Deliverable D1.5

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: D1.5

Deliverable name: Operational SW and user friendly GUI for specifying complex DNA libraries for the editing processes

Contractual Date[1] of Delivery to the CEC: M36
Actual Date of Delivery to the CEC:
Author(s)[2]: Prof. N. Krasnogor
Participant(s)[3]: UNEW
Work Package: WP1
Security[4]: Pub
Nature[5]: R
Version[6]: 0.1
Total number of pages: 14

[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. Collectively, this implements an operational software platform, including user friendly GUI, for specifying complex DNA libraries for the editing processes.

In this reporting period we provide a summary detailing the functionality of the entire software suite we have developed for this project. We also described a new algorithm that simultaneously tackles the specific task for this reporting period, namely, the reverse parsing of unstructured sequences into DNALD libraries as well as contributes to deriving library construction plans for DNALD library designs (WP2)

Keywords[7]**:**

DNA Library Designer IDE, GUI, DNALD, Reverse Parsing & Planning

Introduction

a. Aim / Objectives

DNA programming is the DNA-counterpart of computer programming, that is, the basic computer programming cycle is to modify an existing program, test the modified program, and iterate until the desired behavior is obtained. Similarly, the DNA programming loop is to modify a DNA molecule, test its resulting behavior, and iterate until the goal (which is either understanding the behaviour or improving it) is achieved. One key difference between the two is that unlike computer programming, our understanding of DNA as programming language is very far from being perfect, and therefore trial and error are the norm rather than the exception in DNA-based research and development. Therefore, DNA programming is more efficient if multiple variants of a DNA program, also called a DNA library, are created and tested in parallel, rather then creating and testing just one program at a time. Hence the basic DNA programming cycle takes the best DNA programs from the previous cycle, uses them as a basis for creating a new set of DNA programs, tests them, and iterates until the goal is achieved, ideally, maximising reuse of DNA strands and library designs. The goal of WP1 is thus to create a fully functional combinatorial DNA library editor with possibility of strands reuse. The objectives are:

O1.1. Developing a textual DNA programming language (DNALD)
O1.2. Developing an open source graphical user interface (GUI)

In this deliverable Dr. Jonathan Blake worked until the 1s/November/2013, while Dr. Pawel Widera worked since then for the duration of the reporting period. Prof. Krasnogor worked throughout the reporting period. Please note that UNEW has not received payment from Nottingham university since September 2013.

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

This deliverable tackles O1.2, in particular Tasks 1.3 & Tasks 1.4, and successfully reaches milestone MS2 thus it should be considered successfully completed.

b. State of the Art

Certain tools for editing and manipulating DNA strings were developed long ago. In recent years a new system called GENOCAD[1] was developed, offering a GUI based on context-free grammars, to design DNA molecules based on known building blocks. High-end commercial packages, e.g. DNA2.0[2], Accelerys[3], DNASTAR[4], etc provide advanced features for designing cloning sequences and plasmids. These features include sophisticated GUI, project management, codon-usage optimizations, restriction site handling, etc. However, similarly to GENOCAD, they are intrinsically geared towards building high-level structured entities (e.g. plasmids) rather than multiple, combinatorially dependent DNA sequences, which are more likely to advance R&D and are the focus of this WP. In addition, these tools are limited in their expressive power – with most of them it is impossible to formally define degrees of freedom (for example using amino acids notations together with a custom defined codon table) which can very well ease the actual construction of the desired constructs. Existing tools are also not built for large libraries specifications, where some shared fragments can be defined and reused in several molecules, allowing fast generation of the specification.

c. Innovation

This deliverable is built on solid formal language theory, state-of-the-art software engineering and algorithms. We created a software suite that integrates our DNA library specification language (DNALD) into an integrated software development environment based on ECLIPSE. The specification of DNA libraries in the DNALD IDE is supported by a concise and powerful language (delivered in reporting period 2) with a state-of-the-art graphical user interface that is friendly, fast, robust and portable and that has been tested in Windows, Mac and Linux environments. Furthermore, our software enables extensive manipulation of a DNA molecules' representations while maximizing the use of existing DNA and minimizing the need for synthesizing new DNA, and permits the user to estimate a reverse parsing-planning route to manufacturing the libraries by means of a new and, currently, the fastest algorithm for DNA library design.

2. Implementation

In this report we will summarise the underling software engineering strategy used for building our software suite and its key functionalities. We will also report on the new reverse parsing and planning algorithm produced. We remark that, as this is the FINAL deliverable for WP1 and it collects the work done on previous reporting periods into a comprehensive software suite, by necessity, some parts of this report are carried from the report in the previous reporting period.

Overview

The software suite we have developed is composed of two main components, the DNALD integrated development environment (IDE) and the reverse-parsing & planer algorithm with its associated webserver.

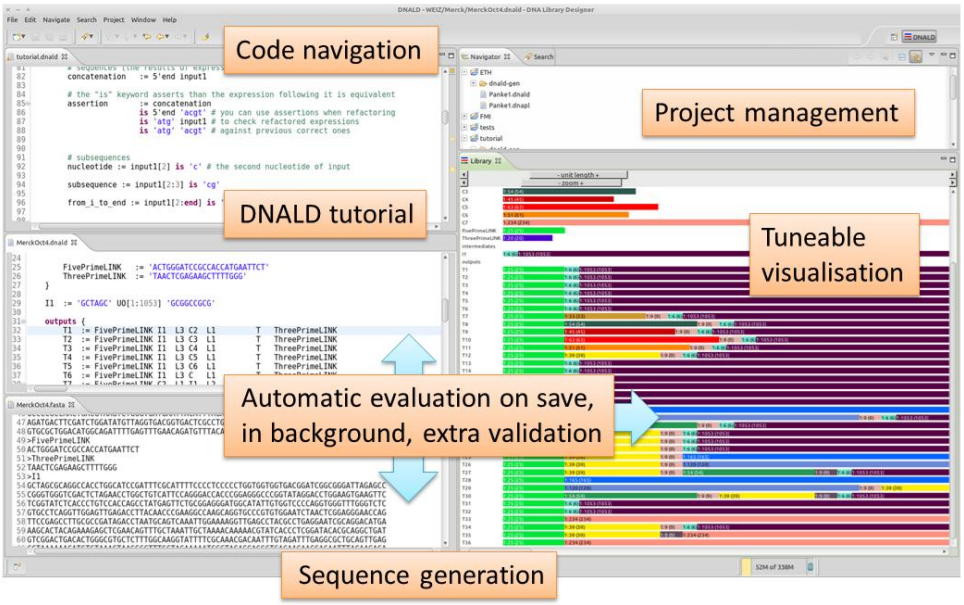


Figure 1: Overview screenshot of DNALD IDE

The DNALD IDE (Fig. 1) and its internal data structures were completed between the end of previous reporting period and the middle of the current one. In particular, the first half of this reporting period saw the debugging of the IDE based on feedback produce by partners in WP5. They utilised the software to design their combinatorial DNA libraries and provided feedback on its functionality and performance. This feedback was used to debug our software and led to the current stable version (see availability section).

Moreover, our work on refactoring our datamodel also helped us to simplify the reverse parsing and planning problems and deliver a simple unified solution for both via a single webserver (Fig. 2), the main new contribution of this reporting period. More specifically, DNA library assembly was defined as the S-T library assembly problem and the results of our new library assembly algorithm on WP5 libraries were presented. In all but one case it matched or surpassed the current state-of-the-art algorithm for planning library assembly, with a minimum speedup of 70X and maximum 0.027 second running time. The library on which the new algorithm achieved the greatest savings in planned concatenation operations (285 vs. 527 on average compared to the state-of-the-art) and running time (4161X speedup) happened to be the UH HRet library which was designed using a prototype implementation of an incomplete factorial sampling method. The efficiency improvement provided by our algorithm enables library assembly plans to be computed as fast as the design can change.

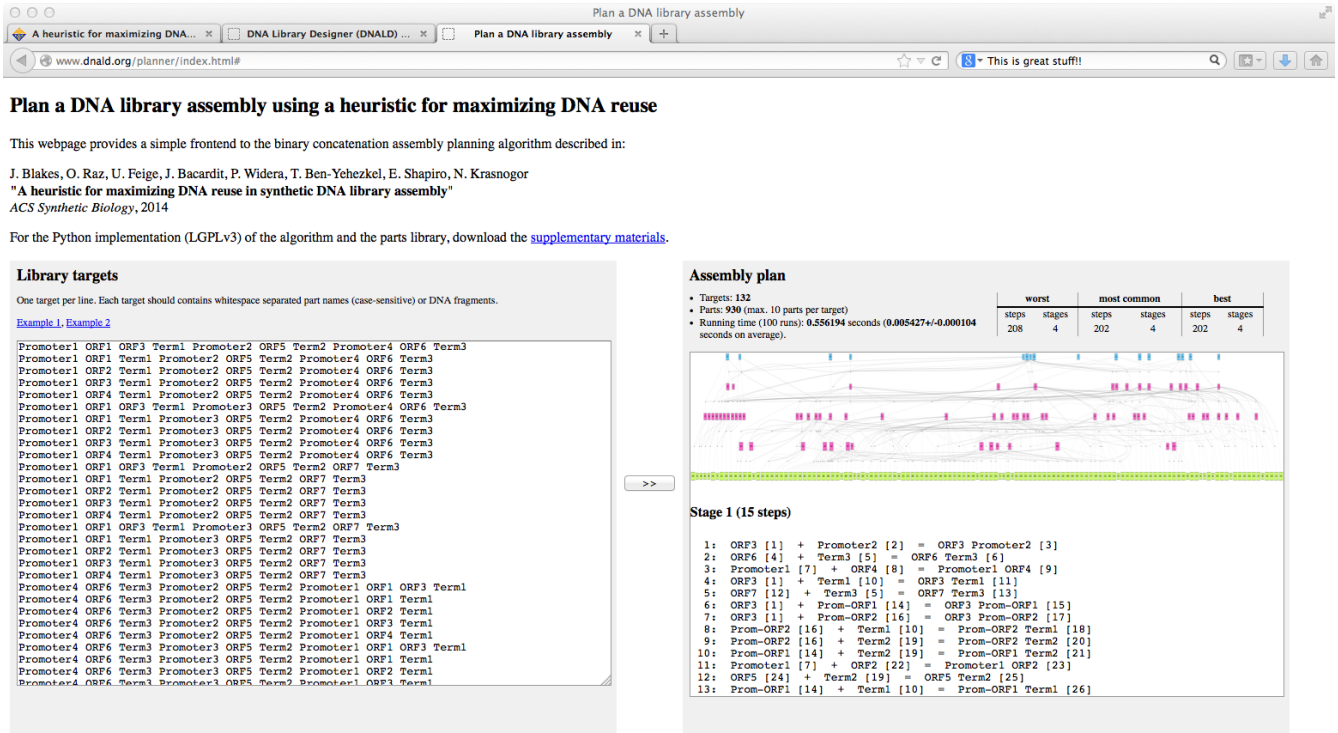


Figure 2: Screenshot of the reverse parsing and planning algorithm webserver

For completion purposes we summarise next both components of our software suite, namely, the DNALD IDE and the reverse parsing/planning algorithm.

Software Engineering

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 Fig. 3 that can be installed alongside other plugins in an existing Eclipse installation or used as a standalone product on Linux, Mac and Windows platforms.

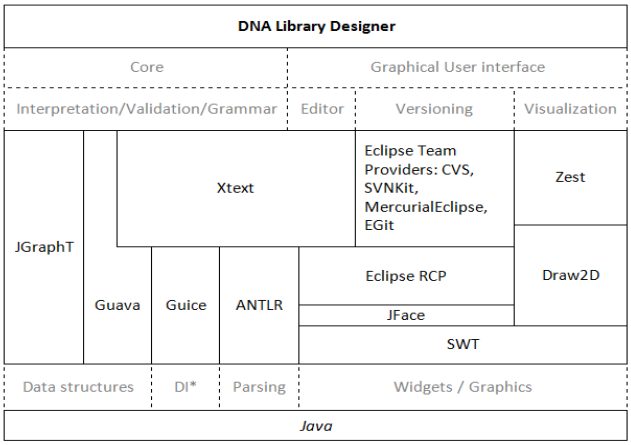


Figure 3: Software architecture of the DNALD IDEA. the diagram is a "stack" in which each layer above is dependent on functionality provided by the layer below.

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 and the previous 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.

Availability

The entire software suite is available for download and installation at:

http://www.dnald.org/

The user friendly GUI (IDE) for DNALD has versions for Windows (XP/Vista/7) 32 & 64 bits, Mac OSX (Cocoa) 32 & 64 bits and Linux (GTK) 32 & 64 bits are available.

The reverse parsing and planning algorithm is downloadable as a python script.

The reverse parsing and planning algorithm is also available as a webserver online at:

http://www.dnald.org/planner/index.html

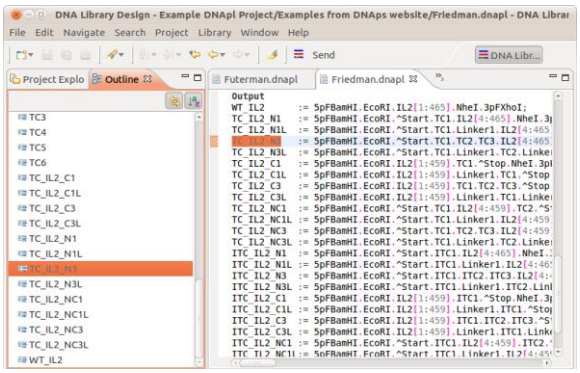
3. Results

We report our results in two sections, namely, DNALD GUI and Reverse Planing & Parsing Webserver.

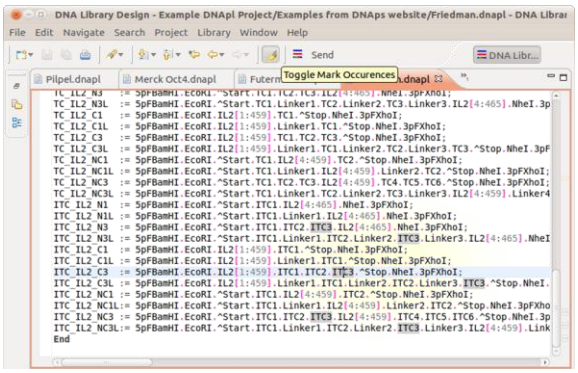
DNALD Friendly Integrated Development Environment (GUI)

Our software 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 4). These features makes our IDE user friendly. In what follows we list the key user-friendly features of our integrated development environment for specifying combinatorial DNA libraries:

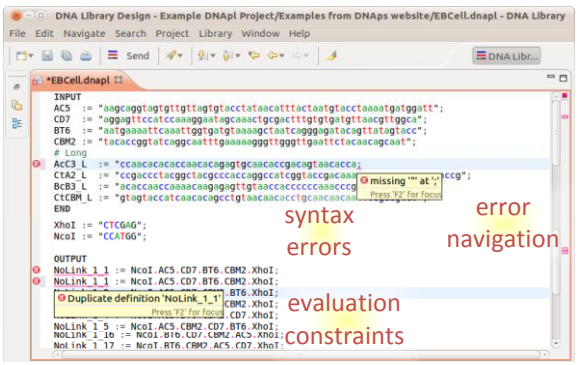
- The IDE includes ***syntactic validation*** (determined by the Xtext grammar used to generate the parser) and informative error messages. For instance, the message "no viable alternative at input '}'" is translated into "library must contain an outputs section" when that can be determined to be the 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.
- As part of the validation-evaluation-validation process described above ***assertions*** have been included into the DNALD language. 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.
- The 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 (Fig. 5) listing the entire API (also reported in D1.4) 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.



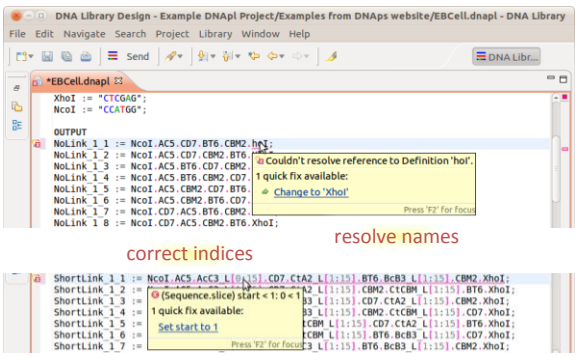
Library navigation



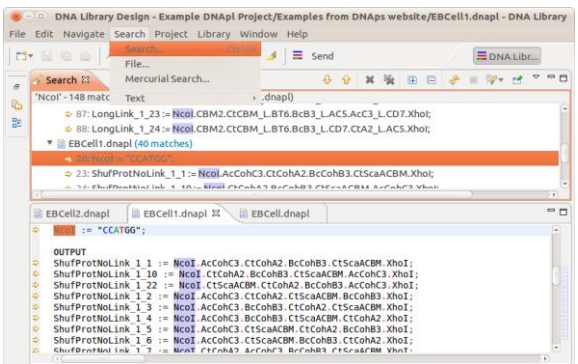
Reference highlighting (reuse)



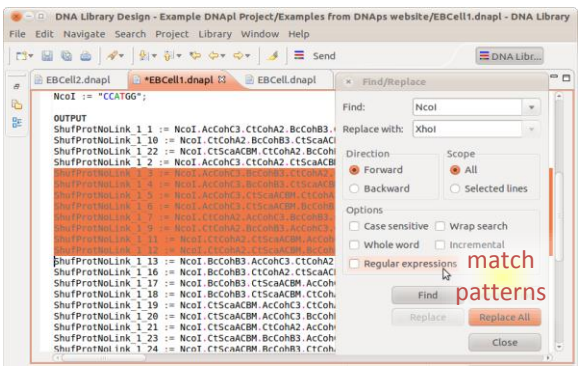
Validation



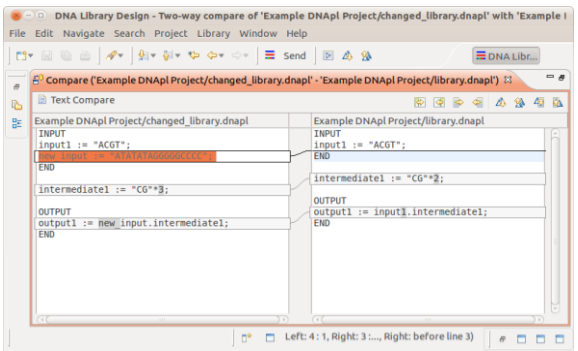
Quick fix suggestions



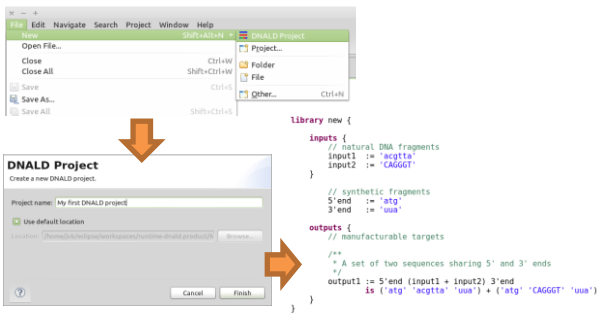
Searching across projects



Find and replace within file



Compare differences between files/versions



Wizard for getting started

Figure 4: GUI's snapshots depicting some of the key functionalities of DNALD's IDE

```
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 5: DNALD tutorial demonstrating how to use the DNALD language API

- Figure 6a shows DNALD's Library ***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 be adjusted as a means of handling libraries with large variations in fragment sizes. 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 6b. 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 an equivalence between the DNALD expression defining the sequences contained within the graph and the structure of the graph itself.

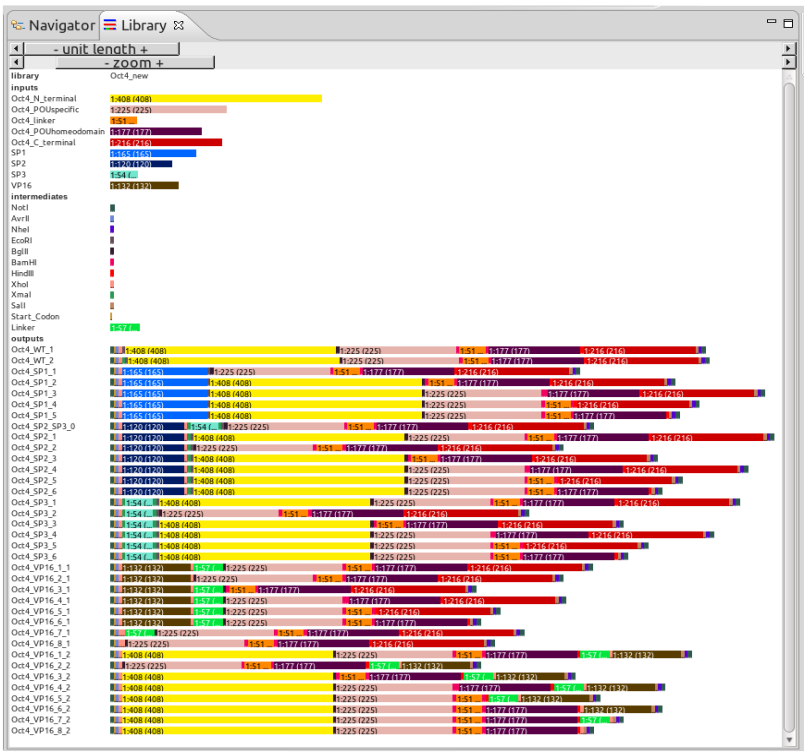
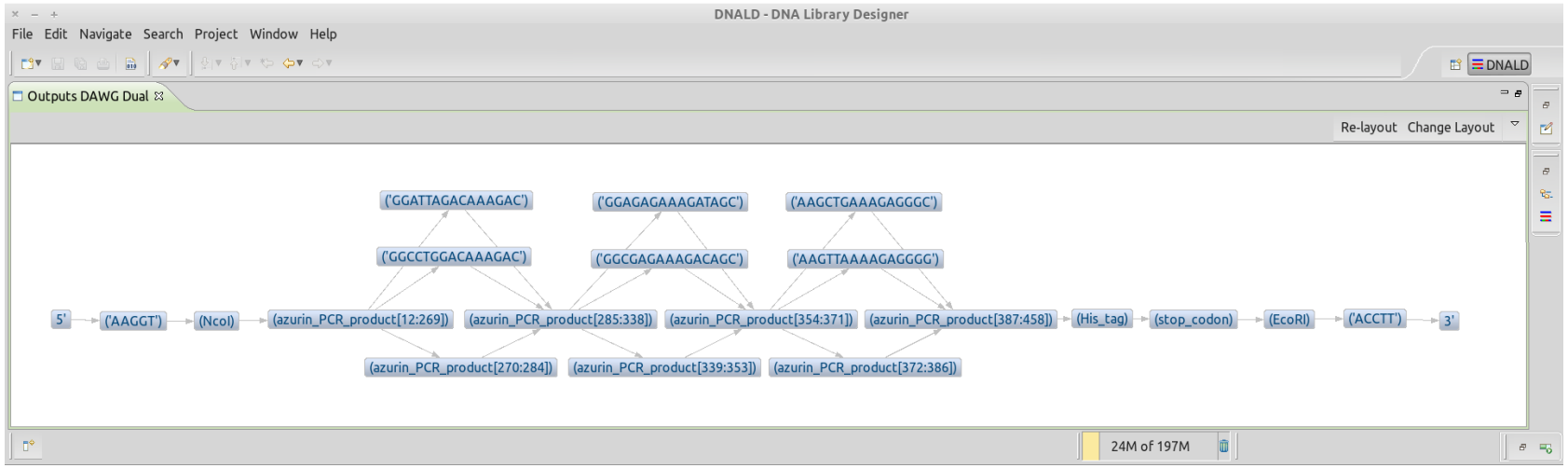


Figure 6a: Graphical view of combinatorial DNA library showing inputs (natural fragments, top), short synthetic intermediates (middle) and 36 computed output sequences for the Oct4 library designed by partner UKB (in WP5)

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 6b: 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

Reverse Parsing and Planning Webserver

When analyzing the problem of reverse parsing, namely, given a set of output sequences generate a plan that produces them using the binary Y operation, we realized that its solution also leads to an optimal planning graph that could be used in WP2. We created a novel greedy algorithm for computing near-optimal DNA assembly graphs and show empirically that it runs in linear time, enabling almost instantaneous planning of DNA library sizes exceeding the capacity of today's biochemical assembly methods. We compared assembly graph quality and algorithmic performance to the results obtained in [5] –the previous state of the art- demonstrating that they are significantly faster to obtain and equivalent to the best results for DNA library assembly with intermediate reuse found in the literature.

The problem we tackled can be specified formally as:

Let S be a set of available DNA parts and T a set whose elements are ordered sequences of elements in S which constitute a DNA library. Define a library assembly graph (as in Figure 1) as a hierarchical clustering, representing the binary convergence (Y operations) of nodes in S to nodes in T.

Given an assembly graph quality/cost function (stages,steps) that describes the ratio in penalties between the number of stages (depth of the construction graph) and number of intermediate steps (nodes that are neither in S nor in T), the problem is finding the assembly graph with optimal quality/minimal cost. In what follows we use the same cost function as [5] which seeks to minimize the number of stages and then the number of steps.

For S={A,B,C,D,E} and T={ABE,ABDE,ACDE,ADE} Figure 7 shows a suboptimal (2,8) and the optimal (2,7) assemblies.

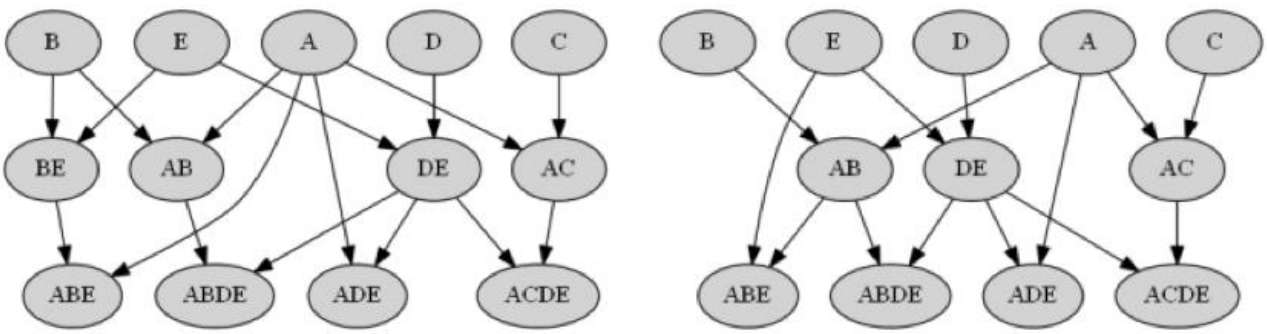


Figure 7: Suboptimal (left) and optimal (right) library reverse parsing assembly plan

Sequences of the DNA library to assemble are encoded in terms of their constituent subparts, i.e. as a list of s-tuples (or any hashable ordered sequence) of p-tuples of hashable objects representing subparts. Figure 8 shows the state of the four sequence example library as it evolves under the optimal assembly plan shown in Figure 7 (sequences are shown as hashable Python objects conforming to this library encoding).

Initial input	After stage 1	After stage 2
library=[((A,),(B,),(E,)), ((A,),(B,),(D,),(E,)), ((A,),(C,),(D,),(E,)), ((A,),(D,),(E,))]	library=[((A,B),(E,)), ((A,B),(D,E)), ((A,C),(D,E)), ((A,),(D,E))]	library=[((A,B,E)), ((A,B,D,E)), ((A,C,D,E)), ((A,D,E))]
steps=[((A,),(B,)), ((D,),(E,)), ((A,),(C,))]	steps=[((A,B),(E,)), ((A,B),(D,E)), ((A,C),(D,E)), ((A,),(D,E))]	steps=[]

Figure 8: Library state and resulting steps while running our algorithm

For each stage, maps from pairs and triples to multisets, counting the number of times they occur in a sequence, are computed and from these, multimaps of pairs to containing triples and triples to contained pairs, are derived.

Using the per-pair scoring function, a list of (pair, score) items is computed, shuffled (to randomize the order of equal scoring pairs in a stable sort - allowing multiple runs with different resultant assembly graphs) and sorted by descending score.

For each (pair, score) a pair is added to a list of excluded pairs if it has a score ≤ 0 (allowing custom scoring functions to exclude certain pairs by design) or, if not already excluded, it is added to the set of steps for this stage and all other pairs that are in a triple with the current pair are excluded instead. If no steps were added in this stage then the list of stages, containing the lists of steps for each stage, is returned.

Lastly, each of the highest scoring pairs that are not excluded by a higher scoring pair are then concatenated by seeking for occurrences in the sequences of the multiset mapped to that pair and moving elements of the right p_r-tuple into the left p_l-tuple, leaving a pair of (p_l + p_r)- and 0-tuples, so as to not create new occurrences of any pairs.

Once all possible concatenations have been applied, all 0-tuples are removed and the algorithm loops.

Evaluation

We compare the performance of our algorithm to that of A1 in terms of assembly graph quality (smallest number of stages then steps, Table 1) and running time (Table 2) using the one synthetic and two real world libraries of [1]: a small dataset for exhaustive search, ‘phagemid’ and ‘iGEM-2008’ (containing 2 duplicates). In the table 1 & 2 A1 (Densmore et al.’s algorithm) values were estimated from [5] Figure 7.

Library |size| / Algorithm	Exhaustive |5|	Phagemid |131|	iGEM-2008 |395|
A_1	(3,11)	(4,202*)	(5, 808*)
A_{new}	(3,11) best (3,15) worst	(4,202) or (4,208) †	(5,808) best (6,841) worst

Table 1: Quality (#stages, #steps) of reverse planing assembly graphs

Table 1 shows that our algorithm Anew obtains the optimal solutions found by A1. Therefore we conclude that this greedy approach can optimally assemble real-world DNA libraries.

Library size / Algorithm	50	100	150	200	250	300	350	395
A_1 ‡	0.1	1	45	120	290	405	620	720
A_{new} mean of 10 runs	.007 $\pm 10^{-5}$	.014 $\pm 10^{-4}$	.020 $\pm 10^{-5}$	.027 $\pm 10^{-4}$	.033 $\pm 10^{-4}$	.039 $\pm 10^{-4}$	.045 $\pm 10^{-4}$	.053 $\pm 10^{-4}$

Table 2: Running time in seconds for subsets of iGEM 2008

Table 3 shows the running times of Anew were at least two orders of magnitude faster than A1 and imply that Anew scales linearly with the number of sequences. The implementation language, runtime and hardware these timings were obtained with may differ; we used Python running on a single i7-2670 2.2Mhz core.

Our new algorithm is provided both as a downloadable (www.dnald.org/downloads) stand alone application (its API is described in D1.4) and as a user friendly webserver (www.dnald.org/planner) in which the user can input a library made of parts and the reverse planning algorithm will calculate the optimal assembly tree. The webserver (see Fig. 1) also provides two examples that serve to illustrate the syntax of the input files.

4. Conclusions

Our goal for this final deliverable for WP1 was to provide a multiplatform user friendly GUI, i.e. an integrated development environment, for the DNALD language and subsidiary algorithms including reverse parsing planning methods.

The key contributions of this reporting period has been the debugging of the DNALD IDE and the completion of the subsidiary data structures that were required for the reverse parsing and planning algorithm plus the reverse planning algorithm itself.

We have successfully achieved this by providing a usable software system that operated for Windows, Mac and Linux platforms in 32 and 64 bits versions as well as a user friendly webserver for the reverse planning algorithm.

Both the DNALD IDE and the reverse planning algorithm and associated webserver advances the state of the art beyond current standards. That is, our DNALD IDE is the only software currently available that facilitates the design and formal specification of *combinatorial* DNA libraries *with* reuse, while our reverse planning algorithm is the fastest algorithm that can assemble *vast* combinatorial libraries.

The reverse planning algorithm has been published in [6,7]

5. References

[1] Cai, Y., et al.,A syntactic model to design and verify synthetic genetic constructs derived from standard biological parts. Bioinformatics 23:20,2007

[2] https://www.dna20.com

[3] ttp://accelrys.com/solutions/science/biosciences/

[4] http://www.dnastar.com

[5] Densmore, D., Hsiau, T. H.-C., Kittleson, J. T., DeLoache, W., Batten, C. and Anderson, J. C. 2010. Algorithms for automated DNA assembly. Nucleic Acids Research 38, 8 (Mar. 2010), 2607-2616. DOI=http://dx.doi.org/10.1093/nar/gkq165

[6] Blakes J., Raz O., Krasnogor N., Shapiro U., A New Algorithm for Combinatorial DNA Library Assembly, Proceedings of the 5th International Workshop on Bio-Design Automation (IWBDA 2013), 2013.

[7] Blakes J., Raz O., Feige U., Bacardit J., Widera P., Ben-Yehezke; T., Shapiro E., Krasnogor N.,A heuristic for maximizing DNA reuse in synthetic DNA library assembly, ACS Synthetic Biology (accepted), DOI: 10.1021/sb400161v, Publication Date (Web): 20 Feb 2014

6. Abbreviations

List all abbreviations used in the document arranged alphabetically.

DNALD	DNA Library Design
IDE	Integrated Development Environment
GUI	Graphical User Interface