INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'07
28 - 31 AUGUST 2007, CITE DES SCIENCES ET DE L'INDUSTRIE, PARIS, FRANCE

A GRAPH TOPOLOGY OPTIMIZATION METHOD FOR THE SYNTHESIS OF ROBUST AND EFFICIENT ROUTES

Sandeep Nair[1], Matthew I. Campbell[2]

[1]Graduate Research Asst., Department of Mechanical Engineering - University of Texas at Austin, 1 University Station, C2200 - Austin, TX 78712-02992, USA
Tel: 512-577-1053, email: nair.sandip@gmail.com
[2]Associate Professor, Department of Mechanical Engineering - University of Texas at Austin 1 University Station, C2200 - Austin, TX 78712-02992, USA
Tel: 512-232-9122, Fax: 512-471-7682 , email: mc1@mail.utexas.edu

ABSTRACT

Several renowned classical tree search methods such as depth first, breadth first, A-star etc. are in place today for dealing with problems of topology and parametric optimization. We propose a novel optimization technique based on mathematical graph transformations in order to synthesize feasible and optimal graph topologies. This paper demonstrates the methodology and results from the application of this new optimization method to the *Route* test problem. The power of the approach lies in the high level of abstraction afforded by the usage of mathematical graphs and the fact that the entire optimization process is viewed as a large tree search. Additionally the generic nature of the method enables tackling problems of a multi-disciplinary nature to be a feasible proposition in the future. The optimization technique is broken up into four separate modules named *representation, generation*, *guidance*, and *evaluation*. The modules correspond to problem formulation (*representation)*, synthesizing a broad spectrum of feasible topologies (*generation)*, navigating the search process towards progressively better solutions (*guidance)* and evaluating the worth of each topology respectively (*evaluation)*. This fundamentally new optimization method is specifically intended for performing topology optimization of graphs and thus has potential application in domains as varied as engineering, computer science and artificial intelligence.

Keywords: grammar, rule, graph, topology, graph transformations, route, networks, ruleset

1 INTRODUCTION

As it stands today there are numerous computational tools which aid in the analysis portion of the engineering design process. Some of these may be Finite Element Analysis (FEA) [3], or computational fluid dynamics (CFD) [4]; however there is a clear lack of any such commonplace tools for the synthesis aspect of the design process. Keeping in mind current design cycle times and technological advances it is obvious that designers would benefit from the development of such a computational synthesis plus optimization tool that would emphasize or focus on topological variations and optima. Although topology optimization problems are being studied by several researchers [5] there is a lack of consensus on commonalities between the problems or techniques being used to solve them. In an effort to deal with topological problems researchers at the University of Texas at Austin have developed a powerful, generic computational tool named GraphSynth [1]. The underlying principles have been derived from fields such as traditional optimization, artificial intelligence [6] and graph theory [7]. Other software tools [8, 9, 10, 11] have been developed capturing and manipulating graph grammars but these tend to lack generality, an object-oriented framework, a modern user-interface, and tools to allow automatic rule execution.

The new search method is based on organizing the set of candidate solutions as a tree of solutions that are represented by repeating calls of grammar rules [12] written in GraphSynth. The test problem that was chosen to demonstrate the optimization method in this paper is named *Route*. In the *Route* problem the aim is to create the best network between multiple locations or cities. In this research, best is defined by minimizing both the cost of establishing all the routes and the cost of traversing a route between every pair of cities. These two objective functions force the search to strike a balance between minimizing the total number of routes, and optimizing the network for quick traversal. These two goals are described in more detail in the *evaluation* section of this paper.

There are several potential areas of application for this new optimization method. One of these is the inventory routing problem (IRP) [13, 14, and 15]. The IRP involves integration and co-ordination of two key components of the logistics value chain: vehicle routing and inventory management [16]. The *evaluation* module of the IRP would involve minimizing the distribution costs during the planning period without causing the inventory stocks to run low thus inconveniencing customers. Flight scheduling [17] is another problem which could take advantage of the new method. This problem is especially interesting in a modern day context with airlines worldwide having to cut costs in several facets of their operations. Parameters such as placement of hubs, lay-over times, ground servicing could be treated as some of the optimization knobs.

We also envisage that this optimization method could be used to study and solve network flow problems [18]. These according to classical graph theory involve an assignment of flow to the edges of a directed graph where each arc has a particular capacity, such that the amount of flow along a particular arc does not exceed its capacity. These network flow problems are a gateway to optimizing transportation networks, electrical distribution systems or even problems in ecology.

The simplest elements of a mathematical graph are a node and an arc. It is on the basis of these atomic constituents of graphs that the new optimization method has been developed. The crux of the optimization method proposed relies on what is classically referred to as graph transformations [12]. In essence the graph transformations are analogous to the basic set theory operations of union and intersection. Thus with an eye on representing any engineering design or network problem such as *Route*, it became imperative to come up with a set of rules that encapsulated the design space. These rules are known as grammar rules. Typically a grammar rule consists of a left hand side (L), right hand side (R) and a commonality graph (K) between them [19, 20]. On a higher level; a set of rules making up a grammar ruleset is the means for a human designer to organize thoughts and capture the complete design space into a compact framework. Thus, the rules become the currency of representing the problem. The real challenge for the designer now lies in creating a grammar ruleset that has the ability to capture every feasible graph topology while at the same time minimizing the number of unfeasible topologies that are encountered.

The entire optimization process of the new method is envisioned as a large tree search. Each state in this tree defines a candidate topology. The various transitions in the tree correspond to grammar rule applications. Traditionally tree searching algorithms such as A* search [2] or Uniform Cost search [2] have been used to solve for optima. However in the *Route* problem these cannot be used as the space is large, multi-modal, and non-monotonic. Classical branch-and-bound [21] is also ruled out for *Route* since there is no clear upper bound that can be calculated or predicted. Hence we have developed our own algorithms with a certain stochastic element that would enable jumping from one branch of the tree to another thus preventing it from getting stuck in local minima. In the *Route* problems the optimal graph lays between two extremities namely the minimal spanning tree and the complete graph for `n` number of cities. Well established algorithms by Prim [22] and Kruskal [23] are already in place to deduce the minimum spanning tree for any connected weighted graph. Our endeavour is to seek the optimal topology which lies embedded in the tree somewhere between the minimum spanning tree and complete graph.

2 METHODOLOGY

Our approach to the entire problem is divided into four generic compartments or modules: *representation*, *generation*, *evaluation* and *guidance.* The method has been depicted as a flowchart in Figure 1 Each one of the modules has been explained below for the *Route* problem.

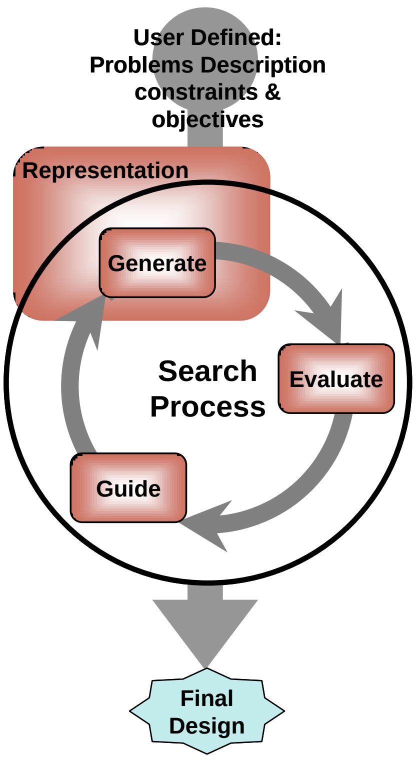

Figure 1

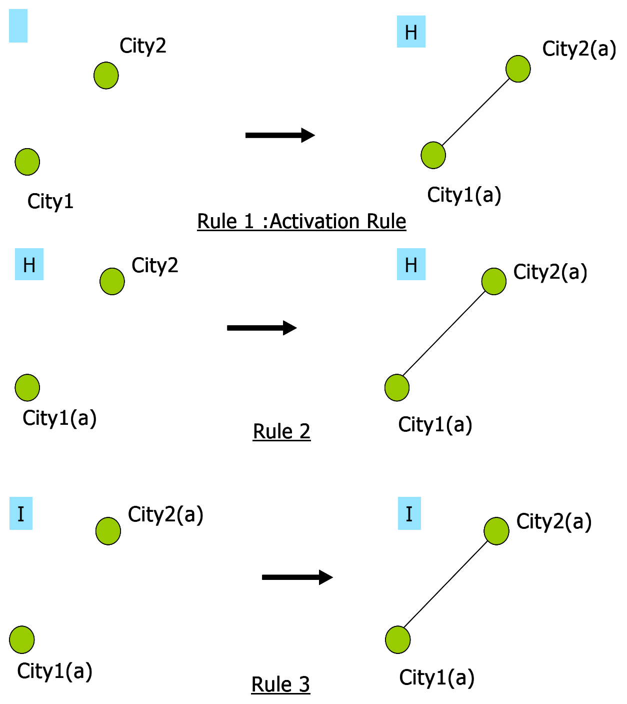

Figure 2

2.1 Representation

The first step to solving the problem lies in problem formulation. The cities in *Route* are represented as nodes and the roads connecting them as arcs. The input to the entire process is a "seed graph" which is basically a non-simplicial graph [24] depicting only the nodes or cities. Additionally there are three grammar rules for the *Route* problem which have been placed under two sequential rulesets. A ruleset has been defined by the researchers from an implementation perspective. From a human designer's perspective a ruleset is a set of rules which may have been grouped according to any number of reasons e.g. global or local labels that are applicable to the rules are common, certain rules occur within a certain stage of the design process etc.

The basic components of any grammar rule are a left hand side of application conditions (L), a right hand side of resulting graph transformations (R) and a graph of common elements (i.e. nodes, arcs or labels) which essentially provide the context for the particular rule's application. This commonality is indicated by the symbol K. The K is a very crucial aspect of any grammar rule as it shows the intersection or overlap between the L and R sides of the rule. The Figure 1 indicates the grammar rules that have been developed for representing the *Route* problem. The labels that may be a part of K are indicated by the superscripts in the top left of the grammar rule.

Computationally rules 1 and 2 have been grouped into a ruleset named "getToSpanningTree" while rule 3 has been put under "getToCompleteGraph". Since we know that the graph representing the optimal solution lies between the minimum spanning tree [7] and complete graph [7] these two rulesets when applied in conjunction generate all possible graph topologies that lie between the said two anchor points. More implementation details about the rulesets have not been discussed as it is beyond the scope of this paper.

2.2 Generation

The generation module is the second major step in the entire optimization process. Essentially the generation module involves performing some graph transformations to synthesize varied topologies. In the *Route* problem this step corresponds to producing graphs between a spanning tree and the complete graph. From classical graph theory we know that for a graph of \`n\` nodes (i.e. cities in *Route*) the total possibilities of unique spanning trees are 2^n [24]. Empirically there are $\frac{n(n-1)}{2}$ arcs in a complete graph of \`n\` nodes. Additionally it is known that $n-1$ arcs must exist in a graph of \`n\` nodes for it to be deemed a spanning tree. Thus if we were to consider the *Route* problem for 20 cities, we would be dealing with 1048576 unique spanning trees! The optimal solution lies anywhere between graphs with 19 arcs (spanning trees) to 190 arcs (complete graph). With such a small set of nodes the large nature of the search space becomes obvious.

As mentioned earlier one of the initial specifications to the optimization process is known as the “seed graph”. The seed graph is viewed to be the embryo that can spawn any possible topology as long as the necessary graph transformations i.e. rule applications are performed upon it. In *Route* however the grammar has been designed in a way such that only feasible topologies are generated thus the search space that has to be dealt with is more manageable.

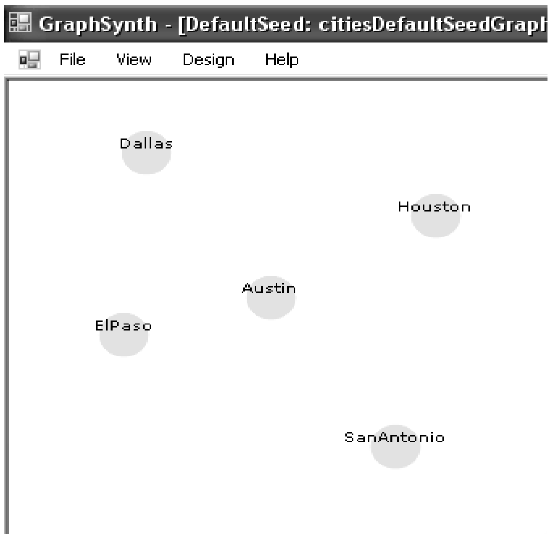

Figure 3: Seed graph

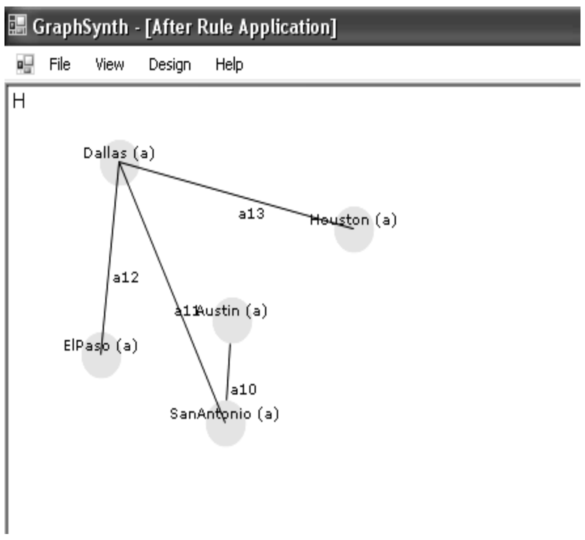

Figure 4: Sample spanning tree

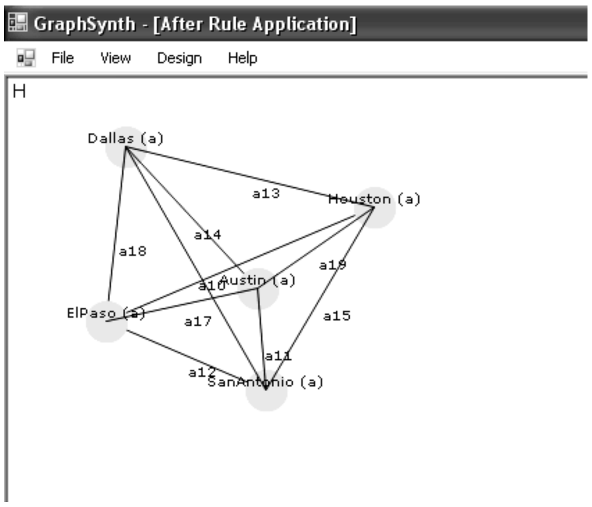

Figure 5: Complete graph

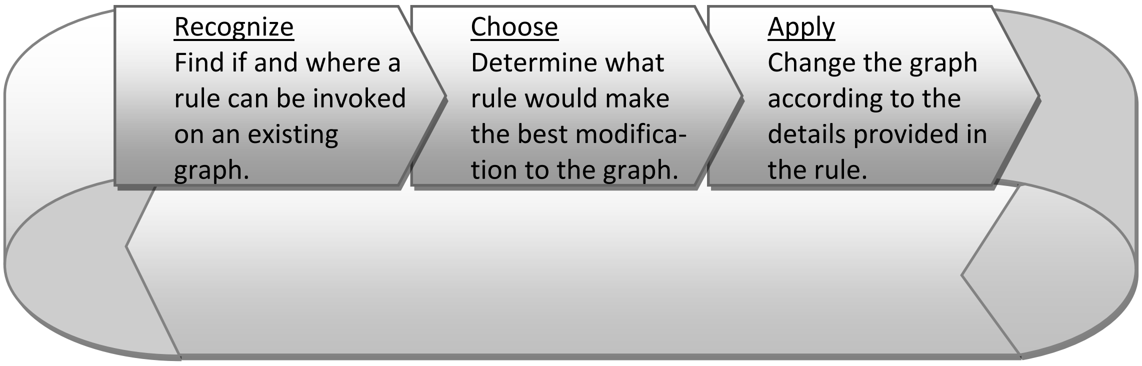
Recognize
Find if and where a rule can be invoked on an existing graph.
Choose
Determine what rule would make the best modifica-tion to the graph.
Apply
Change the graph according to the details provided in the rule.

Figure 6: There are three distinct steps in generation: 1) recognizing what rules are applicable, 2) choosing one of these rules to apply, and 3) the application of the rule which involves a graph transformation of the host to a newly synthesized state.

The Figure 3 depicts a seed graph of five major states in the state of Texas, USA used in the *Route* problem. The seed graph is a non-simplicial or `n` component graph (`n` being the number of cities). Figure 4 indicates a sample spanning tree while Figure 5 shows the complete graph with 10 arcs (i.e. $\frac{n(n-1)}{2}$) for the five cities. The *generation* module is effectuated in conjunction with the grammar. This has been built into the background of the optimization package GraphSynth. From the implementation perspective the entire *generation* module is a distinct three step process: recognize, choose, and apply (Figure 6). At any stage in the creation of various topologies of *Route*, the designer would like to know of all possible changes that could be made. The recognize step involves checking all possible rules of the applicable rulesets with the graph to determine if the application conditions allow the rule to be applied [12]. Once the recognition is done the next step is to choose the rule to be considered for application [25]. This choice may be done via the human designer through a GUI, randomly, or by an intelligent agent [26] based on the feedback of the entire optimization routine. The final step is the rule application. In *Route* upon application new routes (arcs) are introduced into the graph.

From the graph theory perspective the *generation* module is performed by the combined Double Pushout/ Algorithmic Free Arc Embedding approach [27, 28].

2.3 Evaluation

In any computation based optimization routine it is an accepted fact that the evaluation phase is one of the most intensive both in terms of computational memory and time. In the new method proposed evaluation is the third major module. All along the optimization routine it is important to note that a fixed number of candidate topologies are maintained. This is commonly referred to as "pool" of candidates. Once a candidate pool has been generated the candidates are sent to the evaluation module. Here the worth of each candidate is evaluated.

As the title of the paper suggests, the goal is to create candidates that are both *robust* and *efficient*. The term *robust* here means that the solutions obtained have no loss in connectivity between any pair of cities, thus ensuring that there is always a path to get from one city to another. In graph theory terms this means that solutions that are feasible but not necessarily optimal correspond to simplicial graphs or one-component graphs. The term *efficient* means that the candidate is productive without being wasteful. In the *Route* problem, this means laying down the minimum number of routes. As a result, a multi-objective problem is established.

In order to elucidate the evaluation, consider the example candidate topology shown in Figure 7. The names a, b, c, d, e indicates five cities in the *Route* problem. The arcs ad, de, eb, bc depict the routes present in the candidate topology. The heuristic for the problem is the fact that the shortest distance between any two points (i.e. cities in *Route*) is a straight line. Thus the evaluation function has been defined as:

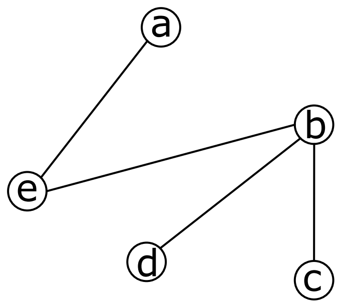

Figure 7

Minimize f such that:

$$f = W_1\{f_1\} + (1 - W_1)\{f_2\} \tag{1}$$

Where:

$W_1 =$ Weighting factor = 0 ~ 1

$f_1 = \{ae + db + eb + bc\}$

$$f_2 = \left\{\begin{array}{l} (ae+eb)+(ae+eb+bc)+(ae+eb+bd)+(ae)+ \\ (bc)+(bd)+(be)+ \\ (cb+bd)+(cb+be)+ \\ (db+be) \end{array}\right\}$$

In this equation, the first term in the braces defines the efficiency measure. It simply represents the sum of lengths of the arcs within the topology. If this objective is the only one used, then the optimization will seek to remove arcs. However, since the rules presented in Section 2.1 constrain our search to one-component graphs, then the global minimum will be the minimum spanning tree.

The second objective function in *Route* is shown in the second pair of braces. Each parenthetical expression corresponds to the distance traversed from each city to every other city. The first term captures the distance from 'a' to 'b'; the second from 'a' to 'c', etc. As the reader can see, individual arcs appear in multiple expressions. If this objective is taken alone, then the optimization will tend toward the complete graph so that one arc corresponds with every single possible journey.

This second objective requires a slightly more complicated algorithm than the first. In general, these objectives do not have a closed form, as each topology will have a different number and arrangement of arcs. In this second objective, a uniform cost search [29] is required to find the shortest path between every pair of cities. Uniform cost search is a type of tree search algorithm that is similar to breadth first search but orders solutions based on their path cost as opposed to their level. It is a complete method and guarantees the optimum, although it is sometimes slow and demands high computational resources for large trees. We have found that for thirty cities, the determination of this objective becomes a significant bottleneck for the optimization method.

2.4 Guidance

This final module of the optimization method is the step where some decisions are made regarding the fate of each candidate topology. The main emphasis of the *guidance* module is to progressively navigate the search process towards better solutions over several iterations or until a particular convergence criterion has been met. It has been noticed that existing approaches to optimization commonly involve usage of the popular genetic algorithms [30]. These have been extensively favoured by researchers for solving various problems. Rather than just pursue the optimum based on a single metric like the fitness value of the candidate, our optimization method harnesses the tree like nature of the search space. The new algorithm that has been developed and implemented for the *Route* problem, though inspired from the genetic algorithm, is still quite different from it.

The optimization algorithm has been named *QuattroElitism*. As a first step inside the *guidance* module the entire pool of candidates that have been evaluated are now sorted in descending order of fitness values which were assigned to them in the *evaluation* module. Once this has been done four basic operators have been defined within *guidance*. The operators are: *preserve*, *makechild*, *invokeparent* and *remove.* At this point it becomes important to mention that before candidate topologies are sent to the relevant operator for further action a check is introduced so as to equalize the sizes of the *preserve* and *remove* computational lists. This is done to keep the lists the same size during each iteration to ensure that the population size does not change over the span of the search process. The various operators have been defined with an eye on being able to move down, backtrack or stay put on the state tree (Figure 8).

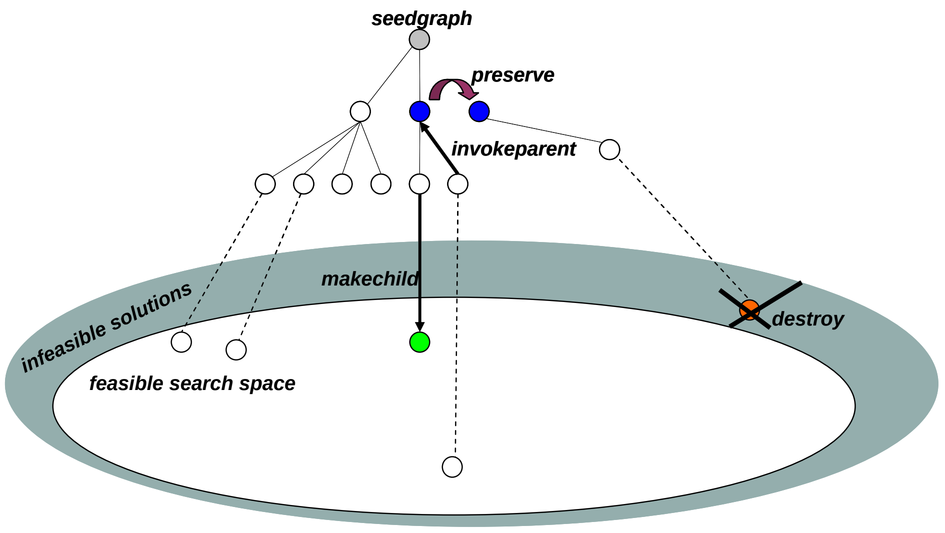

Figure 8: A state tree view of the *guidance* operators

The first step within *guidance* is to divide the pool of candidate topologies into quarters based on decreasing fitness values. The top quarter of candidates are sent to the *preserve* operator (Figure 9). Within this operator a true copy of each candidate is made and half the copies are sent to the *makechild* operator while the other half are sent to the *invokeparent* operator. Thus at the end of applying the *preserve* operator to every candidate in the first list it ends up containing the top quarter of candidates and a true copy of every candidate.

The second quarter of the candidates is sent to the *makechild* operator (Figure 9). Here each candidate is again sent to the *generate* module and a single grammar rule is randomly selected from the various applicable choices and applied to the candidate graph topology to complete the graph transformation. This operator enables the search process to aggressively move down the state tree provided the size of the perturbation is sufficiently large.

The third quarter of candidate topologies is sent to the *invokeparent* operator (Figure 9). Once again upon sending each of the candidates to the *generate* module the grammar rule that had been last applied to the candidate before its current state is removed or "undone". Essentially the *invokeparent* operator enables the search process with the capability of backtracking or moving up the state tree as if it feels that bad design decisions are being pursued. In the case of the *Route* problem this could mean one of several things e.g. the fitness values of candidates suddenly become large upon further pursuing the downward path on a particular branch of the state tree, or a city is left out of the solution etc.

The final quarter of the candidates is sent to the *remove* operator (Figure 9). Here each of the said candidate topologies is entirely removed from the population i.e. deleted. However the overall population of the candidates still remains the same static number as the copies of candidates that had reached the *preserve* operator replace the ones lost due to the *remove* operator.

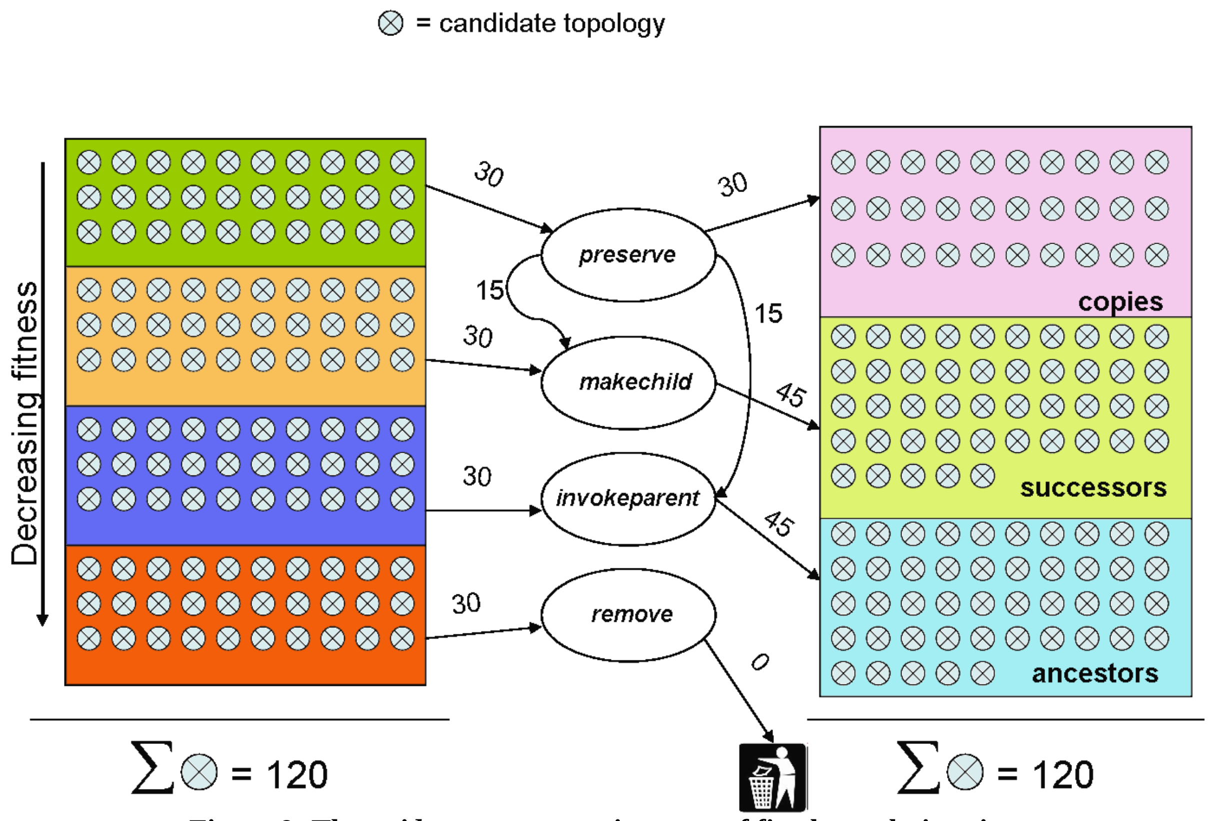

Figure 9: The *guidance* operators in terms of fixed population size

3 RESULTS & CONCLUSION

The graphs resulting from the runs of the new optimization method have been presented in Figures 9-12. The final results have also been tabulated in Table A. The machine used for the purpose was a standard Windows PC with an AMD processor running at 2.21 GHz, 2 gigabytes of RAM. The software development environment was Visual Studio.NET. A candidate pool size of 20 was chosen and the entire optimization routine was run for 1000 iterations for each of one the optimal topologies generated. The approximate amount of time taken was 227 seconds.

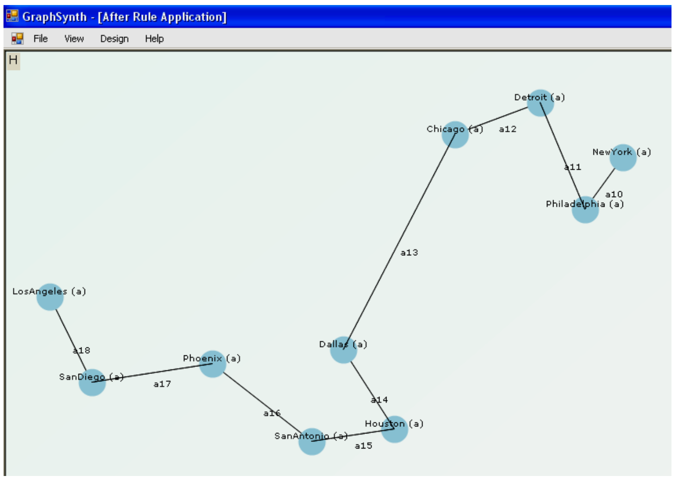

Figure 10: Minimum Spanning Tree

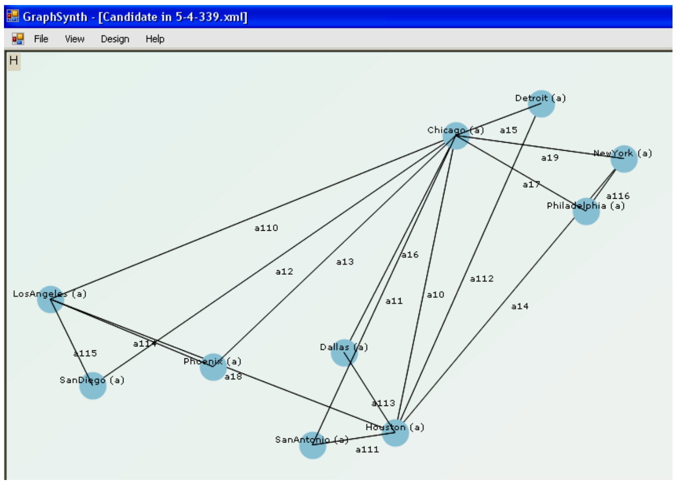

Figure 11: W_1 **= 0.8**

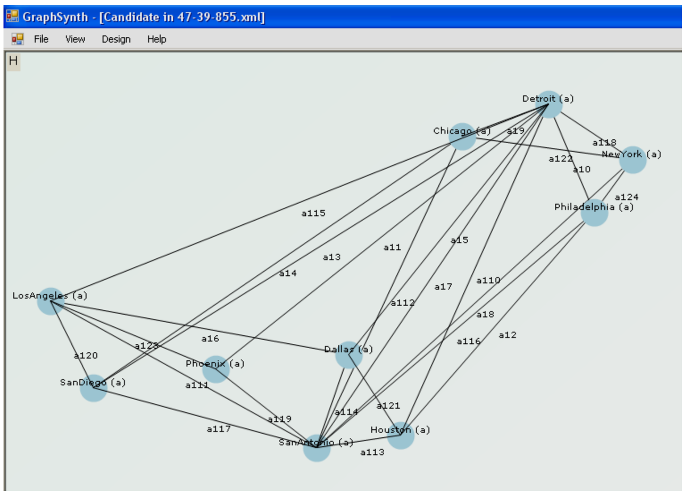

Figure 12: $W_1 = 0.5$

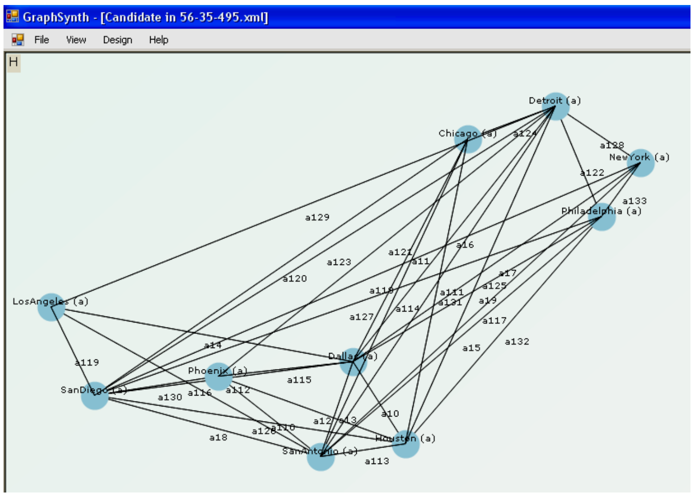

Figure 13: $W_1 = 0.2$

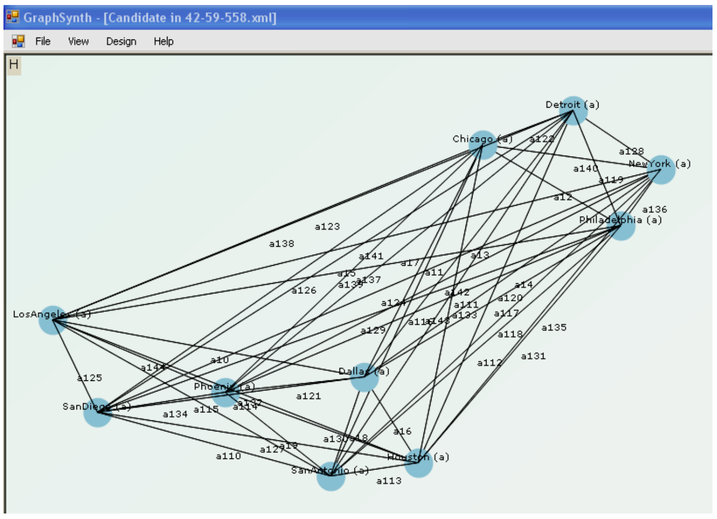

Figure 14: Complete Graph

Tabulated results					
Weighting Factor (W_1)	1	0.8	0.5	0.2	0
First objective (f_1)	3235	14212	23743	36554	53397
Second objective (f_2)	65744	60459	55760	53946	53397
Number of arcs	9	17	25	34	45

Table A.

The final results for the *Route* problem with 10 major American cities have been tabulated in Table A. Upon setting the weighting factor "knob" W_1 equal to 1 it essentially ended up capturing the Minimum spanning tree (Figure 10) which had 9 arcs for 10 nodes. Similarly upon setting W_1 equal to 0 we capture the complete graph with 45 arcs for 10 nodes (Figure 14). Figures 11, 12 and 13 capture three other optima when W_1 has been set equal to 0.8, 0.5 and 0.2 respectively. Upon examining the *evaluation* function it is clear that if W_1 is given a bigger weight then the optimal graph topology will get pulled toward the Minimum spanning tree i.e. the graph will tend to have fewer arcs thus making it more *efficient*. Similarly if W_1 is given a lower weight then the optimal topology tends to get pushed toward the complete graph i.e. the graph will tend to have as many arcs as possible thus making it more *robust*.

In this paper, we have presented a new search method that seeks optimal graph topologies for a specified objective function or functions. This is the first topology optimization method that has been developed specifically for domains representable by a graph grammar schema. The key driver of this new search technique is the fact that the search space behaves like a large tree. With the help of the atomic operators defined earlier under Section 2.4, basic tree traversal techniques have been outlined that are the cornerstone of this technique.

The *QuattroElitism* approach described in this paper is not the only *guidance* method being considered by the researchers. There is evidence that more efficient methods may be used to tackle problems such as *Route*. This proposed new *guidance* strategy would involve a more stochastic approach akin to branch-n-bound techniques. Another aspect that is being investigated and that could greatly speed up the current approach is to enable the algorithm to make an informed or intelligent decision of which branches of the tree are to be avoided. This is not to say that *QuattroElitism* will not work for certain specific network problems. An important learning derived from this research endeavour has been the fact that specific applications may demand specific guidance methods. Hence, we plan to develop more *guidance* techniques which learn ways to target or mimic design decisions such as preferring one grammar rule over another.

As a next step to furthering this work we plan to implement a GA as a *guidance* strategy for the *Route* problem and compare results between *QuattroElitism* and the GA. Other applications such as MEMS devices and neural network problems are planned too as test problems for *QuattroElitism*. The *representation, generation* and *evaluation* modules for the above applications have been developed by other researchers in our lab. This fundamentally new search technique holds great potential for application in other fields within engineering and AI.

REFERENCES

[1] Campbell M.I., "The Official GraphSynth Site", University of Texas at Austin, 2006.

[2] Onwubiko C.O., Introduction to Engineering Design Optimization, Prentice-Hall, 2000.

[3] Bathe K.J., Finite element procedures, Prentice Hall, 1996.

[4] Anderson J.D., Computational Fluid Dynamics: The Basics with Applications, www.cfd-online.com, 2002.

[5] Bendsoe M.P. and Sigmund O., Topology Optimization: Theory, methods and application, Springer, 2004

[6] Pfaltz J. and Rosenfeld A., "Web Grammars," presented at Proc.1st Int. Joint Conf. on Artificial Intelligence, Washington, D.C., May, 1969.

[7] Diestel R., Graph Theory (Graduate Texts in Mathematics, 173), Springer, 1997.

[8] Grammatica. http://www.lsi.upc.es/~valiente/grammatica/

[9] Visual OCL. http://tfs.cs.tu-berlin.de/vocl/

[10] PROGRES. http://www-i3.informatik.rwth-aachen.de/tikiwiki/tiki-index.php?page_ref_id=213

[11] AGG. http://tfs.cs.tu-berlin.de/agg/

[12] Ehrig H., Engels G., Kreowski H.J. and Rozenberg G., Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools, World Scientific Publishing Company, 1999.

[13] Campbell, A., Savelsbergh M., Clarke L. and Kleywegt A., "The Inventory Routing Problem", Fleet Management and Logistics, edited by Crainic T.G. and Laporte G., Kluwer Academic Publishers, 95-112, 1998.

[14] Campbell, A. and Savelsbergh M., "Inventory Routing in Practice", The Vehicle Routing Problem, edited by Toth P. and Vigo D., SIAM Monographs on Discrete Mathematics and Applications, 2002.

[15] Campbell, A. and Savelsbergh M., "A Decomposition Approach for the Inventory Routing Problem", Transportation Science, Volume 38, Number 4, Pages 488-502, 2004.

[16] Campbell, A. and Savelsbergh M., "Efficient Insertion Heuristics for Vehicle Routing and Scheduling Problems", Transportation Science, Volume 38, Number 3, 369-378, 2004.

[17] Feo T.A. and Bard J.F., Flight Scheduling and Maintenance Base Planning, Management Science, JSTOR, 1989.
[18] Ahuja R.K., Magnanti T.L., Orlin J.B., Network flows: theory, algorithms, and applications, Prentice-Hall, 1993.
[19] Schaefer J. and Rudolph S., "Satellite design by design grammars," Aerospace Science and Technology, Vol.9, pp. 81-91, 2005.
[20] Göttler H., Graphgrammatiken in der Softwaretechnik: Theorie und Anwendungen (Informatik-Fachberichte): Springer, 1988.
[21] Lawler E.L., Wood D.E., Branch-And-Bound Methods: A Survey, Operations Research, JSTOR, 1996.
[22] Prim R.C., "Shortest connection networks and some generalizations", Bell System. Tech. J., vol. 36, no. 6, pp. 1389–1401, Nov. 1957.
[23] Kruskal, J. B. On the shortest spanning subtree of a graph and the traveling salesman Problem. Proc. Amer. Math. Sot. 7, 48-50, 1956.
[24] Hartsfield N. and Ringel G., Pearls in Graph Theory: A Comprehensive Introduction. Academic Press, 1990.
[25] Arora J., Introduction to Optimum Design, Second Edition: Academic Press, 2004.
[26] Shen W. and Norrie D.H., Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey, Knowledge and Information Systems, 1999.
[27] Janssens D. and Rozenberg G., "Graph-Grammars with Neighborhood-Controlled Embedding," Theoretical Computer Science, vol. 21, pp. 55-74, 1982.
[28] Vandenbroek P.M., "Algebraic Graph Rewriting Using a Single Pushout," Lecture Notes in Computer Science, vol. 493, pp. 90-102, 1991.
[29] Bolc L. and Cytowski J., Search Methods for Artificial Intelligence: Academic Pr, 1992.
[30] Goldberg D.E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Longman Publishing Co., Inc., 1989.

Contact: **Matthew I. Campbell**
The University of Texas at Austin
Mechanical Engineering
1 University Station, C 2200
Austin, TX 78712-02992
USA
Phone: +1-512-232-9122
Fax: +1-512-471-7682
e-mail: mc1@mail.utexas.edu
URL: www.me.utexas.edu/~campbell