Software project: MIGRAINE - MultIple GRaph AlIgNmEnt tool via Bron Kerbosch and VF2 algorithms

Fortgeschrittene Methoden der Bioinformatik (10-INF-BIO4)

Anna Grunwald, Lena Sünke Mortensen, Christiane Gärtner
University of Leipzig

November 18, 2019

Abstract

In bioinformatics alignments are usually constructed from strings or other totally ordered inputs. A step from these alignments to alignments of general structures could be the multiple alignment of graphs. These alignments can play an important role in chemoinformatics in order to compare molecules and to represent general chemical reaction types. In this tool we implemented a progressive multiple graph alignment using either the Bron Kerbosch (bk) algorithm for finding maximal cliques in a graph or the Cordella (sub)graph isomorphism algorithm for matching large graphs.

1 Implemented Algorithms

In the following the most important ideas, functions and algorithms are described:

- **Principles of multiple graph alignment**

  The basic concept for pairwise graph alignment can be described as follows: The input is represented by two graphs, each of them forming a partial order set. The aim is to find a common induced subgraph between these graphs. Nodes and edges of the two graphs, which are not found in the maximal common induced subgraph, are constructed in incomparable pairs to the alignment. This again results in a partial order set, which can be further aligned, so multiple graph alignment in a progressive way is possible [1]. Finding the common induced subgraph between two graphs is performed via clique finding using the Bron Kerbosch (bk) algorithm or using the subgraph isomorphism search via the Cordella (sub)graph isomorphism algorithm for matching large graphs (VF2).

  The most widely used approach to multiple sequence alignments uses a heuristic known as progressive technique [2]. This procedure is applied here to graph alignment. First, pairwise alignments were calculated using either the bk or VF2 algorithm. Therefore either the largest common subgraph or highest scoring common subgraph between two graphs is calculated. Second, a distance matrix is created, i.e. graphs with large or high scoring common subgraph have similar values, strongly different graphs have a large distance there. This distance matrix is then turned into a guide tree using the UPGMA (unweighted pair group method with arithmetic mean) algorithm. The UPGMA algorithm constructs a rooted tree that reflects the structure present in a pairwise distance matrix, thereby at each step, the nearest two clusters are combined into a higher-level cluster [3]. Then the progressive alignment is calculated by adding the graphs sequentially to the growing multipe graph alignment (MGA) according to the guide tree. In our tool these algorithms are implemented in the classes Matching, DirGraphAlign, and Guide_tree.

- **Random graph generation**
  The built in tool graphgen provides a generator for random graphs with various possibilities for node count, edge probability, random labeling of nodes and edges, as well as the possibility to generate directed and indirected graphs. It also provides options to generate "mesh graphs". The graphs are saved in GRAPH format, that can be used for input in our MGA tool.

- **Parser for different graph file formats, Input and Output**
  The class GraphIO provides possibilities to read and write different graph formats: GRAPH and GRAPHML. In addition it offers the possibility to parse molecules from the PubChem database in JSON file format. The input is converted into a networkX Graph or DiGraph object. While parsing, nodes and edges are optionally labeled. This enables the scoring of certain label combinations or the incorporation of forbidden label combinations in the matching algorithms. Output of matches is provided as TXT file, that lists matching nodes and edges and in GRAPHML file format. Additionally, the guide tree can be outputted via NEWICK format or visualized as a PNG file. Also the final matching object could be saved as a PNG file.

- **Finding all maximal cliques of a graph using the Bron Kerbosch algorithm**
  The bk algorithm is developed for finding all maximal cliques of an undirected graph. As a result it lists all subsets of vertices with the two properties that each pair of vertices in one of the listed subsets is connected by an edge, and no listed subset can have any additional vertices added to it while preserving its complete connectivity [4]. In this project it is used to find either the largest common or highest scoring common subgraph of two graphs. Therefore the modular graph product of two graphs is created and in this the maximal cliques are calculated. Then either the largest clique or the highest scoring clique is selected for optimal alignment of the two graphs. The runtime complexity of the bk algorithm is $O(3^{n/3})$. For runtime optimisation a pivoting strategy is used in this implementation, because the original algorithm is inefficient in the case of graphs with many non-maximal cliques [5]. On this occasion a vertex is chosen from the set of candidate vertices P. Any maximal clique must include either the selected vertex or one of its non-neighbors, for otherwise the clique could be augmented by adding this vertex to it. Therefore, only this vertex and its non-neighbors need to be tested as the choices for the vertex that is added to the set R containing the nodes of the known complete subgraph. This strategy effectively limitates the number of recursive calls of the algorithm. The bk algorithm is implemented in the class Matching.

- **Finding a common induced subgraph using the VF2 algorithm**
  The VF2 algorithm was originally developed for graph-subgraph isomorphism finding [6], and offered a significant improvement in calculating time compared to previous solutions. We modified it to yield maximal common induced subgraphs (MCIS), but the underlying logic remains basically identical. Common induced subgraphs (CIS) are represented as “states” (s), and extended by adding a candidate node pair to the current CIS. The candidate pairs are calculated from the current state, thus the VF2 algorithm employs a depth-first search strategy. For each state s, the terminal sets $T^{out}(s)$ and $T^{in}(s)$ are calculated for G1 and G2 each. T^{out} and T^{in} represent the nodes which are not part of the current CIS, but are successors and predecessors, respectively, for the nodes in the CIS. The candidate set P(s) is calculated depending on the contents of the terminal sets. If all sets are empty,

$$P(s) = (N_1 - M_1(s)) \times func(N_2 - M_2(s)),$$

  where func represents a total ordering criterion. Cordella et al [6] as well as other’s [7]

and our own implementation use the node in G2 with the smallest index, however a recent paper [8] suggests a strategy to calculate an optimal node order. If $T_1^{out}(s)$ and $T_2^{out}(s)$ are not empty, then

$$P(s) = T_1^{out}(s) \times func(T_2^{out}(s)).$$

If $T_1^{out}(s)$ and $T_2^{out}(s)$ are empty, but $T_1^{in}(s)$ and $T_2^{in}(s)$ are not, then

$$P(s) = T_1^{in}(s) \times func(T_2^{in}(s)).$$

If only one of the in- or out-terminal sets is empty, the CIS in the current state cannot be further expanded. After a candidate pair is added to the CIS(s), a feasibility function F tests if the newly generated subgraph is an induced subgraph. If so, a new state s, representing the partial mapping solution is generated. Briefly, F inspects all neighboring nodes of the newly added nodes n and m. If a neighbor of n is part of the mapping, its corresponding node must be a neighbor of m and vice versa. Optionally, semantic information such as node or edge labels can be evaluated. When no further candidate nodes are found, and the CIS is longer or of equal length than previously found mappings, it is returned as a possible alignment solution. From the resulting list of possible alignment solutions, the maximal CIS is selected (randomly in case of >1 CIS of equal length) and used to construct an alignment graph. In case one or both input graphs are disconnected, the components are matched subsequently. For this, the components of each graph are sorted by length, and the largest components are compared. Next, the nodes which were already matched are removed from the graphs and the new components are sorted and matched again. This function is only implemented for undirected graphs. The modified VF2 algorithm is implemented in the VF2_GraphAligner class with its subclasses DirGraphAlign and MappingState. Its runtime complexity is $O(N^2)$ in the best case, and $O(N!N)$ in the worst case [9].

- **Integration of forbidden matches and scoring of alignments**
  The tool offers the user the possibility to prevent matches of nodes or edges of certain labels on each other. Similarly, a scoring list can be used to assign points for each matching of certain labels, thus selecting the best scoring subgraphs and not the maximum common subgraphs. Each matched node and / or edge pair is scored accoring to the responding labels. These functions are implemented in the script keep.

2 Benchmarking

For benchmarking, we ran different tests to calculate and compare the runtimes of the implemented algorithms for progressive multiple graph alignment: the bk algorithm and the VF2 algorithm. The runtimes of the algorithms were compared in different scenarios: firstly, the number of input graphs was increased while the number of nodes per graph remained the same. Secondly, the number of nodes was increased while the number of input graphs remained the same. The graphs were generated randomly with a probability of 0.7 for an edge between two nodes. In addition, some "real world" data were tested.

Generally, the runtime increased with the number of nodes per graph, as well as with the number of input graphs. This is independent of the algorithm used (Figure 1). However, we noted that the alignment of six graphs with 10 nodes each by the VF2 was faster than the alignment of three graphs (Figure 2A). Since our benchmarking graphs were randomly generated, we did not control how similar the graphs were to each other. Since the runtime is also dependent on the similarity of the graphs (data not shown), we attribute this unlikely finding to random variation

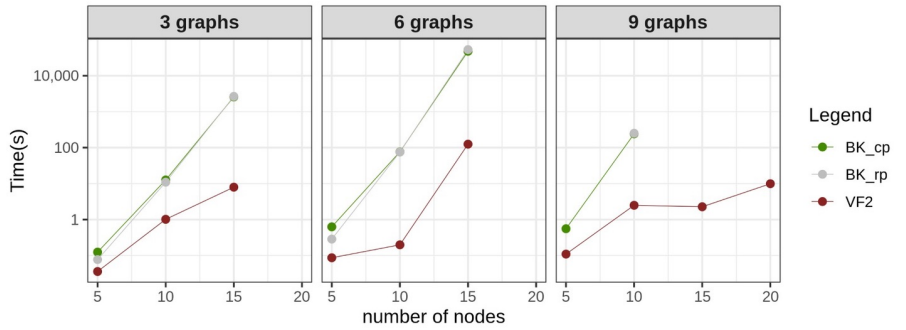


Figure 1: **Running times for raising node and graph count.** The runtime is generally higher using the bk algorithm, increases with the number of nodes per graph and also with the number of graphs that are aligned.BK_cp = bk algorithm using costum pivoting strategy, BK_rp = bk algorithm using random pivoting strategy.

in graph structure. The use of bk does not seem to be practical for graphs with more than 15 nodes, as seen in the extensive increase in th runtime for 6 graphs with 15 nodes aligned via bk algorithm in figure 1. In contrast, the VF2 algorithm seems to terminate with larger and more graphs in a reasonable time. The multiple alignment step always took takes longer than the single pairwise alignments for both implemented algorithms (Figure 2). In addition, we investigated the effects of an optimal guide tree for the MGA compared to a random guide tree. For the bk algorithm, the type of guidetree does not seem to be relevant, but using a random guide tree slows down the running time for a MGA by the VF2 (Figure 3A). Surprisingly, a custom pivoting strategy has no great effect on running time compared to a random pivoting strategy. This could be due to the relatively small graphs used for testing of the bk algorithm(Figure 3B). Figure 4 shows how the VF2 outperformes the bk algorithm by at least 2 orders of magnitude when more than 3 input graphs were given. To test whether some real chemical molecules could be aligned with our tool, the nucleobases adenine, cytosine, guanine, thymine, and uracil referring to DNA and RNA were used as input graphs. For this, 2D JSON files were taken from the pubchem database and parsed. The pairwise alignment step took 4.82s, the MGA step 111.33s.

3 Concluding remarks

The MIGRAINE tool provides two options for solving the problem of finding subgraph isomorphisms in multiple graph objects. The Bron Kerbosch algorithm works well with all kinds of graphs, but is slow in comparison. Large inputs will overload the algorithm due to its recursiv implementation. In general the tool works faster if input graphs have a higher similarity. The VF2 algorithm provides a quicker way to find common subgraphs and is recommended for larger graphs.

MIGRAINE provides different options to modify those algorithms. It is possible to use custom scores, to score only specific labels, exclude custom forbidden label matches or use an anchor for graph alignment. There is also a way to visualize the final matching as a graphic for a quick overview.

Since it is possible to use a custom scoring list for node and edge labels, this tool could be used for many different studies, like finding reaction centres in molecules or analyzing city maps. It is possible to score mainly with size of the largest common subgraph or scores derived from user input. In this way, users can provide additional information, to further specify the scoring. Thus, MIGRAINE provides a basis for all kinds of applications.

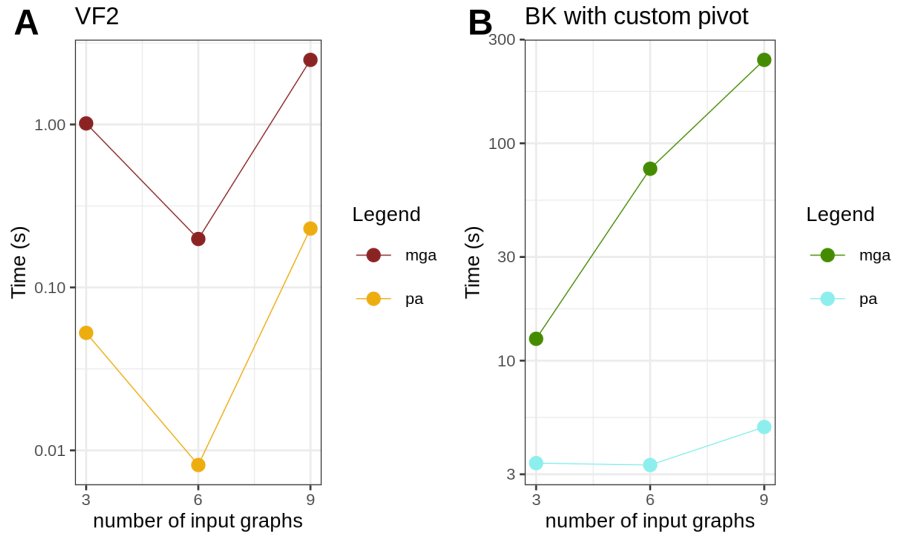
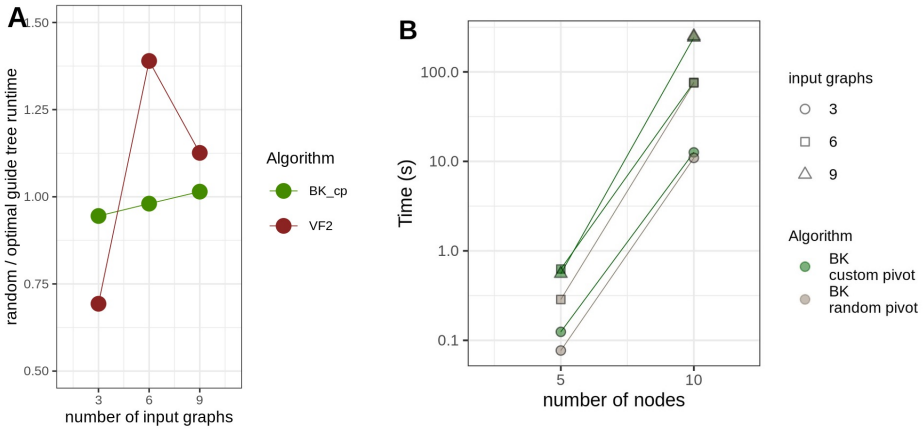


Figure 2: **Running times of pairwise vs. multiple graph alignment step.** For both algorithms running time for pairwise alignment step (pa) is shorter than for multiple alignment step (mga). The used graphs were composed of 10 nodes each.



(a) (b)

Figure 3: **A. Random vs. calculated guide trees.** For the bk algorithm an optimal guidetree does not seem to affect the runtime very much, whereas the VF2 algorithm with an optimal guidetree seems to run much faster, if the number of input graphs rises. Every graph in a contains 10 nodes. BK_cp = bk algorithm using costum pivoting strategy. **B. Random vs. costum pivoting strategy for bk algorithm.** For our example graphs pivoting strategy in the bk algorithm does not seem to influence the running time of the bk algorithm.

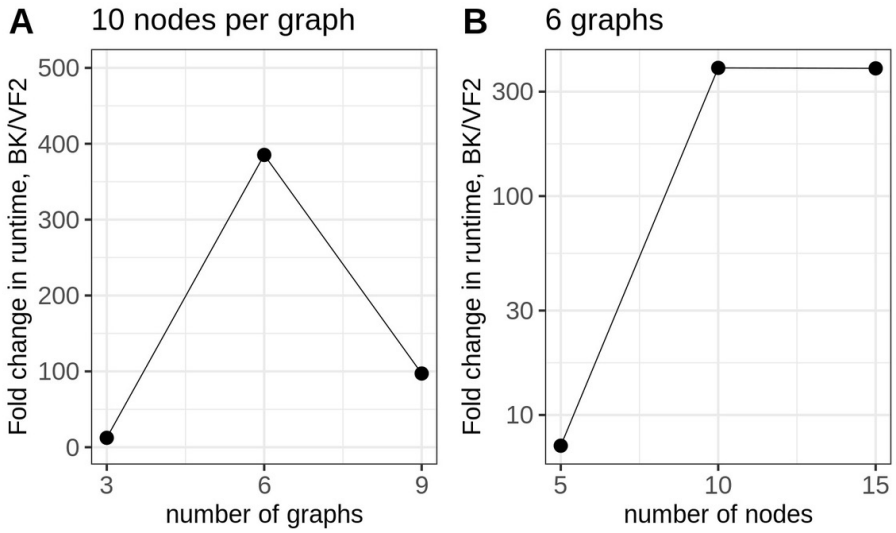


Figure 4: textbfComparing running times of the bk and the VF2 algorithms. A. For 10 nodes per graph and 3 input graphs, running time of bk and VF2 algorithm is approximately the same, but the relative running time of bk to VF2 algorithm increases strongly with increasing number of input graphs. B. Starting at graphs composed of 10 nodes each, bk algorithm is about 300fold slower that the VF2 algorithm.

References

[1] BERKEMER, S.J., HOENER ZU SIEDERDISSEN, C. and STADLER, P.F. (2019): "Compositional Properties of Alignments", Mathematics in Computer Science, submitted.

[2] FENG, D. (1987): "Progressive sequence alignment as a prerequisitetto correct phylogenetic trees", J Mol Evol. 25 (4): 351–360.

[3] SOKAL, M. (1958): "A statistical method for evaluating systematic relationships", University of Kansas Science Bulletin. 38: 1409–1438.

[4] BRON, C., KERBOSCH, J. (1973): "Algorithm 457: finding all cliques of an undirected graph", Commun. ACM, ACM, 16 (9): 575–577.

[5] KOCH, I. (2001): "Fundamental study: Enumerating all connected maximal common subgraphs in two graphs", Theoretical Computer Science 250(1-2): 1-30.

[6] CORDELLA, L.P., FOGGIA, P., SANSONE, C., VENTO, M. (2001): "An Improved Algorithm for Matching Large Graphs", 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen, pp. 149-159.

[7] ELLISON, C., CRUTCHFIELD, J.P. and other NetworkX maintainers, "https://networkx.github.io"

[8] CARLETTI, V., FUOGGIA P., SAGGESE, A., VENTO, M. (2018): "Challenging the Time Complexity of Exact Subgraph Isomorphism for Huge and Dense Graphs with VF3", IEEE Trans Pattern Anal Mach Intell. 40(4):804-818.

[9] CORDELLA, L.P., FOGGIA, P., SANSONE, C., VENTO, M. (2004): "A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs", IEEE rans Pattern Anal Mach Intell. 26(10):1367-1372

Appendix A Help of the Python module

```
----Multiple Graph Alignment Tool----
Aligns multiple graphs in a progressive way using Bron Kerbosch or VF2 algorithm.

GRAPH OPTIONS
-i <input path   filepath from which all the graph files are read. Files can be
                 .graph, .json or .graphml
-o <name>        define name for the current aligment run
-c <input file>  input a graph file as an anchor for alignment. This graph needs to
                 be common to all graphs that are to be aligned and acts as a basis
                 to the alignment

ALIGNMENT OPTIONS
-b or --bronkerbosch       switches to multiple alignment via the Bron-Kerbosch
                           algorithm. Default: VF2
-v                         random pivoting strategy for bk algorithm. Combination
                           with pre-clique option is excluded.
                           Default: costum pivoting.
-s or --score <input file> input a list of scores for matching node labels, edge
                           labels or both. Default: Scoring based on the size of
                           the largest common subgraph found between two graphs
-f or --forbidden          while inputting a scoring list, score via size of largest
                           common subgraph after excluding forbidden label matches.
-g or --labelling <string> define what labels should be used for scoring: node
                           labels, edge labels or both. Default: both

GUIDE TREE OPTIONS
Default: guide tree is generated from scores or size of the largest common subgraph
computed via pairwise alignment
-n or --newick <file> input a guide tree file in newick format
-r or --randomt       additionally creates a random guide tree and saves it in
                      newick format
-t or --showt         saves the generated guide tree as .png
-q or --savetn        saves generated guide tree in newick format

RESULT OPTIONS
-m <string>    save multiple alignment as graphml file. Chose between "end" and
               "all". end only saves total matching. all saves all the matchings
               in between as well.
-d or --drawm  saves multiple alignment visualization .png

ADDITIONAL FEATURE:
--graphgen    a tool, that makes random graphs with properties that are set through
              user input

Note: Clique option only available for BK. Disconnected directed graphs can only be
assessed via BK algorithm. Input graphs must be all directed or undirected.
```

Appendix B Program output for the test data

A possible usage using the VF2 algorithm could be:
`python3 main_func.py -i input_dir -o output_dir`

```
Multiple Graph Alignment: vf2_test
Alignment: ethane, methane, propane
        Evaluating scores for Guide Tree...
There are 48 possible Alignments.
Graph1: 6324, Graph2: 297, Alignment length = 5
Score: 0.0 - ethane & methane
Matching:
```

```
{'6324_2': '297_1', '6324_7': '297_2', '6324_6': '297_3', '6324_1': '297_4', '6324_8
': '297_5'}
There are 72 possible Alignments.
Graph1: 6324, Graph2: 6334, Alignment length = 8
Score: 0.0 - ethane & propane
Matching:
{'6324_1': '6334_1', '6324_2': '6334_2', '6324_4': '6334_3', '6324_3': '6334_4',
'6324_5': '6334_5', '6324_6': '6334_6', '6324_8': '6334_7', '6324_7': '6334_8'}
There are 24 possible Alignments.
Graph1: 297, Graph2: 6334, Alignment length = 5
Score: 0.0 - methane & propane
Matching:
{'297_1': '6334_1', '297_2': '6334_2', '297_4': '6334_3', '297_3': '6334_4', '297_5
': '6334_5'}
        Generating MGA...
alignment step: propane, methane
There are 72 possible Alignments.
Graph1: 6334, Graph2: 297, Alignment length = 5
alignment step: propanemethane, ethane
There are 144 possible Alignments.
Graph1: Alignment, Graph2: 6324, Alignment length = 8
Saved Final Matching as Graphml
Graph aligment used up a total time of: 0.17000937461853027 s.
```

A possible usage using the bk algorithm could be:
`python3 main_func.py -i input_dir -o output_dir -b`

```
Multiple Graph Alignment: bk_test
Alignment: ethane, methane, propane
        Evaluating scores for Guide Tree...
Score: 0.0 -  ethane & methane
Best Scoring Clique:
[('6324_7', '297_4'), ('6324_8', '297_5'), ('6324_2', '297_1'), ('6324_1', '297_2'),
 ('6324_6', '297_3')]
Score: 0.0 -  ethane & propane
Best Scoring Clique:
[('6324_1', '6334_2'), ('6324_8', '6334_5'), ('6324_4', '6334_7'), ('6324_2', '6334
_1'), ('6324_3', '6334_6'), ('6324_6', '6334_3'), ('6324_7', '6334_4'), ('6324_5',
'6334_8')]
Score: 0.0 -  methane & propane
Best Scoring Clique:
[('297_1', '6334_2'), ('297_3', '6334_6'), ('297_4', '6334_7'), ('297_2', '6334_1'),
 ('297_5', '6334_8')]
        Generating MGA...
alignment step: propane, methane
alignment step: propanemethane, ethane
Saved Final Matching as Graphml
Graphs aligment used up a total time of: 11.985685348510742 s.
```

Additionally to the terminal output there is also a .txt file called matches.txt, which lists all nodes and edges of the final matching and their respective matched nodes or edges.