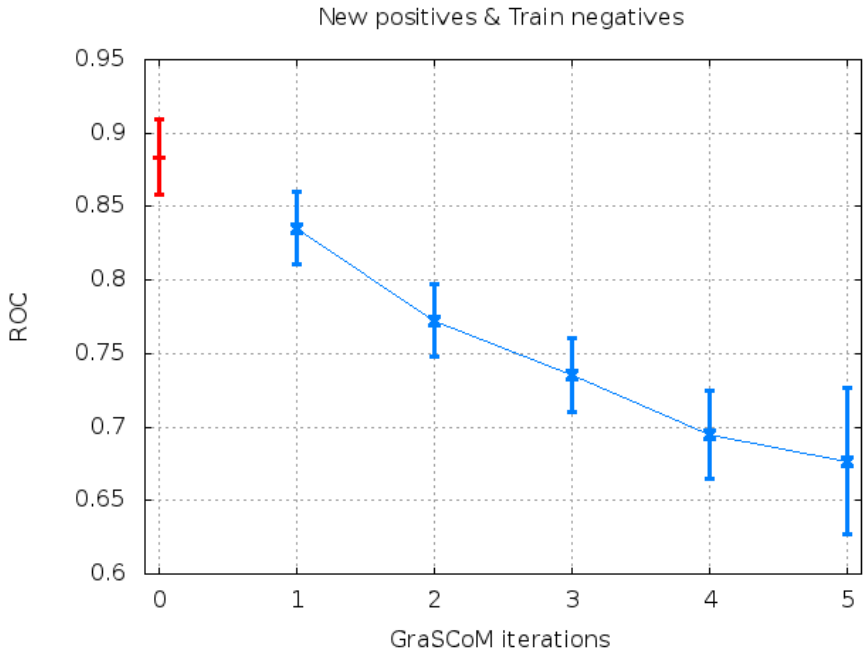

Figure 5.42: ROC value measured for 50 seeds, and radius 5 on *negative train set merge*

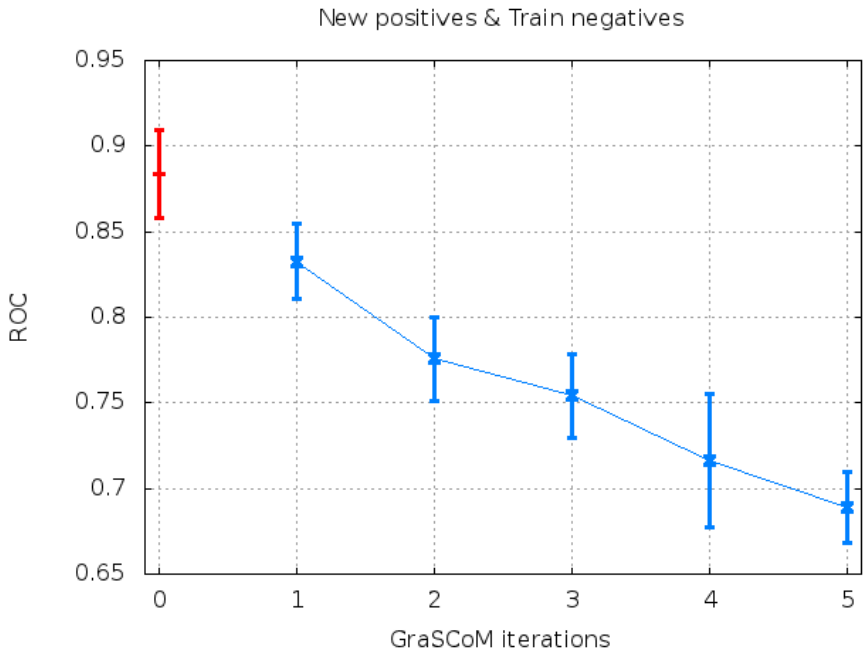

Figure 5.43: ROC value measured for 200 seeds, and radius 5 on *negative train set merge*

5.2.5 Iteration Parameter

These weren't necessarily a standalone set of experiments, but actually we can derive a general conclusion from all previous experiments regarding to how the system performs over all 5 iterations we perform.

First we should notice that the newly generated molecules are not as powerful as the set of positives, as they do not manage to induce a model which could classify data better (the *negative train set merge*). One reason of this could be that the system finds a structure that if believes to be really toxic and the propagates and prefers all molecules that contain it in terms of the scoring function (Fig. 5.45). However, as we mentioned in the chapter 2 section, there can't be immediate high expectations from a computer-based *de novo* design approach, as it could be that a scaffold hop was performed, and then the model that induces the scoring function is irrelevant. It could also happen that the proposed molecules are a good inspiration to the chemist and then again, the approach can be considered successful.

Nevertheless, the second type of experiment (the *full set train merge*) shows that when these compounds are added to an existing set of positive molecules, then the performance measure start to have a increasing tendency.

Even though looking at the standard deviations one could argue that this tendency

is not relevant, considering that we've learned the scoring functions on a relatively small dataset, the fact that for some iterations the performances are almost always above the reference values and that the experiment measuring the similarity to the database showed that there are indeed some hits with the test set, we believe that the approach has a great potential. We are confident that with further research, *GraSCoM* could become a really efficient and robust way of learning context-based features from graphs.

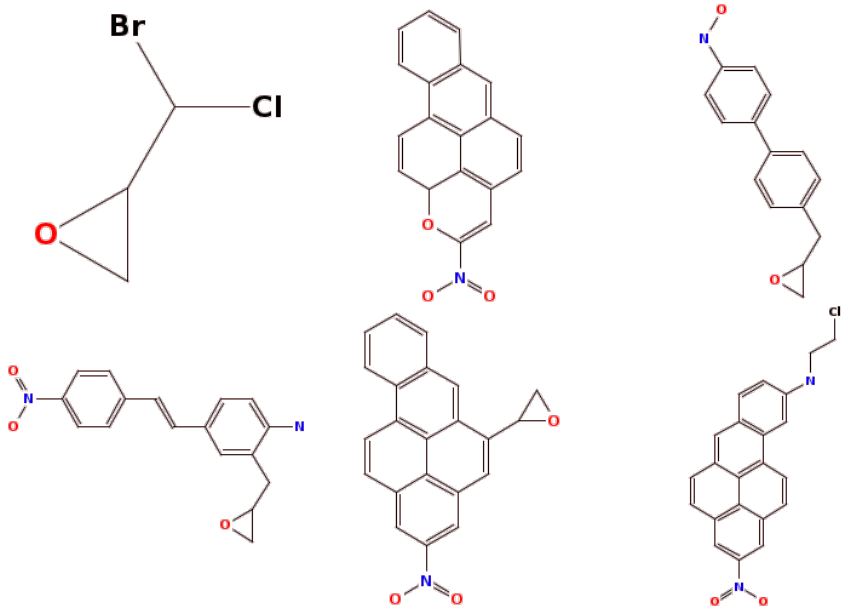

Figure 5.44: Molecules generated by *GraSCoM* after the first iteration

5.3 Complexity

When it comes to evaluating the complexity of this system, the most important factor that influences the run time and memory usage is the exponential growth between generations which makes an unrestricted run of *GraSCoM* unfeasible.

5.3.1 Exponential Growth

While of course the size of the input set is important w.r.t to the number of generated compounds, as we will see insec. 5.3.2 the number of identified structures grows linearly with the size of the dataset and thus has a linear influence to the final time of reassembling a graph. The two parameters that are heavily influencing this exponential growth are the radius and the thickness, and the difference between these values (what we previously introduced as the *creativity factor*).

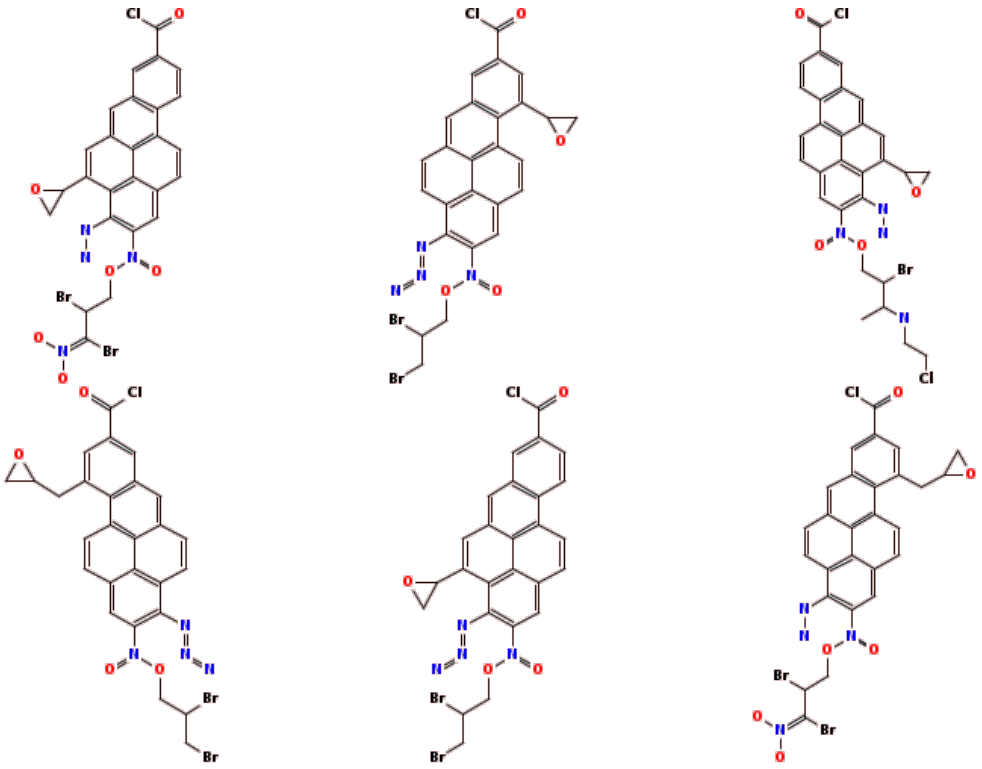

Figure 5.45: Molecules generated by *GraSCoM* after the fifth iteration

Tab. 5.1 shows a rounded average of the number of proposed candidates under different variations of the maximum/minimum radius and minimum thickness values.

Min_r	Max_r	Min_t	Average number of generated candidates
0	1	0	50
0	3	0	1500
0	3	2	90
2	3	0	750
0	5	0	7000
0	5	2	250
0	5	4	30
3	5	0	3200

Table 5.1: Influence of the creativity factor on the exponential growth

5.3.2 Run Time

In this subsection we will provide some run time figures that we have collected from different runs of the *GraSCoM* system on a machine with X5650 @2.67GHz processors.

Populating the database The time needed to identify the cores and interfaces, alongside with computing the hash ids for the interface vertices (for having them already computed when a *DualBFView* is performed) also shows a linear behavior with respect to the size of the input set (Fig. 5.46 and Tab. 5.2). This is an expected behavior: more graphs to analyze, more time needed.

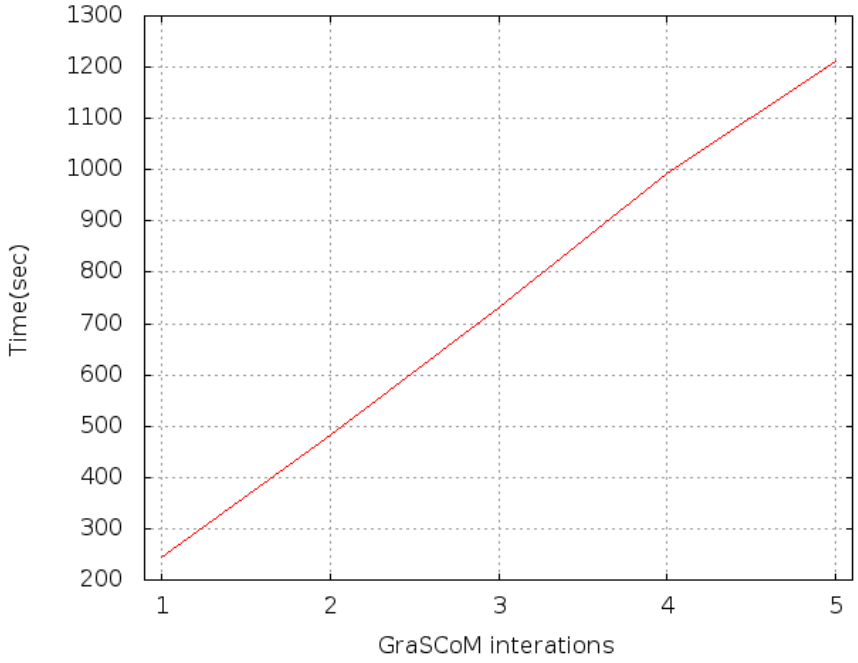

Figure 5.46: Linear growth of the input set causes linear growth on the time needed to populate the database.

However, it is important to mention that the method for populating the database can easily be parallelized thus one can just split the input set into small fractions and then just merge the results ensuring that identical structures are added only once to the final set.

Core substitution The time needed to perform all possible core substitutions given a graph, obviously depends on the size of the graph and the on the radius-thickness parameters, but also a big influence has the similarity of that graph with respect to the dataset used for identifying the structure. In this sense, if a molecule is similar

Size of input set	Time(sec)
348	246.29
692	482.81
1045	736.39
1405	988.62
1696	1196.07

Table 5.2: Time needed for identifying the cores and interfaces w.r.t the size of the input

to the dataset, then there will be a lot of common structures with the database. Through out the runs we have performed, we estimate an average of 10 seconds to rewire 100-200 graphs.

Scoring the molecules The time needed to score the graphs is generally very close to the the one that was needed to rewire them, the differences being in the order of seconds in the advantage of one or another.

6 Conclusion

6.1 Discusion

In this thesis we have introduced *GraSCoM*, a system that proposes a novel way of exploring the chemical/search space using a so-called *core substitution* procedure. The main goal of the thesis was to see how meaningful such a exploration would be and given the results we can say without doubt that it is indeed a very promising and powerful technique.

An interesting fact about the implementation of *GraSCoM* is that internally it deals with graphs (or hypergraphs) and only the scoring function is the one that makes it a molecular design approach. This gives room to a lot of ideas of what such a tool could do. In other words, given an entity that could be represented as a graph and a scoring function for the type of problem one is interested in, the system is capable of extracting meaningful structures from an input set and then propose new (viable) solutions for improving.

6.2 Future Work

As one can imagine, there are a lot of ideas such a setup generates and also a lot of things that one could think of adding.

One useful addition would be allowing another input parameter where one could be specify as a string a policy about how the parameters should change every iteration. For example, one could give the system a big creativity factor for the first iteration (thus allowing big jumps in the search space) and then make the system conservative for the next couple of iterations (similar to the simulated annealing strategy).

Another idea would be to allow a finer way of filtering the top molecules by using domain-specific constraints. In this sense one could prefer graphs with some desired property and discard others that would be of no interest for that particular study.

Bibliography

[1] B. D. Anson, J. Ma, and J.-Q. He. Identifying cardiotoxic compounds. *Genetic Engineering & Biotechnology News*, 9(3):34 – 35, May 2009.

[2] F. R. Bach and G. R. G. Lanckriet. Multiple kernel learning, conic duality, and the smo algorithm. In *In Proceedings of the 21st International Conference on Machine Learning (ICML)*, 2004.

[3] H. J. Boehm. The computer program ludi: a new method for the de novo design of enzyme inhibitors. *Journal of computeraided molecular design*, 6(1):61–78, 1992.

[4] R. Bohacek, C. McMartin, and W. C. Guida. The art and practice of structure-based drug design: A molecular modeling perspective. *Medicinal Research Reviews*, 16(1):3 – 50, 1996.

[5] L. Bottou. Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier and G. Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, 2010. Springer.

[6] A. Clark and R. Eyraud. Polynomial identification in the limit of substitutable context-free languages. *J. Mach. Learn. Res.*, 8:1725–1745, 2007.

[7] F. Costa and K. D. Grave. Fast neighborhood subgraph pairwise distance kernel. *International Conference on Machine Learning*, pages 255 – 262, 2010.

[8] D. S. Dummit and R. M. Foote. *Abstract Algebra*. John Wiley & Sons, Inc., 3rd edition, 2004.

[9] V. Gillet, P. A. Johnson, P. Mata, S. Sike, and P. Williams. Sprout: A program for structure generation. *Journal of Computer-Aided Molecular Design*, 7:127 – 153, 1993.

[10] J. L. Gross and J. Yellen. *Handbook of Graph Theory (Discrete Mathematics and Its Application)*. CRC Press, 1st edition, 2003.

[11] M. Hartenfeller and G. Schneider. Enabling future drug discovery by de novo design. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(5):742 – 759, 2011.

[12] D. Haussler. Convolution kernels on discrete structures. Technical report, Computer Science Department, UC Santa Cruz, 1999.

[13] D. Horvath and C. Jeandenans. Neighborhood behavior of in silico structural spaces with respect to in vitro activity spacesâa novel understanding of the molecular similarity principle in the context of multiple receptor binding profiles. *Journal of Chemical Information and Computer Sciences*, 43(2):680 – 690, 2003.

[14] T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 158–167, 2004.

[15] H. Kashima, K. Tsuda, and A. Inokuchi. Kernels for graphs. *Kernels and Bioinformatics*, pages 155–170, 2004.

[16] J. Kazius, R. McGuire, and R. Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of Medicinal Chemistry*, 48(1):312–320, 2005.

[17] G. Klopmand. *Concepts and applications of molecular similarity.* John Wiley & Sons, Inc., 1992.

[18] R. A. Lewis, D. C. Roe, C. Huang, T. E. Ferrin, R. Langridge, and I. D. Kuntz. Automated site-directed drug design: the formation of molecular templates in primary structure generation. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 236(1283):141 – 162, March 1989.

[19] Z. Luo, R. Wang, and L. Lai. Rasse:a new method for structure-based drug design. *Journal of Chemical Information and Computer Sciences*, 36(6):1187 – 1194, 1996.

[20] Y. Nishibata and A. Itai. Automatic creation of drug candidate structures based on receptor structure. starting point for artificial lead generation. *Tetrahedron*, 47(43):8985 – 8990, 1991.

[21] S. M. Paul, D. S. Mytelka, C. T. Dunwiddie, C. C. Persinger, B. H. Munos, S. R. Lindborg, and A. L. Schacht. How to improve r&d productivity: the pharmaceutical industry's grand challenge. *Nature Reviews Drug Discovery*, 9(3):203 – 214, March 2010.

[22] D. A. Pearlman and M. A. Murcko. Concepts: New dynamic algorithm for de novo drug suggestion. *Journal of Computational Chemistry*, 14(10):1184 – 1193, 1993.

[23] D. A. Pearlman and M. A. Murcko. Concerts:dynamic connection of fragments as an approach to de novo ligand design. *Journal of Medicinal Chemistry*, 39(8):1651 – 1663, 1996.

[24] A. C. Pierce, G. Rao, and G. W. Bemis. Breed:â generating novel inhibitors through hybridization of known ligands. application to cdk2, p38, and hiv protease. *Journal of Medicinal Chemistry*, 47(11):2768–2775, 2004.

[25] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093 – 1110, 2005.

[26] J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. In *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences*, pages 65–74, 2003.

[27] I. Rechenberg. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, University of Berlin.

[28] G. Schneider and U. Fechner. Computer-based de novo design of drug-like molecules. *Nature Reviews Drug Discovery*, 4:649 – 663, August 2005.

[29] G. Schneider, M.-L. Lee, M. Stahl, and P. Schneider. De novo design of molecular architectures by evolutionary assembly of drug-derived building blocks. *Journal of Computer-Aided Molecular Design*, 14:487–494, 2000.

[30] H. M. Vinkers, de Marc R. Jonge, F. F. D. Daeyaert, J. Heeres, L. M. H. Koymans, J. H. van Lenthe, P. J. Lewi, H. Timmerman, K. V. Aken, and P. A. J. Janssen. Synopsis:â synthesize and optimize system in silico. *Journal of Medicinal Chemistry*, 46(13):2765–2773, 2003.

[31] L. J. William, J. Ruiz-Caro, J. Tirado-Rives, A. Basavapathruni, K. S. Anderson, and A. D. Hamilton. Computer-aided design of non-nucleoside inhibitors of hiv-1 reverse transcriptase. *Bioorganic & Medicinal Chemistry Letters*, 16(3):663 – 667, 2006.