Deliverable D1.3

Funding Scheme: THEME [ICT-2007.8.0] [FET Open]

Paving the Way for Future Emerging DNA-based Technologies: Computer-Aided Design and Manufacturing of DNA libraries

Grant Agreement number: 265505

Project acronym: CADMAD

Deliverable number: D1.3

Deliverable name: Report on the features of version 1.0 CADMAD GUI development

Contractual Date[1] of Delivery to the CEC: M24
Actual Date of Delivery to the CEC: M24
Author(s)[2]: Jonathan Blakes, Natalio Krasnogor
Participant(s)[3]: UNOTT
Work Package: WP1
Security[4]: Pub
Nature[5]: R
Version[6]: 1.0
Total number of pages: 10

1 As specified in Annex I

2 i.e. name of the person(s) responsible for the preparation of the document

3 Short name of partner(s) responsible for the deliverable

4 The Technical Annex of the project provides a list of deliverables to be submitted, with the following classification level:
Pub - Public document; No restrictions on access; may be given freely to any interested party or published openly on the web, provided the author and source are mentioned and the content is not altered.
Rest - Restricted circulation list (including Commission Project Officer). This circulation list will be designated in agreement with the source project. May not be given to persons or bodies not listed.
Int - Internal circulation within project (and Commission Project Officer). The deliverable cannot be disclosed to any third party outside the project.

5 **R (Report):** the deliverables consists in a document reporting the results of interest.
P (Prototype): the deliverable is actually consisting in a physical prototype**,** whose location and functionalities are described in the submitted document (however, the actual deliverable must be available for inspection and/or audit in the indicated place)
D (Demonstrator): the deliverable is a software program, a device or a physical set-up aimed to demonstrate a concept and described in the submitted document (however, the actual deliverable must be available for inspection and/or audit in the indicated place)
O (**Other):** the deliverable described in the submitted document can not be classified as one of the above (e.g. specification, tools, tests, etc.)

6 Two digits separated by a dot:
The first digit is 0 for draft, 1 for project approved document, 2 or more for further revisions (e.g. in case of non acceptance by the Commission) requiring explicit approval by the project itself;
The second digit is a number indicating minor changes to the document not requiring an explicit approval by the project.

Abstract

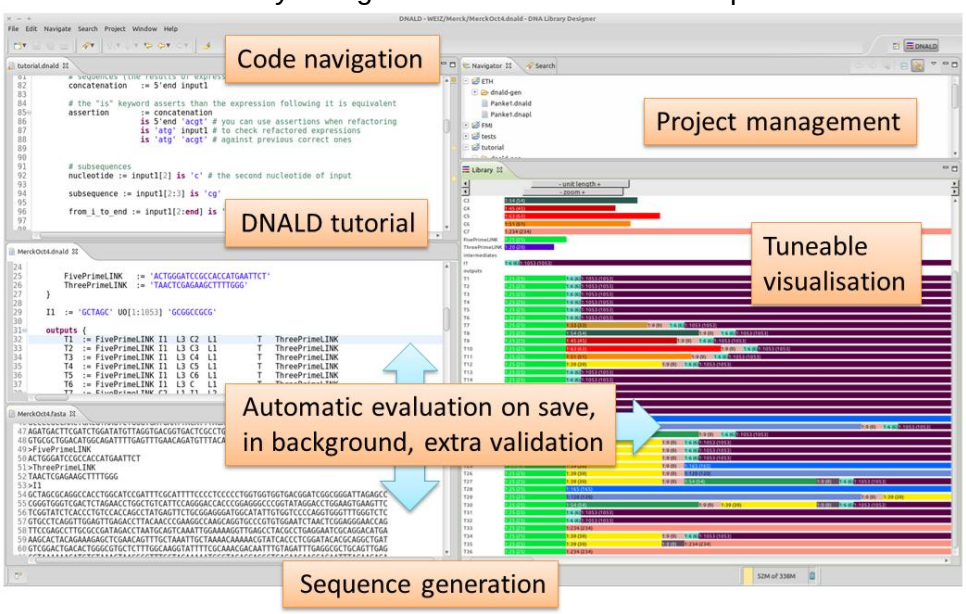
We have designed and implemented a domain specific programming language for combinatorial DNA Libraries design (DNALD) that is supported by a graphical user interface (GUI) the DNA Library Designer integrated development environment (IDE) for DNALD.

In this reporting period we provide a summary of work on updates to the existing functionality of the IDE as well as details of the addition of new language-based features to the GUI presented in deliverable 1.3 (this reporting period). We also present a new graph-based visualisation that completes work on D1.3 but also establishes the basis for the deliverables to be done during the *next* review period, namely D1.4. We describe in detail a novel compact data structure that is at the core of 3 critical CADMAD activities: **(I)** DNALD expression/library visualisation (D1.3, current reporting period), visual programming interface vDNALD (D1.4, next reporting period), **(II)** deriving library construction plans from DNALD library designs (D2.5 led by WEIZ, current reporting period), and **(III)** the reverse parsing of unstructured sequences into DNALD libraries in D1.5 (led by Nottingham, next reporting period).

Keywords[7]**:**

DNA Library Designer IDE, GUI, DNALD, vDNALD

DNA Library Designer's main functions and capabilities:



During this reporting period we have worked on improving the software engineering, language features, the IDE and the data structures.

[7] Keywords that would serve as search label for information retrieval

1. Software engineering

This section briefly describes the architecture of the software that constitutes deliverables D1.3 and D1.4.

DNA Library Designer is a sophisticated integrated development environment (IDE) for DNALD that enables biologists to design and explore combinatorial DNA sequence libraries with the support of a real programming editor. The IDE is an Eclipse-based application developed on top of the software stack shown in figure 1 that can be installed alongside other plugins in an existing Eclipse installation or used as a standalone product on Linux, Mac and Windows platforms.

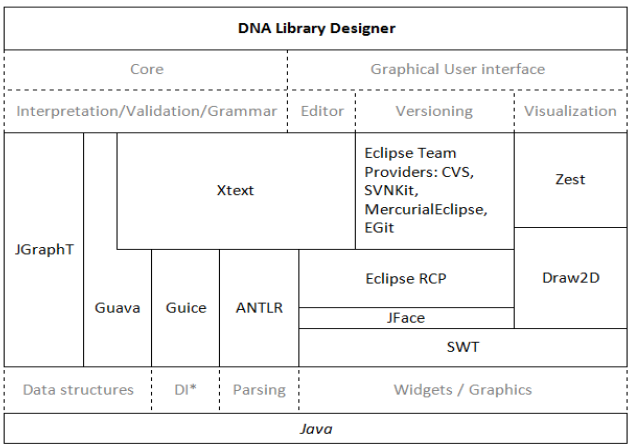


Figure 1: Components of DNA Library Designer. The diagram should be viewed as a stack where each layer above is dependent on functionality provided by the layer below.

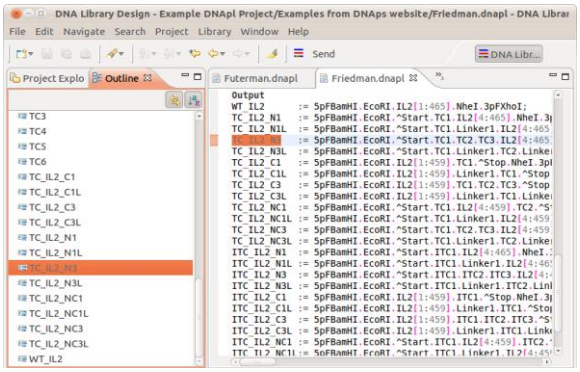
Eclipse provides the plugin framework and Workbench UI for our Rich Client Platform product, including project management, text search and file comparison features. Xtext is an open source domain-specific language project which we use to handle parsing and semantic model-backed editing of DNALD files. Guava is a Google library for the Java language providing high quality basic data structures, the use of which contributes significantly to the readability and performance of our code. We use JGraphT for some graph algorithms related to library evaluation and visualisation, and Draw2D/Zest for drawing and graph layout respectively.

A major component of the work that was carried out during this reporting period was to re-factor and re-structure the “under-the-hood” architecture of the DNALD IDE. We enhanced our underlying datamodel and associated evaluation steps that enable a more meaningful visualisation and faster computation of output DNA sequences. These key software engineering innovations for this reporting period have made the product more robust and, ultimately, more scalable. In this context, scale refers to the potential range of problem-specific plug-ins that could be added to the system. In turn, this helps ensure that DNALD will remain future-proof.

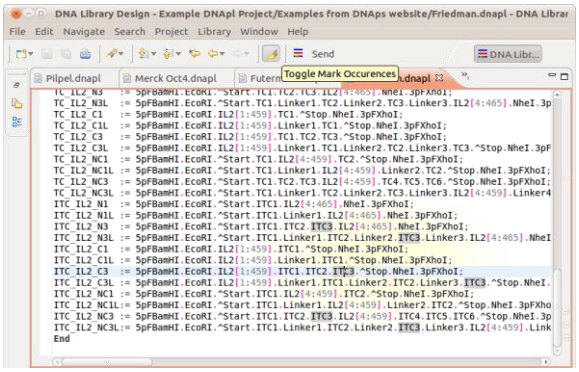
2. New GUI features

Here we report additional GUI features of D1.3 linked to the language.

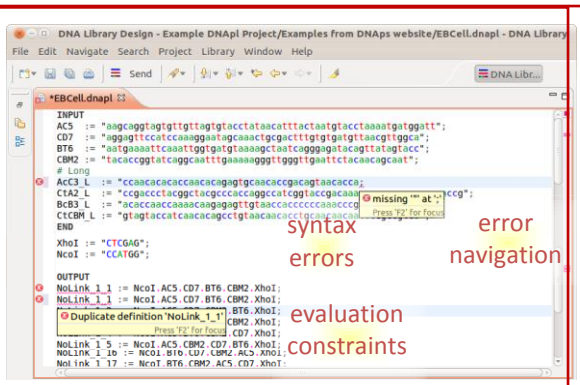
DNA Library Designer fully leverages Xtext and the Eclipse to provide DNA combinatorial library programmers the many features they would be familiar with from other IDEs: syntax/reference highlighting and validation, code completion, source navigation, outline views and rename refactoring, find-replace (with regular expressions) and a workspace model of project and file management with full text searching (summarised in Figure 2). Please note that as some were also described during the previous reporting period (deliverable 1.2), we focus only on those that have been updated and any new features we may have introduced (marked with a red box in Figure 2). These are described in what follows.



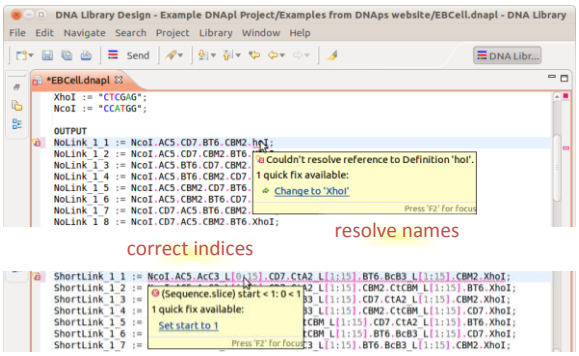
Library navigation



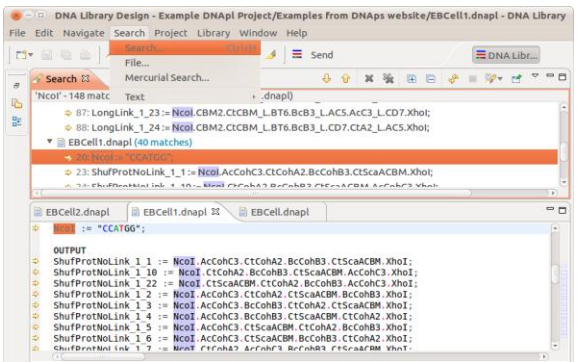
Reference highlighting (reuse)



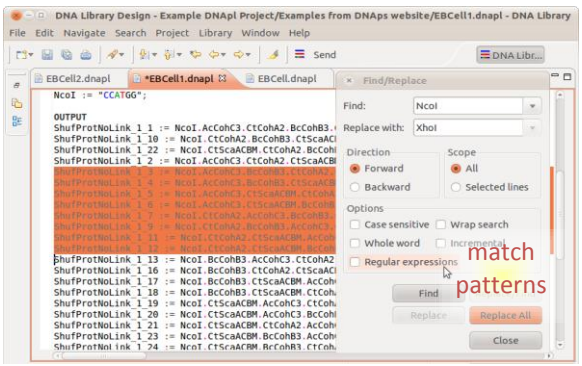
Validation



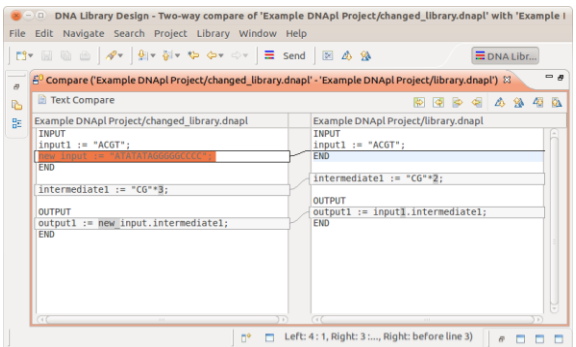
Quick fix suggestions



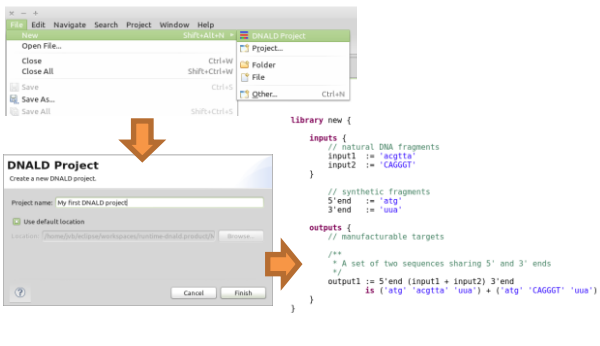
Searching across projects



Find and replace within file



Compare differences between files/versions



Wizard for getting started

Figure 2: Gallery of current DNA Library Designer features. Updates outlined in red.

Validation: Validation has been substantially improved. Syntactic validation (determined by the Xtext grammar used to generate the parser) has been customised to provide less generic and more informative error messages. For instance, the message “no viable alternative at input ’}’” is translated into “library must contain an outputs section” when that can be determined to be the cause. More inputs expressions are now valid, including subsequences and reverse complements of sequences and references to other inputs.

Semantic validation now occurs twice: once when the DNALD is parsed, based on values and relationships that are clear from the textual description (e.g. invalid nucleotide codes, duplicated codons or incomputable codon usages) and a second time when the DNALD file has been evaluated because more information is available at that point. This allows us to catch more subtle errors such as indices that are out-of-range for computed sequences, and report these as errors and warnings overlaid on the DNALD code. Cyclic dependencies between definitions prompt errors such as “Expression creates cyclic dependency: A -> B -> C -> A” for each definition in the cycle. All such definitions are subsequently ignored by the evaluator so that the remaining definitions may still be evaluated and validated. That is, we have introduced a kind of greedy partial evaluation that allows the combinatorial DNA library programmer to make some small errors while programming while still being able to see the partial results of his library design.

Assertions: We have extended the DNALD language as presented in the previous reporting period with the ability to write assertions that are also checked as part of the validation-evaluation-validation process described above, and reported in the GUI. Assertions are appended to definitions by the ‘is’ keyword and another DNALD expression that should evaluate to the same set of sequences as the definition it is bound to. False assertions raise the warning “Evaluation does not match assertion” on the defining expression, accompanied by an explanation of the difference between what was expected and the actual result. The availability of assertions serve three complementary purposes: (a) it enables the definition of complex libraries that have clear check-points for correctness, (b) it allows for the expression of complex libraries via two different mechanisms (the expression & the assertion) in such a way that one can disambiguate and clarify the other and (c) it helps teams of combinatorial DNA library programmers to communicate expectations about libraries outputs. In addition to using assertions for documentation, reference highlighting has been improved such that hovering over a name now shows a tooltip containing a /** documentation string */ (if present for that definition) with which designers can include comments about the various DNA parts in use in free language thus complementing the assertions.

Assistance: The New DNALD Project wizard is available from the File > New submenu. The resultant project contains a simple combinatorial DNA library that new users can easily adapt to bootstrap their own libraries. A brief DNALD tutorial is also available in an example project, and from the download page. It emphasises the ability of DNALD to create and work with sets over single sequences, by demonstrating the set operations: *union* (+), *intersection*, *difference* and *symmetric difference*, using assertions.

```
189         // order of set operations
190
191         union2  := concatenation + input1
192                 is input1 + concatenation // order is irrelevant
193                 is 'atgacgt' + 'acgt'
194
195         intersection    := union1 & union2
196                         is union2 & union1 // order is irrelevant
197                         is input1
198
199         symmetric_difference    := union1 ^ union2
200                                 is union2 ^ union1 // order is irrelevant
201                                 is concatenation + input2
202                                 is 'atgacgt' + 'atcgt'
203
204         difference1 := union1 - union2
205                     is union2 - union1 // order is relevant
206                     is input2
207                     is 'atcgt'
208
209         difference2 := union2 - union1
210                     is union1 - union2 // order is relevant
211                     is concatenation
212                     is 'atg' 'acgt'
213                     is 'atgacgt'
```

Figure 3: The new DNALD tutorial library, demonstrating operations on sets of sequences using assertions.

3. Visualisations

This section presents work that completes D1.3 but also establishes the basis for the deliverables D1.4 and D1.5 to be done during the *next* review period.

Per-sequence visualisation of DNA libraries was already integrated into DNA Library Designer as a view linked to the currently edited DNALD file. It was observed that as the complexity of the library definition increased, memory use became an issue and visualization became cluttered. This prompted us to improve on both the visualization and internal data structures so they could better handle more complex libraries. We have rewritten this visualisation to make it faster and extensible. Figure 4 shows the improved Library view visualisation. Each sequence is visualised a series of colour-coded blocks, where colour relates reused subsequences to their originating sequences and overlaid textual annotations, discernible when zoomed in, describe the exact start and end positions constituting the reused subsequence. The unit length can now be adjusted as a means of handling libraries with large variations in fragment sizes.

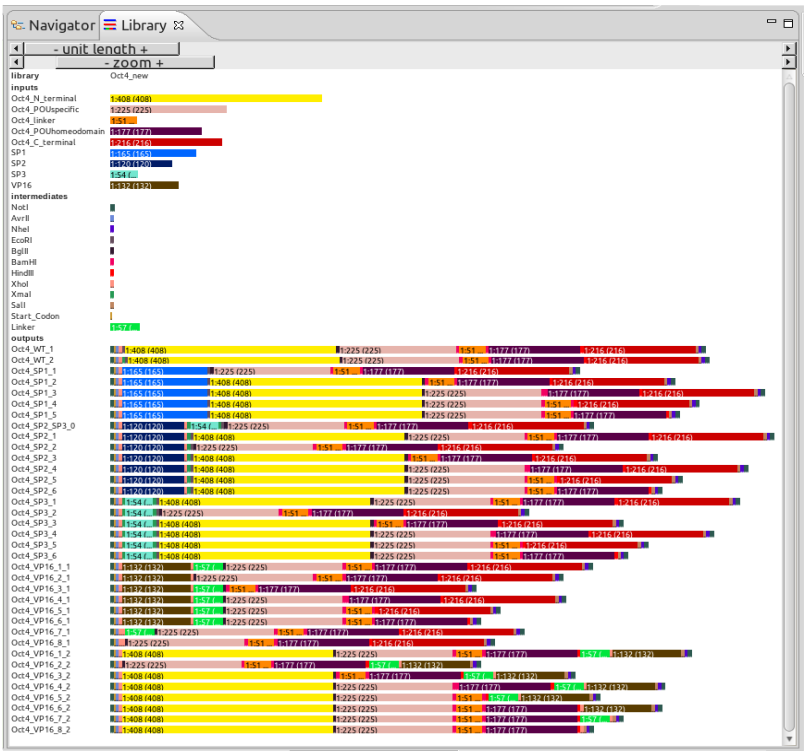
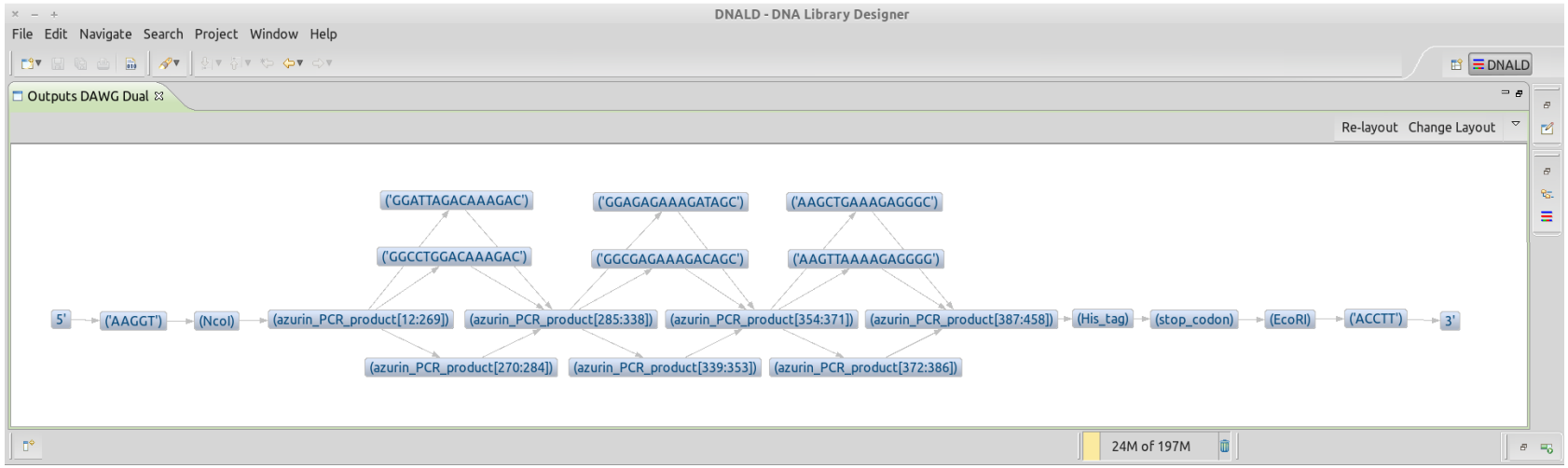


Figure 4: The Library View showing the inputs (natural fragments, top), short synthetic intermediates (middle) and 36 computed output sequences of the Oct4 library produced by UKB.

New visualisation and datastructure:

To better visualize the degrees of freedom that are available to a combinatorial DNA library programmer, we have developed a new data structure with an associated plugin that adds a new graph-based view, as shown in figure 5. At present this view renders the sequences of the library outputs as a graph of its consecutive subsequences. Each path from the 5' node to the 3' node corresponds to one output. Figure 5 demonstrates that there is equivalence between the DNALD expression defining the sequences contained within the graph and the structure of the graph itself. However, the graph is not derived from the parsed syntax tree of the DNALD expression but from the subsequences of the datamodel objects resulting from the library evaluation. Consequently, if the library design had described the same set of the sequences using the same subsequences but in 27 separate non-combinatorial definitions, as opposed to just 1 in the example, then the graph would be the same. In such cases this graph-based visualization highlights a potential refactoring that would improve scalability of the library design for additional combinations of any of the existing degrees of freedom.

A



B

C

```
azurin_Library :=
  'AAGGT'  NcoI
azurin_PCR_product[12:269]
(azurin_PCR_product[285:338] + 'GGATTAGACAAAGAC' + 'GGCCTGGACAAAGAC')
azurin_PCR_product[285:338]
(azurin_PCR_product[339:353] + 'GGAGAGAAAGATAGC' + 'GGCGAGAAAGACAGC')
azurin_PCR_product[354:371]
(azurin_PCR_product[387:458] + 'AAGCTGAAAGAGGGC' + 'AAGTTAAAAGAGGGG')
azurin_PCR_product[387:458]
His tag  stop codon  EcoRI  'ACCTT'
```

Figure 5: Rendering graph-based library visualisations in the GUI. Subfigure A shows the graph of all outputs in a DNA library. Subfigure B shows the same graph rendered using Graphviz based on the DOT language and layout program. DOT uses a sophisticated hierarchical graph layout algorithm to display highly connected directed graphs in a visually accessible manner. To replicate this high quality layout in the GUI we have utilised a contributed Sugiyama graph layout algorithm for the Zest toolkit on which the view is based. Subfigure C shows the DNALD expression defining the library (dependent definitions not shown). Because a single definition yields the total set of library outputs the structure of the computed graph corresponds exactly the structure of the expression.

The graph is computed from the evaluated DNALD library and is the dual of the minimal Directed Acyclic Word Graph (DAWG, described below and in figure 6) of the output sequences (words) where the letters are subsequences of references to DNALD definitions (typically, but not limited to existing inputs, elsewhere described as natural fragments). Where subsequences are shared between sequences and the prefixes/suffixes surrounding them do not preclude it, both sequences will share the subsequence node. DAWGs are typically used to store lexicons in spell checkers as they offer compact storage and faster retrieval for a given prefix [1]. Here they are repurposed to elucidate the relationships between arbitrary sets of sequences build from DNALD expressions.

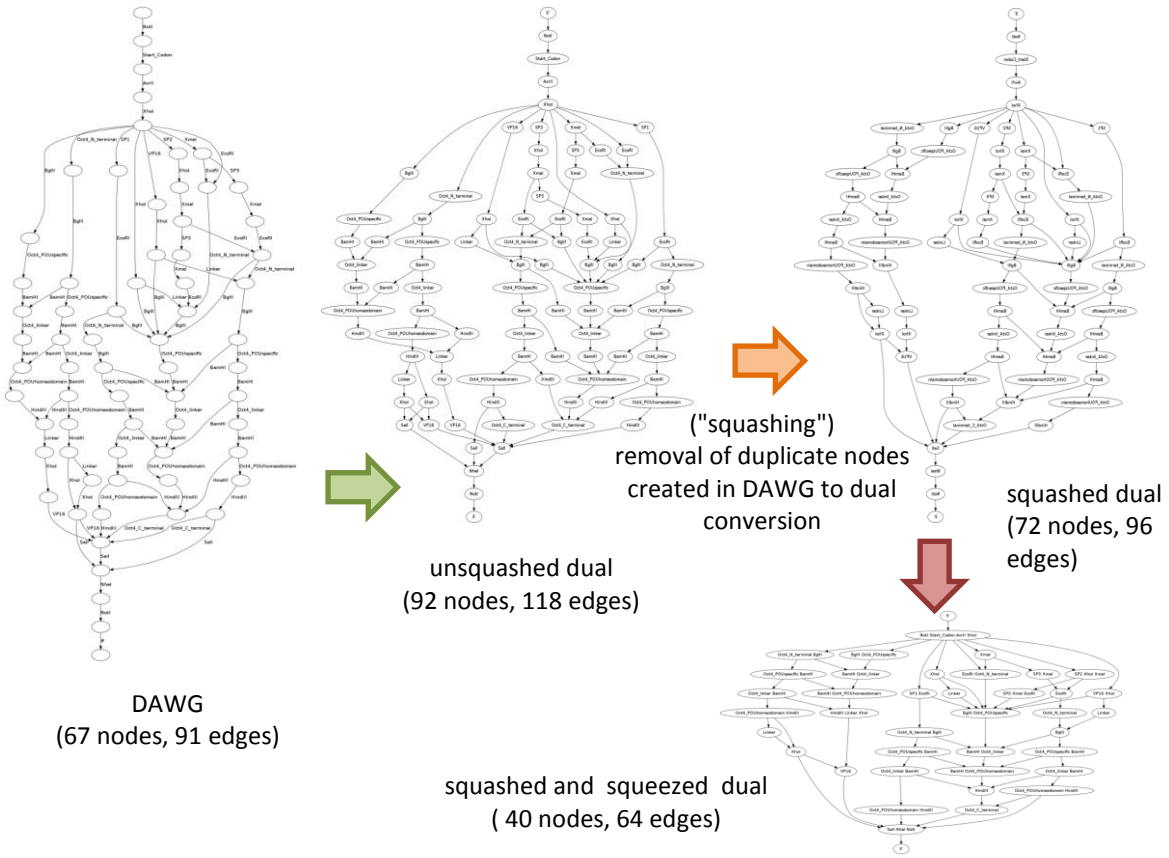


Figure 6: DAWG, unsquashed dual, squashed dual, and squashed and squeezed dual for the 36 outputs of UKB Oct4 library

Figure 6 outlines the process used to obtain the DAWG dual. We first compute the DAWG based on our implementation of a linear time minimality preserving word insertion algorithm [2] for minimal DAWGs (fewest possible nodes and edges). Because all of the sequences are inserted in this way the result is the minimal DAWG. The edges of the DAWG are labelled with a letter of the word being inserted. The unlabelled nodes serve as sources or targets for additional edges but hold no information themselves. Labelled edges and unlabelled are not well-suited for visual interpretation so we compute the “dual” graph where edges of the DAWG become labelled nodes and the correct connectivity is maintained by adding the appropriate directed edges (green arrow in figure 6). Depending on the library this process can add duplicate nodes and edges to the dual, as is seen in the “unsquashed” dual in figure 6 (it does not happen for the azurin_library derived dual shown in figure 5). We use the term “squashing” to describe the sub-algorithm for removing the duplicate nodes (orange arrow in figure 6) because when viewed left-to-right as in the GUI, the height of the graph decreases. We confirm that the sequences contained in the squashed dual are the same as those used to create the initial DAWG, and it is rendered in the GUI as the graph-based visualisation using the Zest Eclipse plugin.

Note that each node of the squashed DAWG dual contains a subsequence object containing a single reference object. The graph can be further reduced by “squeezing” (red arrow in figure 6) any consecutive non-branching nodes into a single node with a subsequence object containing the references of each squeezed node in order. Squeezing produces a smaller graph and therefore a simpler visualization of the same data.

Further applications of the datastructure:

The correspondence between the DAWG dual and combinatorial DNALD expressions makes it an ideal basis for a visual programming interface. The nodes and edges of the graph provide potential interaction surfaces, such that edges could be deleted to remove certain combinations, clicking an edge could insert a new node between two existing, nodes could be rearranged or their contents changed.

For the planning of library construction in WP2, a minimal graph with nodes corresponding to natural or synthetic fragments, derived from a human optimized combinatorial design constitutes a significant improvement on even our previous origin-preserving data model, as the edges of the graph can be considered potential Y operations affecting a subset of target sequences.

An operation analogous to squeezing can be applied to an actual DAWG to yield a compressed DAWG, which is the most memory efficient data structure for holding a set of strings in memory without additional data compression techniques. Although this offers a possible optimisation for our system it cannot be exploited as we would lose important information as to the origin of subsequences, even with a backing array accessed through perfect hashing [3].

Lastly, our DAWG and dual implementations are generic, meaning that the type of the object used as edges/nodes, in the DAWG/dual respectively, can vary but must be specified for the Java compiler. This allows us to create duals of strings as easily as with subsequence objects, i.e. with unstructured DNA sequences, a squeezed graph of the “bases” that can be leveraged for the final reverse parsing task of WP1. Figure 7 shows how such a graph reveals subsequence structures comparable to that extracted from a library design.

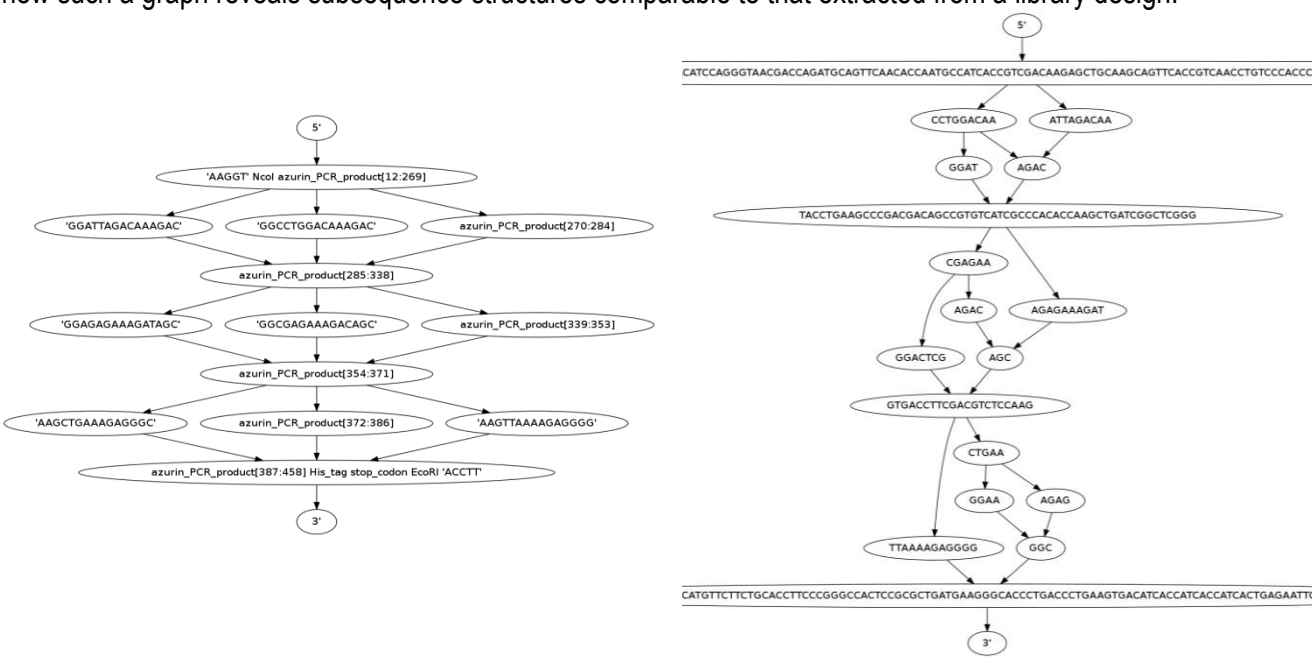


Figure 7: Comparison of the figure 5 azurin_library squeezed DAWG dual (left) and the squeezed “bases” DAWG dual of the same DNA sequences (right). There is a correspondence between the four longer subsequences in right graph to the constant portions of the left graph, and similarly the intervening variable sections.

4. Conclusions

This report has outlined the following contributions: improvements to reliability and scalability of our software and user's designs through software engineering, validation and inline testing with assertions; improvements to our existing visualisation and the development of a new graph-based visualisation. The graph-based aspects of this work can be applied more broadly to other objectives in this work package and others.

The graph for the set of sequences defined by a single combinatorial expression is the correct level of abstract on which to build the vDNALD visual programming interface, the next task of WP1. The squeezed graph of an arbitrary set of DNA sequences can reveal structural similarities that can be exploited for the reverse parsing task of WP1, and for the improved planning of synthetic sequences in library construction.

The graph of *all* library outputs, while sometimes difficult to comprehend visually (as in figure 6), has proved a useful optimisation tool for the planning of library construction (described in detail in deliverable 2.4), which realises an internal goal to exploiting information present in the DNA library design to simplify the planning stages of library construction and improve DNA reuse therein. With sufficient attention to scalability it may even be possible to compute a viable construction plan as part of the library evaluation and thereby incorporate into the GUI a cost assessment that could guide users when designing libraries.

5. References

1. Appel, A. W. & Jacobson, G. J. The world's fastest Scrabble program. Communications of the ACM 31, 572–578 (1988).

2. K.N. Sgarbas et al. Optimal insertion in deterministic DAWGs. Theoretical Computer Science 301 (2003) 103-117.

3. Lucchesi, C. L. & Kowaltowski, T. Applications of Finite Automata Representing Large Vocabularies. Technical Report, Department of Computer Science, University of Campinas, SP, Brazil (1992).

6. Abbreviations

List all abbreviations used in the document arranged alphabetically.

DAWG	Directed Acyclic Word Graph
DNALD	DNA Library Design language
GUI	Graphical User Interface
IDE	Integrated Development Environment
WP	Work Package