DESIGN REPOSITORY & ANALOGY COMPUTATION VIA UNIT-LANGUAGE ANALYSIS (DRACULA) MATCHING ALGORITHM DEVELOPMENT

Briana, Lucero (1); Julie, Linsey (2); Turner, Cameron (1)
1: Colorado School of Mines Golden, United States of America; 2: Georgia Institute of Technology Atlanta, United States of America

Abstract

Analogical reasoning is not the only approach for achieving innovation, but it is a highly effective and noted method. To avoid relying on chance identification of analogies through unique individual knowledge and experience, this research considers the potential impact of a system where an engineer could begin with their existing design, specify to a computer a set of critical functions and desired design performance improvements, and be returned with design analogies intended to inspire avenues for design improvements. This system, called Design Analogy Performance Parameter System (D-APPS), implements performance metric matching to retrieve examples and stimulate analogical mapping, enabling innovative design advances.
The D-APPS software, Design Repository & Analogy Computation via Unit-Language Analysis (DRACULA), produces matches using topological isomorphism. The user provides information that derives a graph using the DRACULA algorithm to locate a matching graph set in an analogy space of the critical pairs and critical chains. DRACULA receives input from the user as a single flow associated with an engineering parameter, and a sequential set of critical functions.

Keywords: Bio-inspired design, Design engineering, Conceptual design

Contact:
Dr. Cameron Turner
Colorado School of Mines
Mechanical Engineering
United States of America
cturner@mines.edu

Please cite this paper as:
Surnames, Initials: *Title of paper*. In: Proceedings of the 20th International Conference on Engineering Design (ICED15), Vol. nn: Title of Volume, Milan, Italy, 27.-30.07.2015

1 INTRODUCTION

The engineering design process is an iterative process wherein customer needs are met via the form and function of the final product. The design process for engineers revolves around requirements and the levels of performance of these requirements. The requirements, at least in engineering terms, can be qualified as engineering performance parameters such as *cost, efficiency, time,* and *weight*. Each of these engineering performance parameters can be mathematically modelled as a design variable.

1.1 Design Analogy Performance Parameter System (D-APPS)

Through Functional Modelling (Pahl & Beitz, 1961) the function(s) of a product can be iteratively identified and can then be correlated with a quantifiable engineering performance metric Computational tools for aiding design-by-analogy have so far focused on function-flow and keyword-based retrieval of analogues. Because of the critical role of performance and benchmarking in design, the need for performance metrics-driven analogy retrieval is currently unmet.

This research defines the basis for a new quantitative approach for retrieving useful analogies for innovation based on the relevant performance characteristics of functions. The concept of critical functionality is the idea of identifying only a certain set of pertinent design functions which define the functionality of the product. A critical function (CF) is a function within a functional model whose performance directly relates to a Key Performance Parameter (KPP) of the system as a whole. Critical flows (CFl) are the flows within a functional model which correlate to the fulfilment of the KPPs. These critical functions and flows will enable multiple analogies to be presented to a designer by recognizing similar functionality across distant design domains and incorporating key performance criteria. The ultimate focus of this research project is to create a performance-metric-based analogy library, called the Design Analogy Performance Parameter System (DAPPS). By focusing on a select set of "critical" functions and engineering performance parameters, more design domains can be included in the database facilitating analogy retrieval.

Using Graph Theory, the D-APPS program takes the initial design problem with the parameters to be optimized as the model output dimensionality. The critical flows are matched with the critical functions as defined by the engineering user. The non-dimensional numbers defining the engineering performance metric to be optimized are associated with the flows of each database entry. The critical flows matching the units are then associated with the critical functions and passed back to the user as feasible solutions via isomorphic matching.

2 GRAPH THEORY

Euler was the originator of graph theory with the seven bridges of Konigsberg problem. In this problem, four land masses are connected by seven bridges (Euler, 1741). The problem proposed that the four land masses could be traversed by crossing each bridge only once. Euler represented the problem as a graph where the land masses became the nodes (vertices) and the bridges became the lines (edges).

Graph Theory (GT) is the organized mathematical representation of at least two data sets in a design space. A set is "a collection of distinct objects"; where an empty set containing no elements is known as a null set. Dots and lines are the premise behind a graph. Mathematically, the graph G is composed of a vertex set (dots) and an edge set (lines). Vertices are a finite non-empty set of objects V; edges are 2-element subsets of vertices E. A graph G, is then made up of edge sets E, and vertices sets V, such that $G(E,V)$ (Trudeau, 2013), (Gould, 2012), (Chartrand, 1985), (Chartrand & Zhang, 2012).

3 DRACULA ALGORITHM

D-APPS research unites the pattern matching of graph theory of design performance parameters and functionality to recover and offer additional design analogies. The DRACULA application utilizes graph theory as the basis for analogy matching. The nodes of the graph are the bond graph categories and are connected by edges corresponding to the engineering performance parameters. These edges of the graph are able to connect to any of the bond graph nodes within the groupings of the Functional

Basis flow categories (Hirtz, 2012). Thus, the graph G is composed of two data sets, the bond graph categories V and the engineering performance parameters E.

Graph Theory is the data representation method used by D-APPS. The edges of the algorithm are the set of non-dimensionalized units of the engineering parameters. The edge set for the D-APPS program is therefore an n x 10 vector of engineering parameters. The unit vector data values are seen in Table 1. The n of the vector is not set in stone as the engineering unit parameter list is expected to grow with added maturity of the both the repository and the algorithm.

Table 1. Dimensional Analysis Theorem (DAT) variables used in categorizing functional basis flows.

Unit	Mass	Length	Time	Electric charge	Absolute temperature	Amount of substance	Luminous intensity	Cost	Degree
Symbol	M	L	T	Q	Θ	N	J	$	°

The vertices of the algorithm are the bond graph types associated with the functions of the functional basis. The vertices therefore are a 28 x 6 vector with the categories seen in Table 2. All the functions are related by the bond graph functions. The bond graph types with similar functionality are returned as potential analogy options to the user.

Table 2. DRACULA Bond Graph/Function correlations for identification in the repository and algorithm.

	Bond Graph Component Type				
	1	2	3	4	5
Function	Resistor	Capacitor	Inertial	Transformer	Gyrator
Actuate	0	0	0	0	1
Branch	1	0	0	0	0
Change	1	0	0	1	0
Channel	1	0	1	0	0
Connect	1	0	0	0	0
Control Magnitude	1	0	0	1	1
Convert	0	0	0	1	1
Couple	1	0	0	0	0
Distribute	1	0	0	0	0
Export	1	0	1	0	0
Guide	1	0	1	0	0
Import	1	0	1	0	0
Indicate	1	0	1	0	0
Mix	1	0	0	0	0
Position	1	0	1	0	0
Process	1	0	1	0	0
Provision	0	1	1	0	0
Regulate	1	0	0	0	0
Secure	1	0	0	0	0
Sense	1	0	1	0	0
Separate	1	0	0	0	0
Signal	1	0	1	0	0
Stabilize	1	0	1	0	0
Stop	1	0	0	1	0
Store	0	1	0	0	0
Supply	1	0	1	0	0
Support	1	0	1	0	0
Transfer	1	0	1	0	0

The two sets of the input data, the bond graph and the engineering performance parameters offer numerous combinations. In mathematical terms, these sets are a bijection function with two sets of permutations, $S(V)$ and $S(E)$. The number of potential combinations of these two sets can be modelled by Equation 1. Thus, the number of solutions can potentially be massive and computationally challenging.

$$\binom{n}{k} = C(N,k) = \frac{n(n-1)\dots(n-k+1)}{k!} = \frac{n!}{k!\,(n-k)!} \quad \text{where } \binom{n}{k} = \binom{n}{n-k} \tag{1}$$

The matching algorithm uses isomorphism, ψ, which is a structure preserving bijection. Two graphs are isomorphic if they contain the same number of vertices connected by the same set of edges. The initial input of the user is defined as $graph_1$, G_1 and the repository graphs are G_n. The matching D-APPS algorithm then compares the vertices of the two graphs, $V(G_1)$ and $V(G_n)$. Mathematically, this

is shown in Equation 2. The isomorphism quality is then used to determine the ranking of the returned solutions.

$$\psi: V(G_1) = V(G_n) \quad \text{and} \quad \psi^{-1}: V(G_1) = V(G_n) \tag{2}$$

The potential use of this capability has numerous applications across the design community. Beginning with the design problem definition, the critical functions are established to seek out alternative solutions for use with bond graphs that operate upon these engineering performance parameters. The returned analogies should relate to the performance parameters identified by the engineer. The design engineer can then integrate the design to achieve the targeted outcome.

The DRACULA algorithm is developed with extensibility in mind and features that allow for future versions and concepts. The current use of three analogies with three critical functions each, constrains the complexity of the design space. The algorithm will provide the top three analogies retrieved from the repository in decreasing score as calculated from the ranking equation. Each analogy set requires at least one critical function, or up to three critical functions pairs together in a function chain.

3.1 DRACULA flowcharts

A top-level view of the analogy matching algorithm, from the perspective of the end user is shown in Figure 1. The DRACULA algorithm needs two items from the user, the critical function(s) and the engineering performance parameter(s). The application then performs the unit matching, the functionality categorization and the final ranking of all the potential entries. As such, there are three critical functions chains the algorithm needs to account for: 1) multiple individual critical functions, 2) single critical functions and 3) critical function chains. A user can identify up to three sets of three critical function chains and corresponding engineering performance parameters.

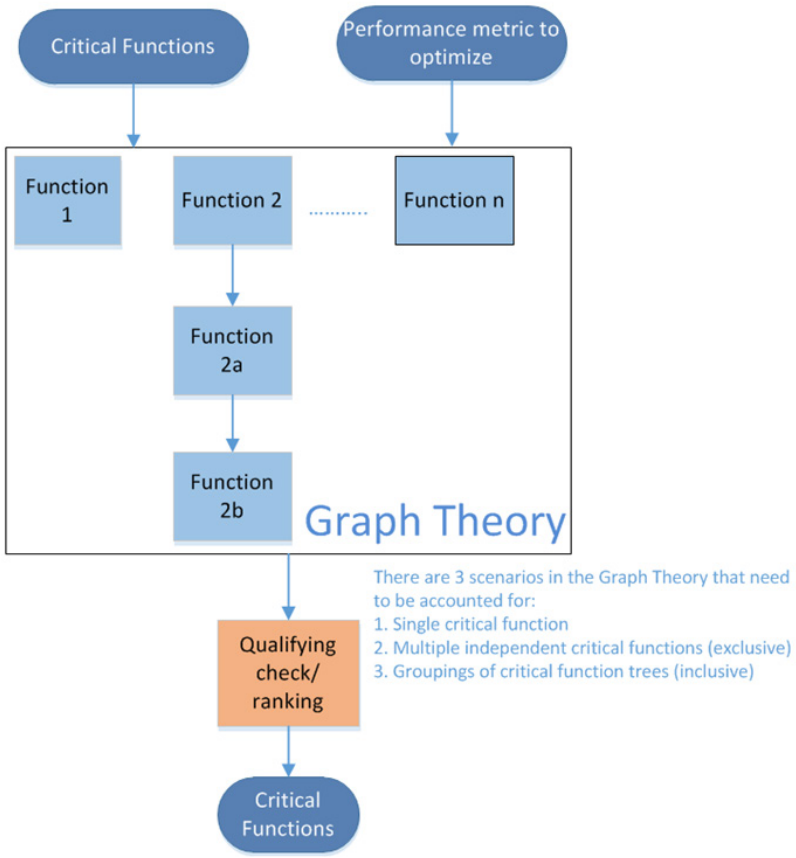

Figure 1. System-level algorithm flowchart.

A more detailed flowchart of the algorithm can be seen in Figure 2. The program begins by initializing the log file and the core data files. The first user input is from loading the repository data files. An error will be thrown if the data file is not loaded. The user now selects the items to be matched, the

engineering performance parameters and the critical functions, then progresses down to clicking the "Compute Analogies" button. Once enabled, the algorithm performs the matching and saves the potential solutions to a results vector. When the entire repository has been searched, the results vector is ranked for the best analogy and returned to the user. The user has the option to save the entire computation to a log file for further analysis.

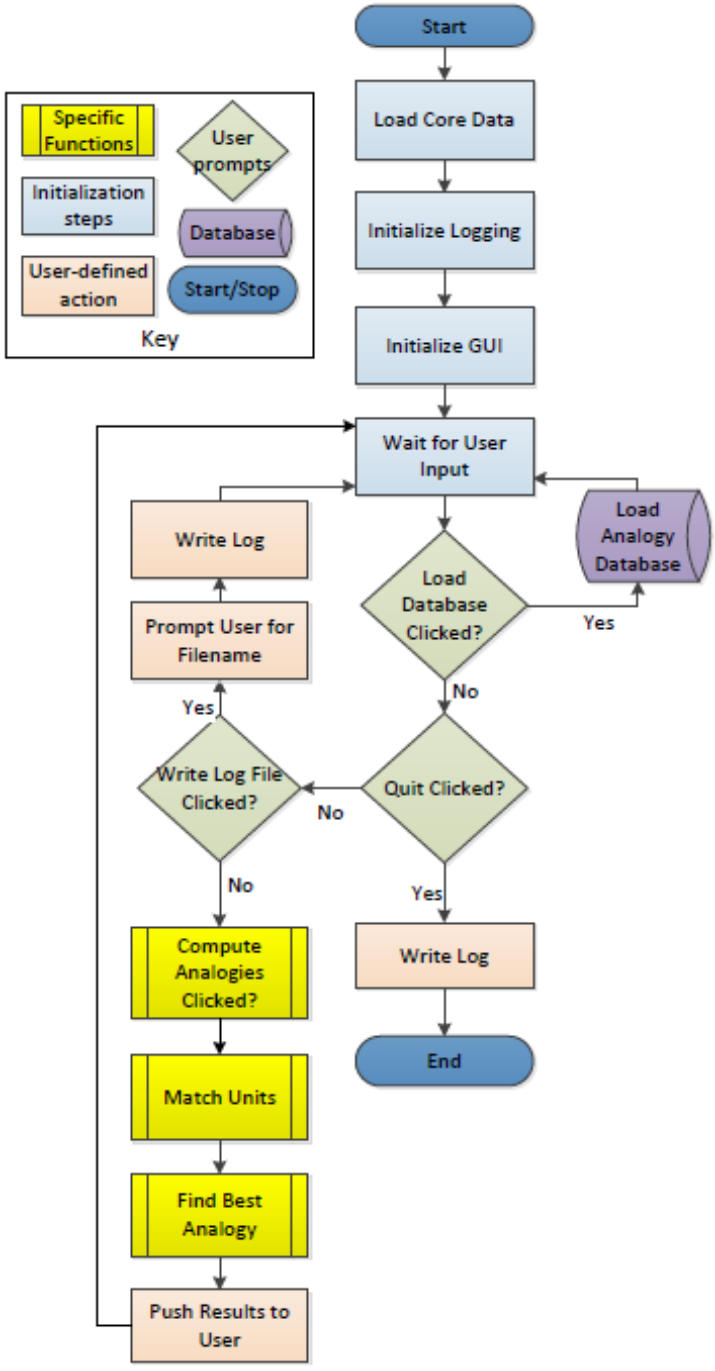

Figure 2. Detailed algorithm flowchart.

The matching is done in a two-step process: by 1) matching the units of the engineering performance parameter and 2) correlating the critical functions to other candidate functions via bond graph category. The unit matching is performed on the 9-tuple "unit vector" of the input engineering parameter with repository engineering parameters. The vector values are compared for a targeted zero-sum in the *matchUnits*() function. The units are also compared for compatibility with the designated bond graph classes in *bondGraphCompatible*(). A match of the same bond graph type results in an identified match and sent to the results vector.

The *matchUnits*() function performs the matching via a 3 step process for each analogy set (up to three values currently):

1. Compare the engineering unit parameters (9 values) to those of the repository via *sumEqual*() function.
2. Compare final critical function of the function chain against those critical functions of the repository by translating the critical function to a bond graph parameter (5 values) via *bondGraphCompatibility*() function.
3. Send repository entry to the results vector for ranking with *scoreChainPair*().

The *findBestAnalogy*() function calls two sub-functions, *scoreAnalogy*() and *scoreChainPairs*() function. The *scoreChainPair*() functions takes the critical functions of the function chain and calculates the proximity of closeness to the original chain input of the user. The repository categories are ranked according to the Equation 3.

$$scoreChainPair(chain_1, chain_2) = \frac{\sum_{i=0}^{n_1}\left(\sum\left(\begin{cases}\frac{1}{1+|i-j|} & if\ chain_1[i] = chain_2[j] \\ 0 & otherwise\end{cases}\right)\right)}{max(n_1, n_2)^2} \quad (3)$$

where

$chain_1$ = input critical function chain
$chain_2$ = repository critical function chain
n_1 = length of critical function $chain_1$
n_2 = length of critical function $chain_2$

The input function chain from the user is compared against the repository critical pairs as identified by the unit and bond graph matching step of *matchUnits*(). The repository entries that have the critical functions in at least a partial sequence get a higher ranking than an entry that has just an individual critical function match. The *scoreAnalogy*() function assesses the proximity of identical critical functions in two function chains. It returns a value of 1.0 when chain1 and chain2 are identical. It penalizes mis-ordered elements by computing the corresponding index shifts and penalizes differences in chain length by normalizing the summation by the square of the greater chain length. The function flowchart of all these aforementioned functions can be seen in Figure 3.

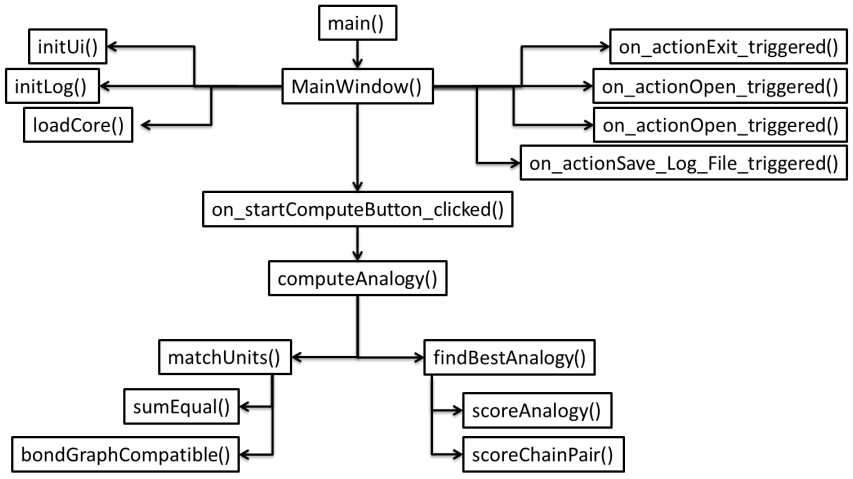

Figure 3. DRACULA function flow diagram. Function relationshipe represented by arrows.

A systems level view of the user inputs and the functions of the algorithm can be seen in Figure 4. The progression of the figure is step dependent, not time dependent. The algorithm will offer solutions when given the user-inputs of critical function(s) and an engineering performance parameter. When prompted, the calculations are written to a log file.

In the future, the algorithm will able to sustain numerous analogy matches with variable numbers of critical function groups into chains. By sequencing the order of the functions in the function chains, it is possible to construct a mufti-dimensional design space. The concept of isomorphism will still be applicable to the matches of the future, however more complex.

The future algorithm will be an optimization problem in finding the shortest path between the critical chains and the final engineering performance parameter. Repository entries that have the final engineering parameter, can then be ranked on the shortest, most efficient or effective match possible.

3.2 Interfaces

The DRACULA algorithm has two sets of interfaces, one with the database, and one with the user Figure 5. The database interface uses text files generated from the Excel spreadsheet repository. The text file is derived from the critical pairs of the repository entry as shown in Figure 5.

Each analogy variable is associated with three sequential functions. The first function of the functional basis is the first function acting on the flow. The second and third functions are then the next critical functions in the critical function chain. The engineering parameter is then correlated to the last critical pair as the final calculation for the ranking scheme. However, the engineering performance parameter is checked against each critical function.

DRACULA provides the top three repository matches to the user input in the GUI. It is also possible to write the operation steps and the solutions to a log file. The log file includes the calculations of each repository entry, the unit matching and the score of each entry with respect to user's design problem. The final ranking score of the matching algorithm is included at the end of each analogy set.

3.3 Inputs/Outputs

DRACULA has currently limits the complexity of the design space to three analogy variables. This means that there are three single engineering variables being optimized individually and being acted upon by up to three critical functions. This prototype version was designed such in order to provide a proof of concept for the method of analogy mapping to the flows. It is anticipated that future versions will be able to sustain more analogy variables.

4 MATCHING ALGORITHM OPERATION

To use DRACULA, it is necessary to identify the key performance parameters and the Critical Functions (CFs) acting upon the parameter. The DRACULA program allows up to three sets of CFs and a single key performance parameter to be selected as a single set. To operate the tool, the user must load the input database file from the File menu. Then select one of the engineering performance parameters from the drop down list in the first analogy section. Next, select the first critical function that will be acting on the corresponding flow. Up to three critical functions modelled as critical function chains can be represented in the current version of the software. DRACULA can operate with one, two or three CFs acting on the performance parameter. To execute the search, click on the “Compute Analogies” button. The program will return the top three database matches with the corresponding engineering performance parameter and the critical functions. The entries are ranked based on how close they match the CF Chains and are returned to the user for evaluation.

1. Load database repository: File → Load Analogy Database: load the .txt file of flattened repository data.
2. From the drop-down list of Engineering Parameter, select the parameter to be optimized.
3. From the drop-down list of Critical Function(s), select the first critical function of the design problem.
4. From the drop-down list of Critical Function(s), select the second critical function of the design problem.
5. From the drop-down list of Critical Function(s), select the third critical function of the design problem.
6. Repeat steps 2-5 for the other 2 analogy sets.
7. Click “Compute Analogies”.
8. Save the log file: File → Save Log File: select the directory to save the log file.
9. Exit the program: File → Exit.

Currently, the DRACULA program does not tell the engineer why the repository option is the best option or contextualize how the technology can relate to the current design problem. Instead, it offers

solutions based upon the engineering performance parameter and the functionality of the system. It is up to the design engineer to provide a context for the suggested abstraction.

5 LIMITATIONS

The DRACULA 1.0 program has several current limitations. The first limitation is dependent upon the repository having at least three analogy entries in order for the algorithm to perform searches. This limitation is a trivial matter as the repository has already surpassed this immediate threshold value. However, it is included here for completeness.

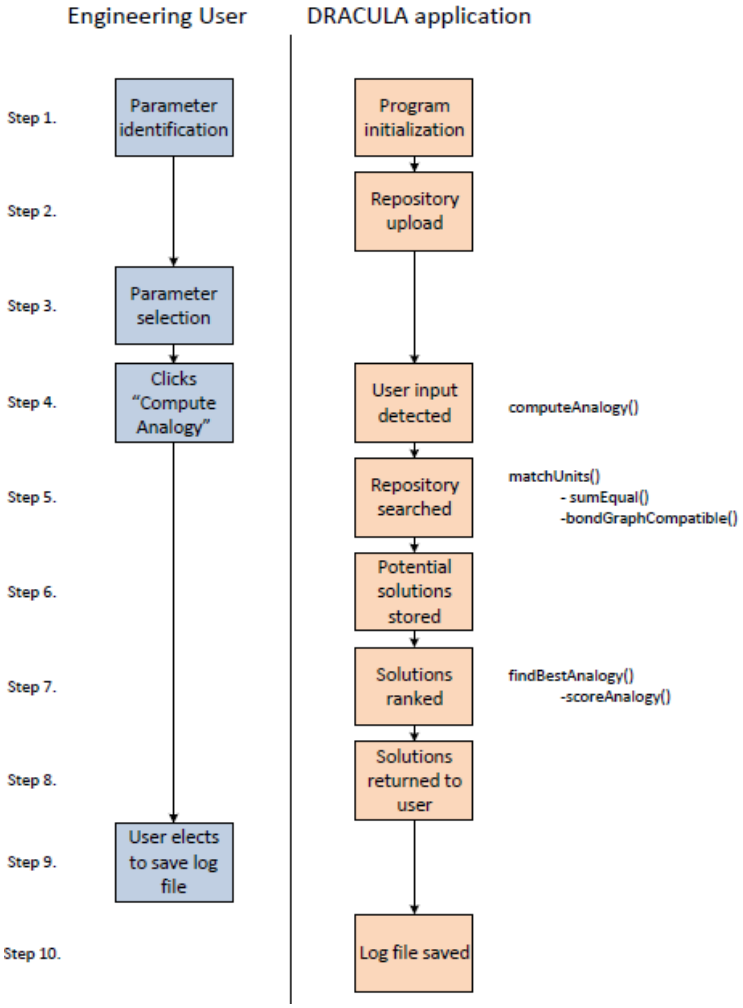

Figure 4. Time line representation of user inputs and application functions.

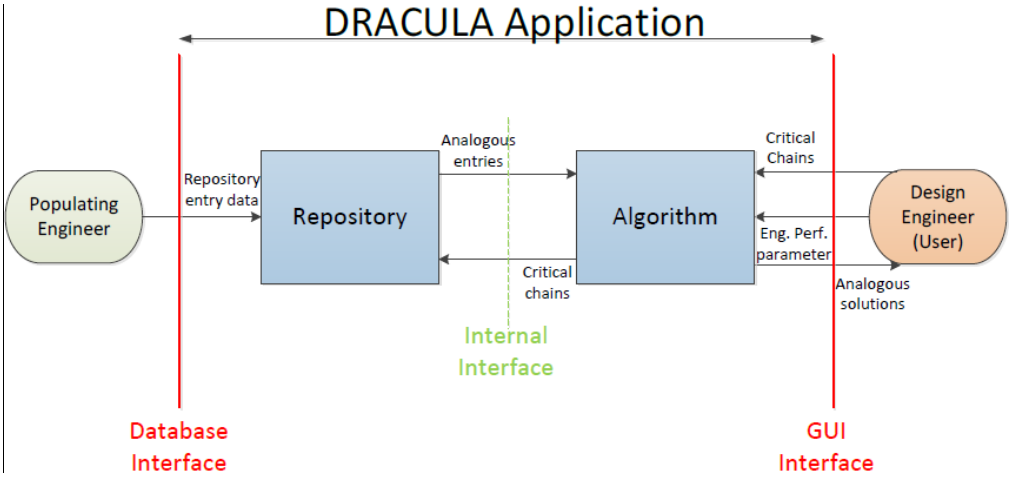

Figure 5. DRACULA interfaces from repository and algorithm.

Second, the algorithm must have the input files in the stated formats. The engineering performance parameter file and bond graph category type must remain in the flattened syntax. Should additional performance parameters be required, they can be added to the file, but the vector dimensions in the mainWindow.h file of the program also must be updated.

The final known limitation is the search depth of the critical function chains. The current proof-of-concept software only allows three critical functions to be combined into a single function chain. Design problems can potentially have more than three sequential chains, so this limitation can be handled in future versions of the software. The depth of the chains should be variable and up to the user to define based upon their specific design problem.

5.1 Build Details

The build details for the DRACULA software can be found in Table 3.

Table 3. Software build environment.

Version	1.0
Runtime	Desktop Qt 5.3.1, using Qt Creator 3.1.2 development environment
Compiler	Microsoft Visual C++ 2012
Graphics	OpenGL 32-bit
Compatibility	MS Windows XP, 7, 8
	Apple OSX
	Debian / Ubuntu Linux

6 EXAMPLE PROBLEM

The DRACULA 1.0 program has been tested for feasibility and validity of the implementation. The program was given an example problem of a wind turbine optimization. The design parameters of the wind turbine have specifically targeted to reduce the drag, increase the lift, and decrease the weight of the turbine in order to increase the efficiency. The KPP for the turbine system is to *efficiently convert wind energy into electrical energy*. The critical chains identified for the wind turbine operation, are *guide*, *convert* and *transfer* for each of the engineering performance parameters of Figure 6.

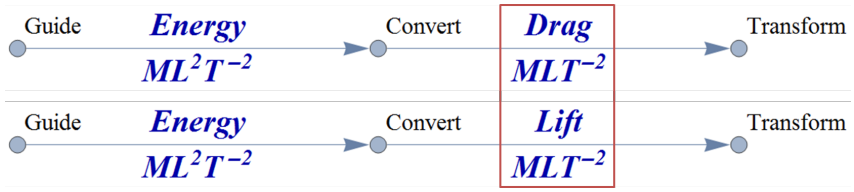

Figure 6. Wind turbine critical function chain.

The critical function chain and engineering performance parameter input when run with the DRACULA application returns the values in Figure 7. The sequence of the critical function chain was not altered between the analogy sets; only the engineering parameter was varied. This was done purposefully in order to show the functionality and operation of the application. The three parameters when paired with the *guide*, *convert* and *regulat*e function chain, produced the same three analogies. This is expected, as the *lift* and *drag* units are of the same order and the sequences are the same.

The three returned analogies offer the options of the bird shell loading (AskNature - bird, 2014), the stream bryophytes (AskNature, 1999), and the whale fin option with tubercles on the trailing edge (AskNature – whale, 2014)The bird shell option resists structural damage by using strong materials. In the context of the wind turbine blade, smooth surfaces, free of fractures are ideal for avoiding drag and increasing efficiency. The stream bryophytes are aerodynamically shaped to increase the lift of the organism. The whale fin option is thus the best entry to produce an analogy to reach the KPPs. So by altering the shape of the blade depending on the angle of attack of the wind, the turbine can garner a greater efficiency.

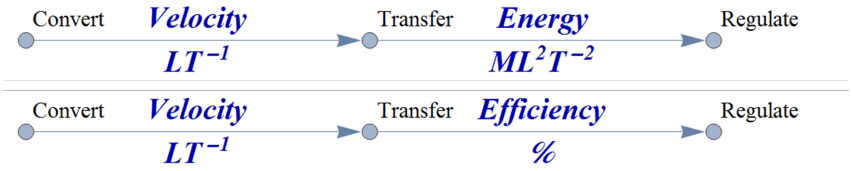

Figure 7. Wind turbine critical function chain.

For each of these three options, it is up to the user to understand how the analogy could be applicable to the current design problem. The DRACULA tool is only the conduit for increasing innovation, not how the application can be employed. In the current case, the bumps on the back of the turbine blade would allow for some optimization of the *lift* and the *drag*. But by also altering the shape and adopting the stream-line shape of the bryophytes, the wind turbine can garner even a greater *efficiency*. Finally, the material of the turbine blade needs to be considered when incorporating design changes.

7 CONCLUSION AND WAY FORWARD

DRACULA 1.0 is a proof-of-concept demonstration of the underlying theory of this research, presented in this paper. Using the analogy repository of DRACULA, and the matching algorithm outlined herein, this proof-of-concept software has been able to generate performance based analogies. Further work will need to explore the effectiveness of this approach, populate the repository to a greater extent, and consider how this new DbA method can improve design and engineering.

The current DRACULA algorithm uses the concept of graph theory to match target and analogous solutions. Functional modelling via function structures allows the system to be represented graphically. The application of graph theory with the critical functions and critical flows follows in the same vein. The critical functions with their bond graph component act as the nodes of the graph. These nodes are then connected through the edges, or the non-dimensionalized engineering performance parameters. The critical flows are associated with the non-dimensionalized engineering performance parameters.

The matching of the algorithm uses the edges and nodes to isomorphically compare graphs in a design space. The DRACULA application compares the output engineering performance parameter with the functions of the function chains. The sequence of the chains is important for identifying the order of action of the final outcome. The order of these functions dictates which actions should be sequentially completed and can result in different analogies depending on the desired user input.

REFERENCES

AskNature.org, (2013) Shells resist external loading: birds. [Online], http://www.asknature.org/strategy/2a55e373b6355bda01aa84c12f322b63#.VDyWNvldV8E (December 2013).

AskNature.org, (2014) Flippers provide lift, reduce drag: humpback whale. [Online] http://www.asknature.org/strategy/3f2fb504a0cd000eae85d5dcc4915dd4\#.VDHTO_ldV8E (May 2014).

Chartrand, G., (1985). Introduction to Graph Theory. Mineola(New York): Dover Publications.

Chartrand, G. & Zhang, P. (2012) A first course in graph theory. Mineola(New York): Dover Publications.

Euler, L., 1741. Solutio problematis ad geometriam situs pertinentis. Commentarii academiae scientiarum Petropolitanae, Volume 8, pp. 128-140.

Gould, R. (2012) Graph Theory. Mineola(New York): Dover Publications.

Hirtz, J., Stone, R.B., McAdams, D.A., Szykman, S., and Wood, K. L. (2002) A functional basis for engineering design: reconciling and evolving previous efforts. *Research in Engineering Design*, 13(2), pp. 65-82.

Lucero, B. (2014) Design Analogy Performance Parameter System (D-APPS). Thesis (PhD).

Lucero, B., Vishwanathan, V., Linsey, J.L., and Turner, C. (2014). "Identifying Critical Functions for Use Across Engineering Design Domains." Journal of Mechanical Design, 136(12). 121101.

Otto, K. N. and Wood, K. L. 2001. Product design: techniques in reverse engineering and new product development. Upper Saddle River, NJ: Prentice Hall.

Pahl, G., and Beitz, W. 1996 Engineering Design: A Systematic Approach. London: Springer - Verlag.

Stream Bryophyte Group. (1999) Roles of bryophytes in stream ecosystems. Journal of the North American Benthological Society, 18(2), pp. 151-184.

Trudeau, R. J. (1993) Introduction to graph theory. Mineola (New York): Dover Publications.

ACKNOWLEDGMENTS

Partial support for this work was provided by the National Science Foundation Award No. CMMI-1234859 and CMMI-1304383. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.