A wrapper for *segemehl* output

Hermann Pauly

Abstract

Alternate splicing is a cellular process that massively increases the amount of transcripts encoded in a single gene. Apart from commonly spliced mRNAs from single genes, different splicing variants, both functional and pathological, have been observed. The NGS mapping software *segemehl* maps multi strand fusion and circular transcripts, but offers no method to visualise them yet. Here I present a software to automatise selective visualisation of *segemehl* data files. I applied it to sequenced and processed skin cell data and found numerous multi strand fusion transcripts with low abundancies each.

1 Introduction

Alternate splicing is a cellular process that massively increases the amount of different RNA and proteins available to organism without the need to store more genetic information. The variability is achieved by cutting gene transcripts at defined splice-sites and fusing desired parts to different functional mRNAs. Next to gene silencing and methylation, alternate splicing is another mechanism that allows cells to differentiate. On the other hand, erroneous splicing events can cause various diseases, thus making their understanding crucial to genomic research.

According to Hoffmann et al. [HOD+14], data from NGS sequencing however has shown, that not all splice events connect exons from a single gene to a regular mRNA, and not all seemingly aberrant splicing needs to be pathologic. Splicing can connect both ends of an RNA strand so it forms a circular RNA, which may have regulatory roles. In some cases and organisms also the fusion of RNA transcribed from different strands of a chromosome or even from different chromosomes could be observed.

To analyse genomic data, in the past, we were limited to low-throughput sequencing methods like Sanger sequencing. However, the next generation sequencing (NGS) methods of nowadays enable us to obtain near complete quantitative and qualitative genomic snapshots of cell states in a single go with very high accuracy.

Such proceedings typically produce huge amounts of short RNA sequence fragments at base-pair level accuracy. To further analyse - or visualise - these datasets, it is neccessary to recompile proper fragments and reassemble the RNAs they stem from, thus reconstructing the whole transcriptome. Three assembly strategies are commonly followed: *ab initio* assembly based on a reference genome, *de novo* assembly and methods combining both.

De novo assemblers usually work by constructing De Bruijn Graphs, that is graphs grouping sets of overlaps. The *Rnnotator, Multiple-k*, and *Trans-ABy-SS* softwares construct multiple De Bruijn graphs and postprocess them to reassemble contigs. Other assemblers like *Trinity* and *Oases* reassemble RNAs and isoforms directly from single De Bruijn graphs. *De novo* assembly does not require reference genomes, so it can be applied to organisms for which no complete genome sequencing has been done yet. As it only works on transcribed RNA, it cannot be confused by long, ambiguous, or unconventional DNA intron sequences.

Ab initio assemblers rely on existing reference genomes for their task. They align all fragments to the reference genome, create graphs of overlapping fragments and then traverse the graphs to reconstruct isoforms either by statistical (*Scripture*) or by pathfinding (*Cufflinks*) methods. To align fragments to genome, two methods are common. Burrow Wheeler Transform finds all suffix combinations of a string and thus allows to match strings with few differences very quickly, finding all combinations of possible matches. Seed-and-extend algorithms search exact matches for parts of their input strings and try to extend them along the genome with Smith-Waterman methods, thus allowing parametrisation of the scoring. They tend to match around known splice sites. These reference-based techniques can split down assembly problems and thus work on lower powered machines, they can match fragments with low abundancy, bridge sequencing gaps, and discard sequencer artifacts (which will not match the genome very well in any place) (compare [MW11]).

Segemehl is a reference-based NGS mapping utility following a seed-and-extend approach. It deliberately omits searching for known splice sites to be able to find circular RNAs and new kinds of *fusion transcripts*, transcripts that stem from distant loci of different strands of a chromosome, or fusions of RNA transcribed from multiple chromosomes. In this work the latter two events are termed multistrand events.

Here I present the foundation for a software that visualises the output of *segemehl* mapped sequencing data.

2 The program

My goal was to develop a software that is able to create visualisation plots for RNA mappings from *segemehl* processed files in an automated way. It should be able to visualise "standard" RNAs, with exons originating from a defined area on a specific chromosome, as well as novell kinds of transcripts detected with the *segemehl* method. As NGS methods produce large data files, it had to be fast and needed to store the results efficiently in RAM while running. Due to the large numbers of RNA transcripts that are typically processed in NGS machines, user-definable selection of output needs to be implemented. I chose the following filtering options: detection of all multistrand RNAs, detection of all circular RNAs, and all isoforms containing user-defined chromosome and position pairs. Future versions also aim to further refine the search by allowing to filter by link depth, that is the number of times a connection of any two splits within transcripts and isoforms occured in the input files. The software should abstract the temporary representation, input and output backends, so future versions can be extended to work with other mapping tools' output files and customise resulting file formats and visual appearance.

2.1 Overview and used third-party software

The visualisation program with the working title *chipboard* is written in *C++* for a balance of speed and memory efficiency on the one hand, and safe and readable code on the other hand. I automated compiling by using the *cmake* build managing system([cma]). For GUI creation, I used *Qt4.8* ([qtp]). For development I used the *gcc* compiler suite v4.8.2 and *cmake* v3.1.0.

2.2 Plot types

2.2.1 Isoform tree plots

In an effort to increase the information content and readability within plotted RNAs with multiple isoforms, I decided to visualise RNA not as a linear strand with connection lines (see fig. 1), but as a tree structure. Split reads are nodes, with their basepair lengt reflected in the node length, while splicing events are edges, allowing a more intuitive understanding of existing isoforms.

2.2.2 Circular plots

Circular RNA will be visualised as ring structures. The size and angle of ring fragments for each split indicate its relative length, compared to the whole ring.

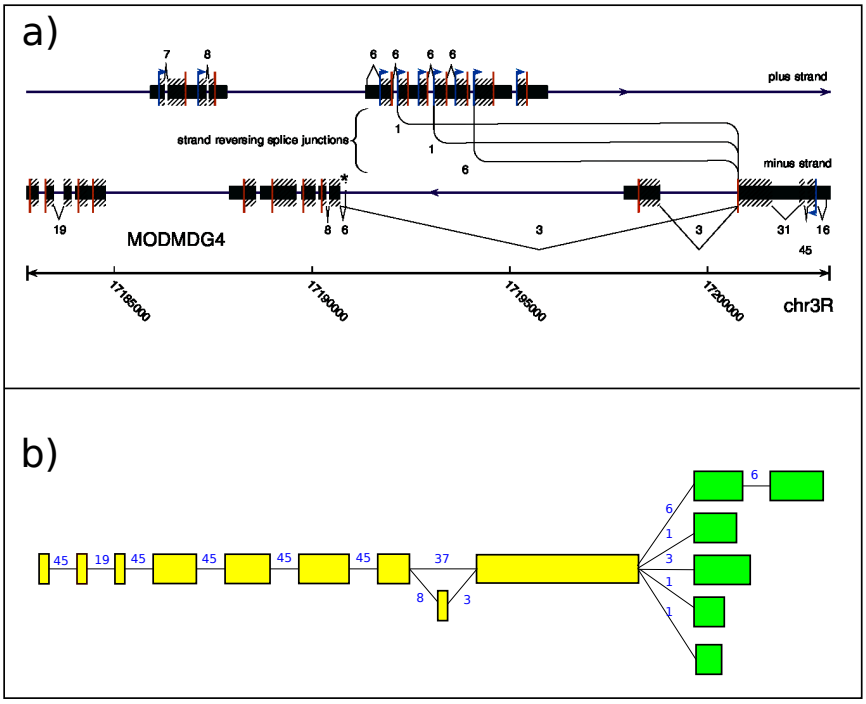

Figure 1: The same information displayed **(a)** in a traditional linear presentation and **(b)** in a treemap mockup (note that the reverse strand is presented in another colour, as it will be detected differently by *segemehl*

2.3 Reassembly of *segemehl* split reads

Segemehl with its *split* option enabled divides input RNA reads into fragments, maps each of those split fragments to the chromosomes and position it fits best and stores them, line-wise and in order, inside the *sam* file, together with information how it had performed the splitting and mapping. It thus provides a guideline how to reassemble the input reads. With each split the following information is stored: (1) the chromosome, read position, and read direction ("strandiness") it was mapped to, (2) information about mapping length and quality, (3) a number specifying the order of splits along the original read, (4) information about chromosome, position and strandiness of the previous and next split in this read, if there are any (see [smm]).

For reassembly I treat reads as doubly n/m-linked directed, acyclic graphs, where nodes resemble exons and edges resemble splicings. Each node stores the length in base pairs and the chromosome of origin of the split it represents, as

well as edges resembling detected 3' and 5' splice events. The chromosomes are represented as ordered maps, which map 3' and 5' positions to the respective nodes (compare fig. 2) to achieve $O(\log n)$ lookup times. By continuous adding nodes to existing subgraphs, the sequencer read fragments get reassembled to full RNAs, with all their isoforms encountered merged into a single graph.

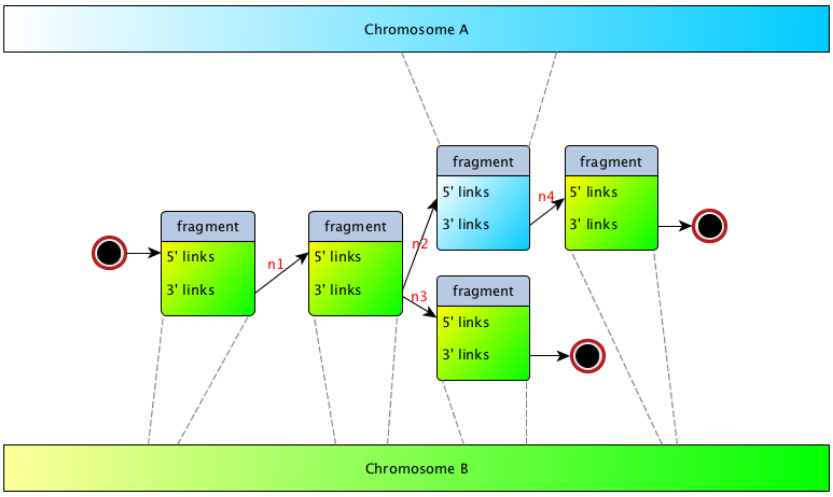

Figure 2: *chipboard*'s internal datastructure. Graph nodes resemble split fragmens/exons and edges resemble splicings. Chromosomes are resembled as ordered maps of node-pointers to allow for quick selection.

2.3.1 Reassembling the original RNA sequences

Following *segemehl*'s philosophy according to Hoffmann et al. [HOD+14], I did not test for meaningfulness of splice sites. I used the information I had extracted (see 2.3) to reassemble the original inputs by a series of simple steps on every line of *.sam* data: First the program determines the chromosome positions the 5' and 3' ends map to. As *segemehl* does neither fill in the "match length" property nor the match position of the 5' end, the program reconstructs them from the *cigar* string, which encodes assumed matches, mismatches, insertions and deletions in the mapping process (see [sam]). Next, the program compares the resulting chromosome positions with existing nodes' data and creates a new node if no matching one exists. Then the program looks, if the current split's 3' and 5' ends are linked to other splits. If there are any linked splits, a check for existing modes is done. Existing nodes are doubly-linked immediately. If a split's upstream predecessor

is linked, the predecessor's link depth counter to the current split is increased. As *segemehl* writes all encountered fragments in order, this is assured to find correct link depths. If the *.sam* file has been sorted or modified after the *segemehl* processing, link depths may not be counted correctly, but due to the usage of doubly linked nodes, all isoforms from the original input file will be reconstructed. Assuring correct link depths in randomly ordered input files is possible if the input file gets processed twice, but was considered impractical standard behaviour, as it would double the relatively long runtime.

2.3.2 Detecting multistrand reads

One of the key features of *segemehl* is the mapping of read fragments to different chromosomal strands of origin. To detect such multistrand chromosomes on user request, the information from 2.3 is applied straightforward: if a split's successor is on a different chromosome, the respective split is added to a list of multistrand seeds, which can be expanded to a full isoform tree (see 2.2.1) for plotting on the available backends.

2.3.3 Detecting circular reads

Another feature of *segemehl* is the detection of circular RNAs. The split segments of a linear RNA read follow each other in a definite order in the resulting *.sam* file. This can be seen in both an ascending ordering number and an ascending position on the chromosome (or descending in case of reverse direction). A circular transcript can be identified by an ascending order number combined with a position which lies upstream the chromosome position of a split with lower read number (with respect to the reading direction). A split with these properties gets added to a list of circular seeds, to be expanded to full circular graphs (see 2.2.2) if the user requests circular detection.

2.4 Plotting

In 2.2.1 and 2.2.2 I indicated, that only single splits of subgraphs interesting to the user get saved to a list for later expansion to full (sub-)graphs. There were two reasons for the decision to save graph seeds instead of full graphs: (1) memory consumption was a huge concern during development, and (2) as there is no way to find out, when all copies and isoforms of a read have been processed, a full copy of each graph would have to be updated every time another read adds to it. Thus only one node of the interesting graph is saved and expanded with a breadth-first traversal of its linking edges, as seen in algorithm 1. Note that this approach

evaluates the full subgraph of nodes connected to the seed, which may also contain nodes that share no primary connection to it, e. g. are an isoform of an exon that exists only in some isoforms of the queried graph, but never occur in combination with the query seed. No filtering of possibility or probability of isoforms is applied. There is no immediate drawing done, the method generates coordinates which can then be handled or modified by the drawing backend.

The drawing backend encodes basepair length in the size of the resulting fragments, chromosome association in colours, and displays link depth numerical.

Input: $node$: one node of a graph
Output: The whole graph which is connected to $node$
$q \leftarrow$ empty queue
push $node$ to q
while q *not empty* **do**
  $N \leftarrow$ pop first element from q
  mark N as visited
  create a visualisation node for N
  forall the *unvisited 5' links* $el5$ *in* N **do** push $el5$ to q
  forall the *3' links* $el3$ *in* N **do**
    create a visualisation edge $N \rightarrow el3$ with link depth label
    if $el3$ *unvisited* **then** push $el3$ to q
  end
end

Algorithm 1: Breadth first traversal (BFS) to expand a complete graph from a single member node

2.4.1 Isoform trees

The tree visualisation is generated from the coordinates generated by algorithm 1. Starting from the first node without links on the 3' end drawn at the leftmost x position, the nodes are drawn. Depending on the number of nodes linked to a node's 5' end, those 5'-linked nodes get drawn recursively with an offset in y-position (compare algorithm 2).

2.4.2 Circular reads

The basepair length of each split is compared to the basepair length of the complete circular RNA to determine which fraction of the ring will be assigned to it. This method does not treat the special case of a circular graph with isoforms properly.

Input: $node$, x-coordinate x, y-coordinate y
Result: Draw tree graph representation

Start with $node$ = *node without predecessors,* $x = 0, y = 0$

function *naiveLayout(*$node, x, y$*)***:**
    draw $node$ *at position (*x, y*)*
    $nextX \leftarrow x + 1$
    $y0 \leftarrow -$*(number of 5' links* $/ 2$*)*
    for $i \leftarrow 0$ **to** *number of 5' links* **do**
        $nextY \leftarrow y0 + i$
        $nextNode \leftarrow$ *5'links[i]*
        draw edge to $(x + 1, nextY)$
        naiveLayout(nextNode, nextX, nextY)
    end
end

Algorithm 2: Naive tree layout. More complex graphs may create colliding coordinates for nodes

2.5 Image export

At the moment, only the export of programmatically generated Adobe Encapsulated PostScript (*.eps*) is supported, but the program is designed to ease implementation of drawing backends. As another option, the reconstructed and filtered graphs can be exportet to *GraphML* format for visualisation with 3rd party software.

3 Results

3.1 Test cases

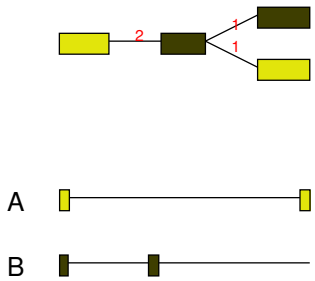

Figure 3: Treeplot of RNA that contains multistrand events and two isoform, generated from test data.

To create test cases, I used a custom *python* script which simulates chromosome data through randomly drawing from the nucleid base letters $[A, C, G, T]$ and writing them into a *.fasta* file. From these simulated chromosomes, I copy-pasted segments into another *.fasta* file to simulate sequencer reads. Then I used *segemehl* to remap the simulated reads to the simulated chromosomes. This allowed me to know the desired results and quickly spot errors during development.

When I allowed *segemehl* to split input reads and map the fragments to different chromosomes (multistrand reads), it found the origins of all fragments correctly. However, when read length exceeded 120 bases, *segemehl* often crashed with memory access errors.

Visualisation of output files with *chipboard* worked well with multistrand RNAs that have only a small number of isoforms (see fig. 3). More complicated transcripts will result in skewed output, however, as nodes farther down in the tree may have multiple 5'links themselves, thus changing their respective y coordinate offset in ways that collide with sibling nodes' positions.

3.2 Real world data

To test *chipboard* on real world data, I successfully ran it on 41 - 69GB files from [KWF+12], where it detected millions of multistrand RNAs per file. Sampling of generated output files showed that detected strand-switching events tend to be short (2-3 exons) and have a low link depth (never above 4 in 20 randomly picked output images), see fig. 4 for an example. Full evaluation of all found events was not done.

The available data has been pre-filtered for poly-A reads, so circular transcripts are not contained, as poly-A tails are signalling structures of valid linear mRNA. Hence the search for circular transcripts was not enforced.

3.3 Performance

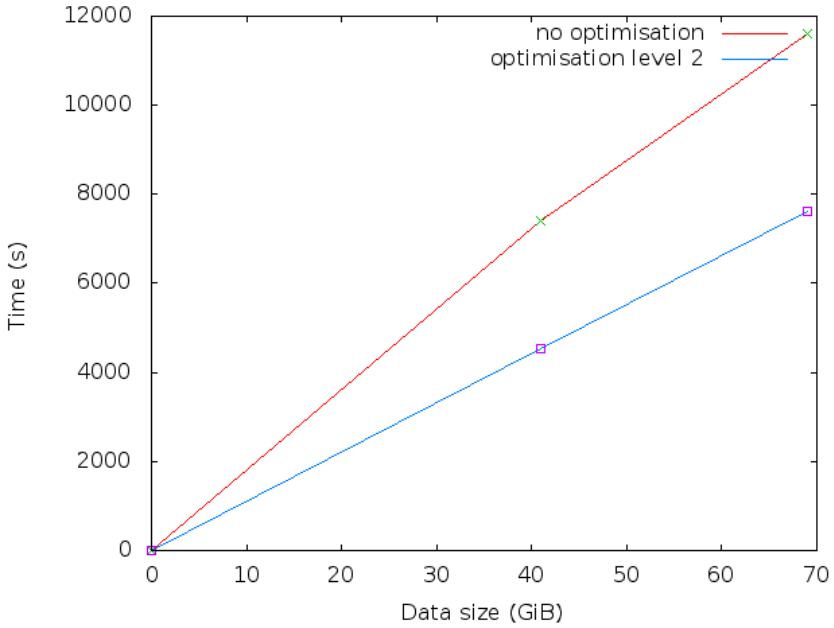

Figure 5: Runtime comparison on *rhskl5* workstation. Runtime increases in linear fashion with data size, while optimised code runs 1/3 faster. Times were taken for 57 MiB, 41 GiB and 69 GiB files.

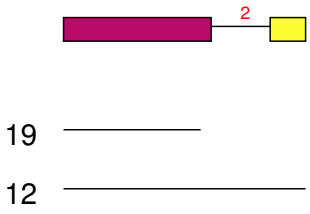

Figure 4: Treeplot of a multistrand split event found in real dataset. Short RNA consisting of a 100-base fragment from chromosome 19 and one 24-base fragment from chromosome 12, found twice in the dataset.

Running on datasets of different size, *chipboard* showed linear ($O(n)$) runtime. On the *rhskl5* workstation of Universität Regensburg, about 20GB of data could be processed per hour without optimisation flag; setting the optimisation to level 2 (`-O2`) increased the performance to 32 GB/h (see fig 5). Profiling showed, that 50% of the runtime is spent tokenizing text strings from the human-readable *.sam* input files to parse them for data.

4 Discussion

The software *chipboard* is a tool to visualise NGS sequencing data which has been mapped with the *segemehl* tool. It allows to scan for RNA assembled of exons from different chromosomes and is at the time of writing the only software known to the author that automates the visualisation of such events. In addition, it allows the user to select RNA which contains exons from specific chromosome positions. With tree-like isoform graphs, it tries to increase visual information content in comparison with more common visualisation approaches. Filtering for possible of probable event is omitted; *chipboard* shows the raw findings of *segemehl* directly.

In its current state, *chipboard* is stable but not complete. Detection and visualisation of circular graphs has been deactivated in the current build, as it is only rudimetary and not yet thoroughly tested. Also, as hinted in 2.4.2, circular RNAs with splice variants can be ambiguous. When traversing the graph, the program could get stuck in a non-circular isoform. To avoid this, some shortest path search like Djikstra's algorithm could be applied to find a complete path from the start-node to the end node.

The visualisation of tree graphs works satisfyingly for simple graphs with a low number of isoforms. RNAs with many complex isoforms will create graphs with overlapping node coordinates. To adress this, a full-fledged graph layout algorithm must be used. I suggest to refrain from force-based methods in favour of hierarchy-based methods like Sugiyama's method. Although force-based approaches create graphs which are tendentially more aesthetically pleasing, their average runtime is far higher (see [Tam13]).

Although the processing speed of 32GB per hour seems quite moderate, some optimisation is still desirable. When parsing the *.sam* input files, it is impossible to predict, which split will be read next, and what graph it may belong to. This makes parallellization very hard. Mutexes could be used to lock all nodes of a graph for a single process, but this would include traversing up to two complete graphs for every split that is added, plus the time needed to wait for other processes releasing locks, so no critical speed gain should be expected from this. However, when running the program, about 50% of the runtime is used tokenizing string data. This is done serially, so a dual-thread approach could be used, where one thread tokenizes strings and pushes them to a thread-safe dequeue buffer, while a second thread pops the tokenized strings and constructs the graphs from them. This way, it should be possible to process the same data in half the time.

To improve the usefulness of *chipboard*, visualisation should be extended in various ways. Unused regions of the chromosomes, against which splits are mapped, should be shortened in the graphical output. Chromosome position numbers should be displayed, and the splits in the output graphs should display identi-

fiers to allow connecting them to their respective exon regions. Also a method to list all findings of interest in a text file for further processing would prove a usefull addition and should be trivial to implement.

The analysis of real world data (3.2) showed numerous findings of strand-switching events in RNA synthesis, but due to the short length and low link depth found in subsamples the reliability of those findings must be doubted. The samples seem to imply sequencer artifactcs rather than real discoveries, but to make any reliable statements, proper statistical analysis has to be done on all the findings. The subsampling of 20 singular events out of 2 million possible findings is far from being representative.

A development snapshot of the program's source code can be accessed on my github page.

List of Figures

List of Algorithms

References

[CLRS99] CORMEN, Thomas H. ; LEISERSON, Charles E. ; RIVEST, Ronald L. ; STEIN, Clifford: *Introduction to algorithms*. 1999

[cma] *CMake*. http://www.cmake.org

[HOD+14] HOFFMANN, Steve ; OTTO, Christian ; DOOSE, Gero ; TANZER, Andrea ; LANGENBERGER, David ; CHRIST, Sabina ; KUNZ, Manfred ; HOLDT, Lesca ; TEUPSER, Daniel ; HACKERMULLER, Jorg ; STADLER, Peter: A multi-split mapping algorithm for circular RNA, splicing, trans-splicing and fusion detection. In: *Genome Biology* 15 (2014), Nr. 2, R34. http://dx.doi.org/10.1186/gb-2014-15-2-r34. – DOI 10.1186/gb–2014–15–2–r34. – ISSN 1465–6906

[KWF+12] KRETZ, Markus ; WEBSTER, Dan E. ; FLOCKHEART, Ross J. ; LEE, Carolyn S. ; ZEHNDER, Ashley ; LOPEZ-PAJARES, Vanessa ; QU, Kun ; ZHENG, Grace X. ; CHOW, Jennifer ; KIM, Grace E. ; RINN, John L. ; CHANG, Howard Y. ; SIPRASHVILI, Zurab ; KHAVARI, Paul A.: Suppression of progenitor differentiation requires the long noncoding RNA ANCR. In: *Genes & Development* (2012)

[MW11] MARTIN, Jeffrey A. ; WANG, Zhong: Next-generation transcriptome assembly. In: *Nature Reviews* (2011)

[qtp] *Qt cross platform application framework V4.8*. http://qt-project.org

[sam] *Sequence alignment/map format specification*. https://samtools.github.io/hts-specs/SAMv1.pdf

[smm] *A segemehl manual, v0.1.7 r1*. http://www.bioinf.uni-leipzig.de/Software/segemehl/segemehl_manual_0_1_7.pdf

[Tam13] TAMASSIA, Roberto: *Handbook of graph drawing and visualization*. 2013