The Thesis Committee for Mukund Kumar
Certifies that this is the approved version of the following thesis:

Rule Based Stochastic Tree Search

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor: ______________________________
Matthew I Campbell

____
Richard Crawford

Rule Based Stochastic Tree Search

by

Mukund Kumar B.Tech

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2011

Abstract

Rule Based Stochastic Tree Search

Mukund Kumar, for M.S.E.

The University of Texas at Austin, 2011

Supervisor: Matthew I Campbell

This work presents an enhancement of a search process that is suited for a problem that can be solved using a graph grammar based generative tree. Generative grammar can be used to generate a vast number of design alternatives by using a seed graph of the problem and a set of transformation rules. The problem is to find the best solution among this space by doing the least number of evaluations possible. In a previous paper, an interactive algorithm for searching in a graph grammar representation was presented. The process was demonstrated for a problem of tying a necktie and the work here builds on top of this process to be useful for solving engineering problem. To test the search process, two problems, a photovoltaic array topology optimization problem and an electromechanical product redesign problem, are chosen. It is shown this search process converges in finding the best solution within a few hundred evaluations which is a manageable number compared to the large solution space of millions of candidates. Further optimization and tweaks are done on the process to control exploration vs. exploitation and find the parameters for fastest convergence and the best solution.

Table of Contents

List of Tables vii

List of Figures viii

Introduction 1

Related work 5

Representation 9

- Stochastic algorithm for tree search 13
- Rule Based Interactive Tree Search 15
  - Knowledge Repository 16
  - Selection of an option 18
  - Stochastic choose 19
  - Necktie Experiment 23
  - Modification to the RBITS process 24
  - Evaluation Mechanism 25
  - Option information 27

Solar panel optimization problem 30

  - Representation 32
  - Specifications of the problem 39
  - Evaluation mechanism 39
  - Optimization problem 42
- Results 43
  - Experimenting with Q 45
  - Experimenting with B 48
  - Selecting best values of B and Q 51
  - Application to problem of higher complexity 54

Product assemblies 57

  - Experiment 61

Conclusions and future work 67

Bibliography... 70

List of Tables

Table 1 Snapshot of rule knowledge as used by RBITS 16

Table 2 Selection of option based on preference 20

Table 3 Connectivity information represented using labels and arcs............ 34

Table 4 Variation of Convergence and Rating with B and Q 53

Table 5 Candidates from which rules were generated 59

Table 6 Max rating and average convergence point of search process for B and Q values from 0 to 1 and 0.1 to 0.9 respectively with seed of hydraulic jack FS. 66

List of Figures

Figure 1 Using graphs for representation of a 4 bar mechanism..................... 9

Figure 2 Example of a graph grammar rule to modify a node with label x to y10

Figure 3 Tree based generative grammar method to find solutions starting from a seed node 12

Figure 4 RBITS search process. Candidates evaluated and added to knowledge repository are highlighted in black. Dotted arrows represent option with highest probabilities 22

Figure 6 Rules applied to same location but in different order lead to same candidate 29

Figure 7 Construction of a PV array. Connectivity between modules of PV array is open to be optimized 32

Figure 8 Graph grammar representations of certain connectivity patterns 35

Figure 9 Nodes not positioned on a grid cannot be represented in this approach 36

Figure 10 Rules for creating connection between nodes in a seed graph. Fig a, b, c, d show rules to create arcs in N, E, S, W directions. Rule creates arcs to connecto global positive/negative terminals 37

Figure 11 Example seed graph for a 9 node PV array. A possible solution is shown in figure b 38

Figure 12 Different possibility of target solution for solar panel optimization problem 40

Figure 13 Convergence of rating vs. number of iterations. Graphs are generated with same settings but different stochastic runs produce different output (a, b, c). Plot d shows an average of 20 such runs (a, b, and c plus 17 more). 44

Figure 14 Progress of average rating vs. number of iterations with different sets of B and Q values grouped by Q 47

Figure 15 Candidates that contain some same options but have different rating49

Figure 16 Progress of average rating vs. number of iterations with different sets of B and Q valuesgrouped by B 52

Figure 17 Progress of candidate rating vs. iteration number for a candidate of size 24 nodes 56

Figure 18 Screenshots of Function Structure (top) and Component flow Graph (bottom) of a common hair dryer 58

Figure 19 Rules that transform a single function node to multiple component nodes 59

Figure 20 Convergence of search process when used with seed graph as hydraulic jack 64

Figure 21 Convergence of search process using a portable compressor seed of size 25 nodes 65

Introduction

Graph grammars are becoming a powerful representation technique for conceptual design generation. Unlike other graph based representation techniques such as bond graphs or circuit diagrams which are specific to a certain kind of problem and aimed at solving for a specific design requirements, graph grammars are more generalized. In addition to the properties of a graph, it can be made to contain more information in the form of labels and directed arcs. This information can be valuable to a design problem where the labels could hold any property that describes the node which could be useful from an evaluation or a generation point of view.

Apart from representation, the power of graph grammars is present in the generation process where graph transformation is achieved by the use of grammar rules. Generative processes usually use a vector representation such as in the case of Genetic Algorithm where a bit string represents a solution and individual bits are modified to generate variations in solution. Grammar rules are more powerful as they test the graph before application (Recognition) and therefore do not give infeasible solutions. Apart from this, rules can be made very complex where a single rule application could open up to an unpredictable number of possible designs, but the number of designs that can be defined with a bit string vector representation is limited to 2^n where n is the length of the vector. From a designer's point of view, an important advantage is that the representation is easy to understand. Rules and graph can be easily represented on a GUI and can be easily understood by a designer.

In this work, a grammar is used to solve topology optimization problem in two different problem domains. The first problem is solving a topology optimization problem to improve the performance of Photovoltaic (PV) cell arrays. Solar power is one of the most commonly used renewable energy source but each solar cell gives very low voltage and is affected by the amount of irradiation on each cell [1]. Control methods such as maximum power point tracking (MPPT) have been tried to solve this. However, in domestic applications, shading is a common issue due to restrictions on where the solar panels can be placed and partial shading of the solar panel array can greatly reduce the power output. In such cases it has been found that by modifying the connectivity between the PV modules, a stabilized power output can be obtained. The problem of finding the optimum connection topology has to be done by considering different variations and evaluating each to find the best among them. But the number of different combinations in a PV array as described later is in the order of several millions and therefore a more intelligent approach must be taken to reduce the number of evaluations. The final solution can be a topology with arbitrary connectivity that is unknown at the onset of the search and the focus in this work is to develop a search process that is capable of finding the final solution without any dependence on empirical knowledge.

The second problem uses grammar rule and tree search for selecting and configuring the components of electromechanical products. There have been several such techniques for attempting to solve the problem of component selection in the past. They have largely revolved around detailing the functions of the product and breaking down the functions to sub functions and finding the right component to satisfy each function.

However, real design automation here is much more complicated when we consider components that can solve multiple functions or we use components that can solve a function only in the presence of other components. With graph grammars, these additional complex relations are easily captured in the form of rules and a database of rules can be used to solve a function structure of a product. The fundamental difference between a database of rules and a database of components is that rules can capture additional interaction aspects of how a function or a group of functions is realized while a components database simply captures the use of each of the components. Rules can be generated from other products thereby capturing how ideas can be reused across different products. In the problem that is described here, rules were extracted from products such as a hair dryer, water pump etc. and later used on these same products to see how they can be redesigned.

The work described here focuses on building a search technique that works on a generative grammar search tree for the topology optimization problems described above. Optimization and tree search processes are closely related and some techniques are present in both domains. Several classic techniques exist for tree search techniques such as A* and IDA*, uniform cost search etc. that work by using a heuristic function and have been shown to give good results without making any assumption on the nature of the problem. Optimization techniques that work on vector representation of a problem such as simulated annealing, hill climbing and biologically inspired search processes such as ant colony optimization, particle swarm optimization work on trying to find the minimum value of the objective function. The method here creates a tree search technique that aims at finding the minima on a rule based generative tree problem. A search algorithm that precisely does this is Rule Based Interactive Tree Search which presents a powerful framework for search within a generative grammar framework. The work presented here builds upon RBITS to suit for the nature of a topology optimization problem that encompasses a wide range or problems. To test the search process, an evaluation function is simulated that rates each design as the distance from a preset ideal solution. The search process does not have any information other than this distance metric and the aim is to arrive at the final solution at the fastest rate. There are several challenges and optimizations that can be done to speed up the search process which are discussed in detail and the effect of the search parameters are studied and tweaked to give the best results.

Related work

Engineering design uses graphs extensively for representation. In laying out electric circuits or chemical process diagrams, they make the design process systematic and break down complex systems into simpler modules. In conceptual design synthesis, function structures [2] and flowcharts have proved as a quick and clear way to describe large and complicated designs.

Graph grammars have been adopted for design synthesis for their ability to easily create topological changes to graphs in a rigorous and systematic fashion [3]. Designs are represented by graphs and this could be modified in several different ways by addition or deletion or modification of nodes and arcs. Using this model, in mechanical engineering, graph grammars have been used by researchers to solve a variety of problems such as mechanism synthesis [4], synthesis of gear trains [5], sheet metal design [6], function structures [7], coffee makers [8], truss structures [9] and mechanical clocks [10]. A key advantage of using graph grammars is the use of graphical software to display and interpret grammar in intuitive fashion [11] that speeds up problem representation.

In this work, graph grammars are used as a representation technique in two completely different problem domains. In the first case a PV array topology optimization problem is taken where the best connectivity within an array of PV modules is to be determined considering shading patterns. Without shading, standard patterns for connectivity have been derived that maximizes the power output. The graph grammar representation for this problem has been developed previously [12]. This has been found more effective than a genetic algorithm approach due to the intuitive representation of the feasible design solution.

The second problem is conceptual design synthesis for electromechanical systems. This involves optimization of the topology and selection of components that constitute the system. Hundal [13] designed a program for automated conceptual design that associates a database of solutions for each function in a function database. Ward and Seering [14] developed a mechanical design “compiler” to support catalog-based design. Both these methods can be used for electromechanical systems and the underlining principle is that they work by breaking the product into a functional model and find components to satisfy each function while trying to achieve a maximum desirability. Both these models can only consider a level of complexity at which each function is satisfied by each component. A graph grammar representation however encodes much more complex forms of function-component mapping by using grammar rules. Previous work on computational design synthesis for product assemblies using graph grammar representation [15] describes a two layer approach as a simplification of the Function Behavior Structure approach [16]. This uses function structure and a Component Flow Graph to represent electromechanical products such as compressors, blowers, hair dryer etc. The work shows that millions of alternative designs solutions can be generated from a generative tree. The work aims organize the space of design solutions into groups using a clustering methodology in an attempt to find the best solution. In this work, we try to use search techniques in such a generative tree to find the best solution.

There are several tree search algorithms that can be used for graph grammar based generative trees. Exhaustive methods such as breadth first and depth first search scan every node before finding the best possible solution. However these methods cannot be used in problems that will be presented here due to the sheer size of the number of nodes in the tree. Using heuristics, several methods have been developed in the past that scan only a part of the tree by placing a certain assumption on the nature of the heuristics. Methods such as A*, IDA* and uniform cost search [17] scan every node as they progress into the search tree while making a decision on the branch to be selected for search in the next step. This is faster than exhaustive methods as it scans a lesser number of nodes but they need a heuristic function to give an approximate evaluation which is not always achievable in the class of problems presented here.

Another category of methods are based on optimization techniques that operate on a vector representation of the problem but can be ported to a graph grammar representation. A recursive search method for creating simple mechanical block diagrams is developed by Schmidt and Cagan [18] that builds structures from a graph grammar. The work uses a simulated annealing approach to navigate the tree of design solutions as if it were a multidimensional design space. The method TP^2 (Topological and Parametric Tune and Prune) has been explicitly developed for graph grammar based problems such as a truss topology optimization problem [19]. In this the fundamental issue of automation in conceptual design is realized as a search problem in topology optimization and a problem of optimizing the specifications of the design.

Genetic algorithm is a search technique based on a vector representation that has been used for many optimization processes. However formulating the design problem as a vector is a challenge in many cases leading to long vector lengths. The generation of new solutions from the bit string by arbitrarily changing the individual bits can lead to infeasible designs that have to be restricted by the use of explicit or implicit constraints or using penalty functions. This greatly slows down the search process and a comparison of the TP^2 search technique and genetic algorithm shows that graph based search technique perform better as the generation process is unrestricted [19].

Rule Based Interactive Tree Search is a method developed that adopts a completely different approach for tree search technique [20]. Instead of building on top of existing optimization techniques which are based on vector representation, RBITS uses the rules as the fundamental building blocks of a design generated from a graph grammar approach. The methods described above extend a technique that is applicable for a vector representation to graph grammar based representation. However, RBITS uses the representation as an aid in guiding the optimization process and the work that is presented in this paper builds on top of this method. It will be described later as why this method cannot be optimally translated on a vector representation model due to the nature of the generation process.

Representation

A graph grammar is used as the representation technique where the components and their interactions are laid out in the form of a graph. A graph is defined as a collection of nodes and the connections between nodes known as arcs. Graph grammars provide a language for creating graphs that are usable for a particular domain of problems. Often and as is the case in the included examples, graphs in engineering design use the nodes in a graph to represent components and arcs represent the interactions between components. Distinguishing characteristics amongst nodes and arcs are defined by a list of labels that exists within a given node or arc. An example of using graph grammar is shown in Figure 1 to represent mechanisms [3] where the nodes represent the pivots and the links while the arcs represent the connectivity between them. Labels on the nodes and arcs define the type of the joints and the position of the node directly determines the position of the link in a 2D space.

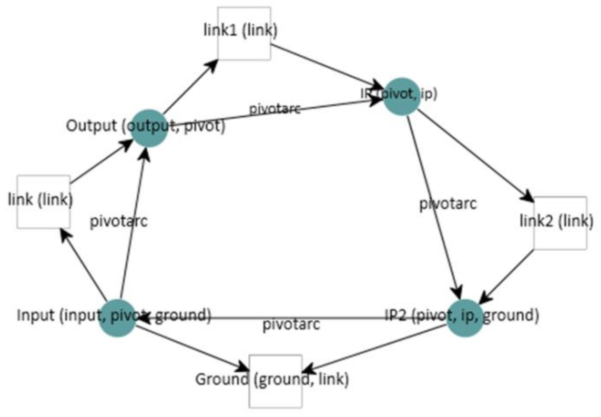


Figure 1 Using graphs for representation of a 4 bar mechanism

Graph representations manifest themselves in many places such as bond graphs, circuit design etc. and they serve as an intermediate step before a set of equations can be inferred from them for evaluation. However, the power of graph grammar lies in the transformation of graphs using grammar rules that enable generation of variations from the original graph. Grammar rules, detect certain properties from the graph and if they are met, modifies the graph is specified methods. For example, a rule can be defined that detects the presence of a node with a particular label “x” in the graph and changes it to a node with a label “y”. This rule could transform graphs representing motors of type “x” to type “y”. The rule is visually represented as shown in Figure 2. The L part of the rule is recognized in the input graph and is substituted with the R part of the rule. K contains the common nodes and arcs between the two rules that aid rule transformation and in reconnecting arcs that originally connected the nodes between them.

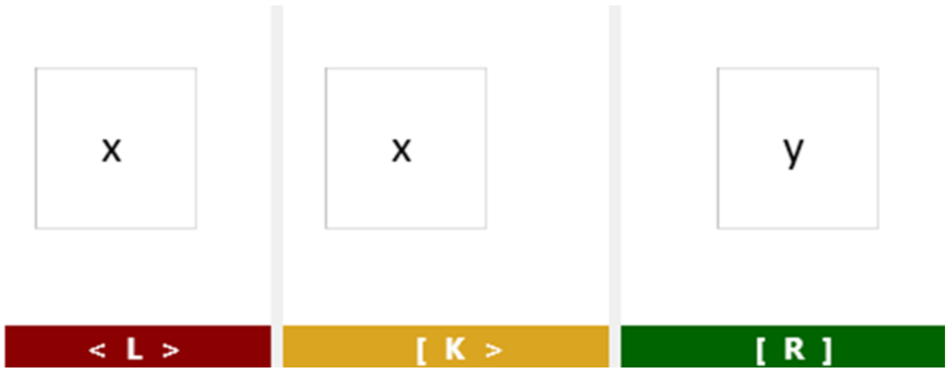


Figure 2 Example of a graph grammar rule to modify a node with label x to y

Rules can be much more complex adding to the versatility of the representation. Rules can also negate the matching of certain labels i.e. if a node contains a particular label, it is excluded from being recognized. Other restrictions such as matching positions of nodes can also be encoded that will be used later in this work. The rules are generated and viewed using GraphSynth [10].

A sequence of rules can potentially transform graphs. In a generative process, a seed graph is used as the starting point with a set of rules that are allowed to transform the seed. In each step, from this rule set, a subset of rules can be recognized in the graph that lead to a set of transformed graphs and on each of these graphs, a different subset of rules can be recognized. The result is a tree of solutions as shown in Figure 3 that continue until no more rules can be recognized. In the problem domains that we consider, only the leaf nodes are considered as valid solutions and the list of rules that lead to the solution is called the recipe or the genotype while the graph itself is the phenotype. While a graph itself can be examined for properties and evaluated for a rating by a human or an automated evaluator, the rules exist to provide the graph with features that modify the ratings and cannot be attributed with any particular characteristics or evaluated a priori. The analogy is that of living species that are the final solutions of the generative tree where the sequence of genes that gives the characteristics is similar to the list of rules that create the solution.

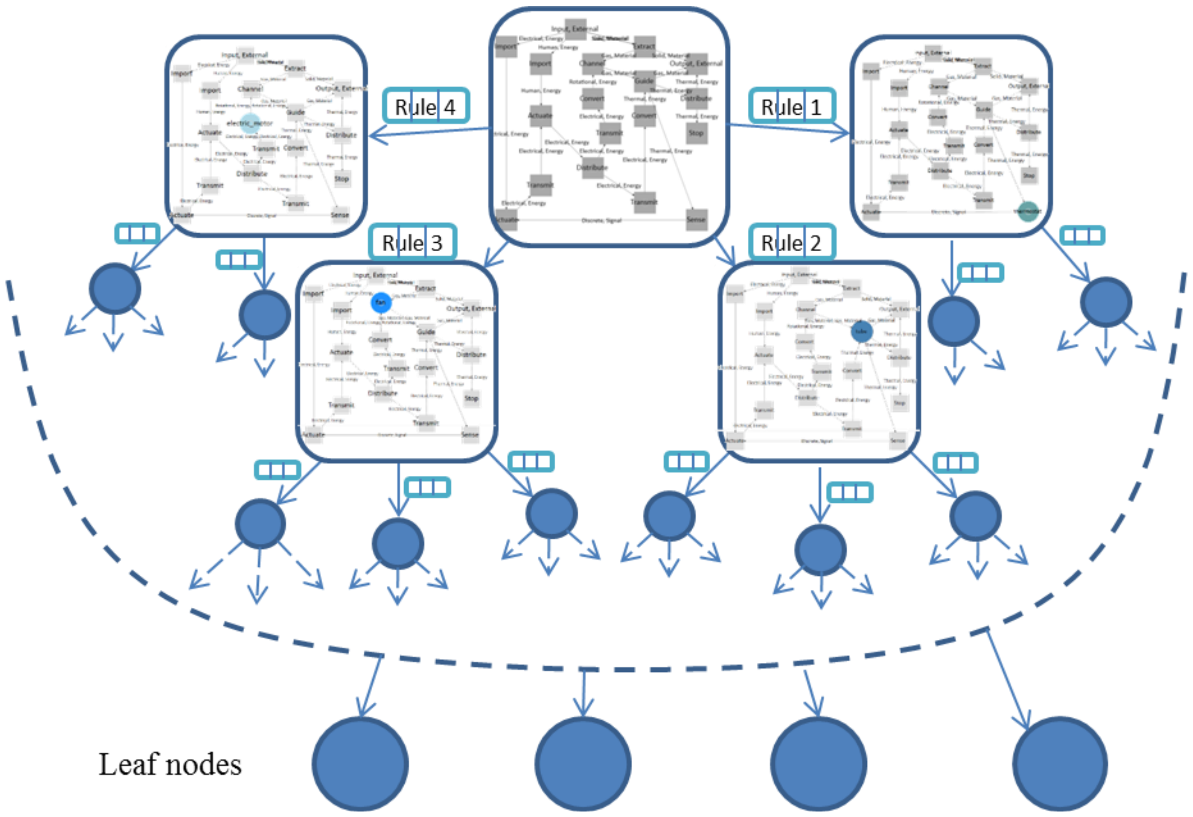


Figure 3 Tree based generative grammar method to find solutions starting from a seed node

STOCHASTIC ALGORITHM FOR TREE SEARCH

An issue with the graph grammar based representation is the potentially large number of design alternatives as the number of solutions of the tree is exponential in the order of the branching factor. At level n, the number of solutions considering average branching factor b is b^n and so exhaustive search techniques such as breadth first search and depth first search will lead to an exponential time complexity to find the target solution. A more efficient search process must therefore arrive at a solution in polynomial time. Any method that involves scanning a constant fraction – such as half of the search space still takes exponential time. An intelligent approach that learns certain characteristic of the tree to narrow down the search process can eliminate a major portion quickly.

Another issue with generative grammar based optimization is that while generation, the nodes and arcs in the graph (phenotypes) cannot be changed directly, but it can be modified by changing the rules sequence applied on the graph (recipe/genotype). This means that during generation of the candidate from the seed to the final solution, there is always a list of recognized rules but there is no preemptive vision for the type of solution that will be generated.

More advanced tree search methodologies such as Dijkstra's algorithm or A* or IDA* are implemented by forming a heuristic that determines at each level of the tree how close it is to the solution. Without a heuristics function, the closeness of a partial solution in a tree to the target cannot be evaluated without exploring all possibilities. Genetic algorithm is an alternative that finds the final solution by evaluating only the leaf nodes of the tree or the complete solution. However the problem has to be represented as a bit string which is not an intuitive approach to all problems and the possibility of generation of infeasible solutions has to be handled by using constraints (explicit or implicit) each of which impact the speed of the search process [12]

The above search techniques are designed for use in the case of a graph search and do not use any information from the generation process of using grammar rules for generation. Given these information, a search methodology is described that builds upon a technique the Interactive Stochastic Search [20] described in the following section

Rule Based Interactive Tree Search

Rule Based Interactive Tree Search (RBITS) [20] implements a search methodology that scans a very small portion of the tree in order to build knowledge to find the best solution. The search process relies on the rule based generative nature deriving a correlation between the genotype and the phenotype representation of the solutions. The basis of the method is that such a correlation exists thereby selecting specific options during the generation (creation of the genotype) to make a predictable change to the quality of the solution (phenotype). The methodology used is to evaluate a set of candidates and trace back this rating to the rules that were used for generation. Distinction is also specifically made for candidates that share rules used at the same level or used in a similar context. This information is built from each candidate that is generated and evaluated which leads to a wealth of knowledge about the rules and options used. This can be used to guide the search to the optimal solution.

Knowledge Repository

The RBITS method uses a repository for storing information called Rule Knowledge (RK). As candidate fitness ratings are broken down to constituent rule ratings, they are stored in the rule knowledge. This is a table that keeps growing with every additional candidate that is evaluated and added to it. Each rule present in the recipe of the candidate is added as a new row along with the previous recipe and the fitness value of the candidate. Table 1 shows an example of rule knowledge that contains the information from two candidates, one generated with the recipe 4, 5, 2, 3 and has a fitness rating of 2.0 and the other candidate with a recipe 5, 1, 2, 3 and a fitness of -3.0.

Rule #	Previous Recipe	Fitness
4		2.0
5	4	2.0
2	4 5	2.0
3	4 5 2	2.0
5		-3.0
1	5	-3.0
2	5 1	-3.0
3	5 1 2	-3.0
(Extends as more candidates are added)		

Table 1 Snapshot of rule knowledge as used by RBITS

If using a particular rule results in a candidate with both high and low ratings, it can be inferred that the rule does not play a part in the candidate rating and need not be considered for future candidate generation. If a rule results in candidates with consistently high rating, then it needs to be identified as a rule with high fitness. This property is easily achieved by averaging the fitness values of all rows where the rule appears.

More information can be extracted from rule knowledge. In the scenario that a particular rule has an average fitness of 0, it means the rule contributes positively and negatively in different scenarios. It is possible that the rule applied at a specific level during generation causes this variation. It could be most popular when applied at the beginning or the end of the generation process (such as an initiation or a termination rule) and information such as this can be extracted by averaging the fitness values of those rows of the rule where the recipe is a particular length. In the failure of level fitness, a rule could show consistent behavior given a particular recipe of rules that precedes it. This is done by matching the entire recipe for a rule and averaging the fitness values of these rows.

These give us a lot of information about the fitness of a rule. However, the fitness need not be accurate until a required number of rows for averaging are obtained. The accuracy of the fitness is determined by the popularity of the rule. Fitness and popularity information is used to direct the search process towards finding a variety of candidates, evaluating them to get a good picture of rule properties.

Selection of an option

Figure 4 shows a tree search process using the RBITS approach. At every level of the tree some options are recognized in the graph and one of them is selected and applied giving a partial solution. This process continues until there are no more options recognized which gives the final candidate. Choosing which option to select is the critical decision that leads to the required candidate. To make the decision at each level, fitness and popularity values of the option is used. Using fitness data alone is equivalent to directing the search process to select the best candidate by using the options that have high fitness values. However, this would lead to a case when a select set of options are repeatedly rated and to account for this, options with low popularity are also selected. This becomes a multi objective decision problem to balance between exploration and exploitation where a knob B is used as shown in the following equation where f is the fitness and p is popularity. To suit the nature of this equation, fitness and popularity values are normalized such that their value is between 0 and 1. Options that are least popular are preferred to be selected and therefore the most frequent option is given a popularity of 0 and options that have not been given a rating before get a value of 1. The minimum and maximum values for fitness are selected as the extreme ratings that have been assigned to candidates generated so far. Minimum popularity is set as 0 and maximum is the maximum number of times an option has occurred in the knowledge database.

$$u = Bf + p(1 - B) + u_{min}$$

Using this relation, a metric u is calculated for each option recognized and based on this a choice is made. The knob B is used to change the selection from being based on fitness or popularity values. When B is 1, fitness value alone is given importance and candidates are generated with a few set of options that are considered best and when B is 0, all options are explored increasing the variety in candidates explored. Also a minimum probability u_{min} is added to the above equation so no option gets a preference of zero.

Selecting the value of B is critical to our search process and dictates the quality of results obtained and the time taken for convergence. The value of u_{min} is set as 0.003.

Stochastic choose

In this search process, only a small part of the tree is scanned. It is implicitly assumed that the knowledge gained while scanning this portion of the tree is enough to get the overall picture. However, in multimodal cases the solution need not be guaranteed to converge in the global optimum as the search process can exploit in the direction of a local optimum. Similar to methods such as genetic algorithm or simulated annealing where there is always a finite probability of selecting a wrong search direction so that the search space can be explored for possibility of local and global extremes, a stochastic nature is introduced in this process such that even if an option might have the highest preference, all options get some chance of being picked. The preference of an option is now converted to probability of selecting the option and the option of highest preference gets the highest probability but is not necessarily selected. An example is shown in Table 2. Among the three options listed, option A which has the highest preference gets a 75% probability of being selected but there is no guarantee that it will be.

Option	Preference / Probability of selection
A (Apply Rule 1 at location A)	0.75
B (Apply Rule 1 at location B)	0.12
C (Apply Rule 2 at location C)	0.13

Table 2 Selection of option based on preference

An additional level of control is applied to the process as the stochastic nature of the process might not be suitable to all problems. To modify the level to which the process is stochastic or deterministic another parameter p is used as shown in Equation (1).

$$Probabiltiy(option\ i) = \frac{u_i^p}{\sum_{i=1}^{N} u_i^p} \tag{1}$$

The above transformation is called the p-norm. If p is 0, the probability of all options reduces to the same value as the u values are raised to their 0th power. This means that all options get an equal probability of getting selected and the process is random inconsiderate of the popularity and fitness value of the option. When p is ∞, it can be shown that all the u values reduce to zero except the highest value that becomes 1. This is a method of selecting only the option with the highest probability. The process becomes deterministic in always selecting the option with the highest probability. When p is 1, the probability values are retained without modification and the process becomes stochastic where there is some element of randomness but options with a higher u values have a higher probability of being selected. This is a middle ground between the process being completely random and being completely deterministic.

This parameter significantly affects the quality of the final solution and also the speed of convergence of the search process as will be shown in the later sections. In order to study the effect of changing from random to stochastic to deterministic, p is sampled such that are an equal number of points between 0 and 1 and between 1 and ∞. This implies that the gap between the points increases or in other words, they are generated with an increasing step size. There is also the issue that computers cannot handle values close to ∞ and a double precision value can hold only values between 10^{300} and 10^{-300}. Therefore, for a practical implementation, a parameter Q is selected that has a one-to-one mapping with the values of p and such that as Q increases with a constant step size p will increase in larger steps achieving the required sampling characteristic. Q is converted to p using a polynomial relation as mentioned in Equation (2). Q value of 0 and 1 correspond to p values of 0.003 and 300 and Q value of 0.5 corresponds to P value of 1. The effect of B and Q values on the convergence of search process is described in detail along with results from the PV optimization problem.

$$p = 0.003 + 299.997 * Q^{8.23314} \tag{2}$$

Figure 4 illustrates graphically how a RBITS process works. The optimal candidate in the search process is marked with a '$*$' and is to be obtained by following the options marked by the dotted circles. The candidates marked in black are those that are evaluated by the process and those in grey are options left out for reasons of low fitness or popularity. The process need not be searching anywhere close the ideal candidate in the tree. Rule knowledge is built while candidates 1 to 4 are generated and evaluated and using this information, the best candidate is selected.

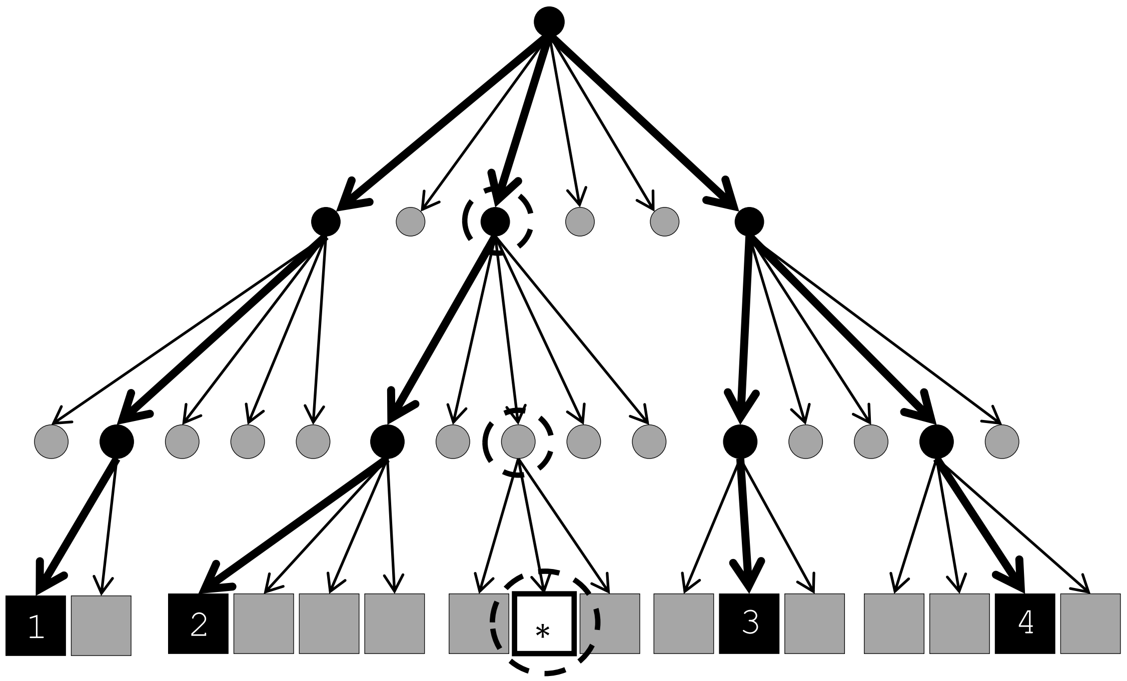


Figure 4 RBITS search process. Candidates evaluated and added to knowledge repository are highlighted in black. Dotted arrows represent option with highest probabilities

Necktie Experiment

The above process has been experimented with an example problem of finding the best method to tie a necktie [20]. The reason for choosing this is that there exists a clear set of rules to tie a necktie that can be represented in the form of graph grammar. The final solution obtained is evaluated by an interactive process where a human designer comparatively evaluates a batch of candidate solutions qualitatively. Qualitative rating is converted to an approximate quantitative rating that is used as fitness of the candidate which is applied to the rules that formed the candidate solution and is stored in the rule knowledge. The process of selecting new candidates and evaluating them was done for a preset 30 candidates and the solution converged to 4 solutions from a batch of a possible 65000 solutions or in other words, the result obtained was at around 99 percentile.

Modification to the RBITS process

In the following sections, modifications made to the RBITS process are described to suit a different experiment and the optimizations made for the same.

Evaluation Mechanism

RBITS uses an interactive evaluation mechanism. A dialog box is shown with a small batch of candidates where the user supplies a comparative rating of -1, 0 or 1 between all pairs of candidates in the batch. The dialog box shows a square matrix with each candidate representing a row and a column in it. Each element in the matrix corresponds to the rating of the candidate represented by the row in comparison with the candidate represented by the column and takes the value -1, 0 or 1. It can be easily concluded that this is a skew symmetric matrix and therefore the user needs to fill in only one half of it. The individual rating for the candidate is obtained by summing up the values in the row or the column corresponding to the candidate. There is obviously a limit on the maximum and minimum rating a candidate can get. For example, if 3 candidates are shown, the rating is always between -1, 0 or 1 or with 5 candidates, the rating is between -3 and +3. This works very well for an interactive evaluation, where human designers can be only expected to arrive at qualitative and relative rating between solutions presented to them. This is easily improved in the case where a human designer could be replaced by automated evaluator that provides an absolute quantified rating. A rule with a relative fitness value of -1 is never considered for future evaluations unless exploration is set high (B is set to value close to 0) and the stochastic process "happens" to pick it up for further evaluation. With absolute rating put into place, rules are picked based on the actual rating of the candidates which leads to a more informed decision.

Candidates are carefully selected to be shown in the dialog box. This is done by using rule knowledge into creating candidates with rules having appropriate fitness and popularity values. Each time a dialog box is presented, it is an opportunity for the process to carefully select candidates that will improve the quality of the rule knowledge faster. If there is a limit on the number of candidates to be evaluated, increasing the number of candidates in a batch implies lesser number of dialog boxes. A more desirable option would be to reduce the batch size but in a relative rating scale this implies candidate ratings will vary within a smaller range. In an absolute rating scale however, this restriction does not apply and the batch size can be reduced to simply one candidate per batch.

Option information

Rule knowledge is retrieved at three levels: total, level and context. This is a hierarchy of information where context is the most specific form, explicitly mentioning the ordering of rules that precedes the option. If a candidate c_1 has fitness r and a recipe with rules R_1, R_2, R_3, R_4, R_5, then context rating for rule R_4 for applying on a candidate c_n with recipe R_1, R_2, R_3 is also r. On the other hand, level rating requires that options be repeated at the same level of the tree with or without a matching previous recipe. Thus if candidates c_2 and c_3 with recipe R_1, R_2, R_3, R_4 and $R_{10}, R_{11}, R_{12}, R_4$ will both contribute to the level 4 rating for rule R_4. Total rating is the highest level in the hierarchy where the ratings of all candidates generated with rule R_4 add up to make this value. Figure 5 explains the hierarchy where there are 10 entries for context fitness values exist for Rules 4 and 5. Level fitness is obtained by aggregating certain entries from context fitness and total fitness aggregates the fitness rating of certain elements from level fitness.

This hierarchy of option information is suitable for the neck tie experiment that was considered for the original RBITS approach where the order of rules and options matter the most. However, consider the rules shown in Figure 6a, one to create an arc in the East direction and another rule to create an arc in the south direction. Figure 6b shows a seed graph in which the generation process happens in two branches where the two rules are applied in different orders to get the same solution. In this case, the fitness of the rules is independent of the level at which they are applied and in the example problems that is used in this work, the PV array optimization problem and the product assemblies problem, it will be clear that they behave similar to the seed graph in the figure. Thus the level fitness for these rules is 0 and therefore not useful for the search process. Context popularity and fitness values determine the popularity and fitness value for a rule given a particular recipe that precedes it. However, in the case shown in Figure 6, it is shown that the order at which the rules are applied do not matter and the context fitness of the rules will not give any additional useful information. Both rules perform equally independent of whether they were preceded by a rule or not and thus context fitness is also not useful for this case.

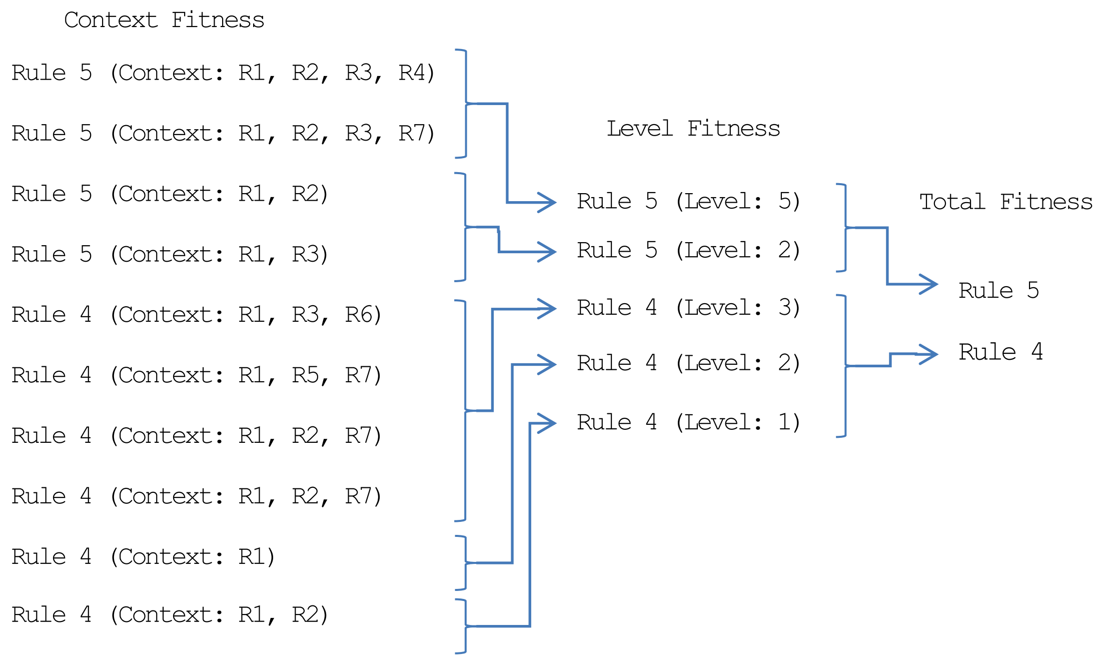


Figure 5 Hierarchy of Rule knowledge. Each level aggregates a set of elements from the previous level.

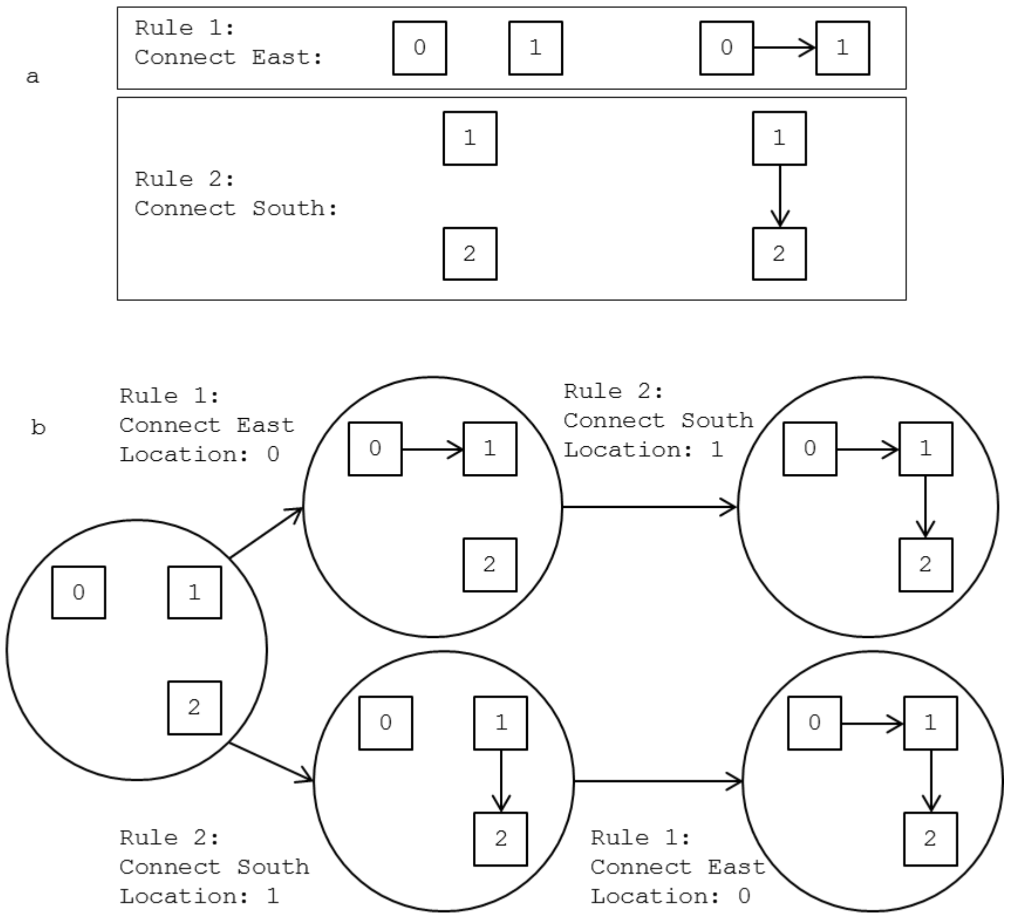


Figure 6 Rules applied to same location but in different order lead to same candidate

The nature of the problem in such a case is that rules are dependent on location and characteristic of the final solution is the location where each rule is applied. Location fitness and location popularity are thus new quantities introduced in this methodology and they apply to options rather than rules. Option contains both rule number and the location in the graph where it is going to be applied. Ratings of candidates that use the particular rule at the specified location add up to the location fitness of the option. The RBITS search process and the above modifications are written in C#. In order to make the process of manipulating graphs easier, the code is packaged as a plugin for the GraphSynth [11] software.

Solar panel optimization problem

PV cells are the most important part of generating power from solar energy and the fundamental blocks of a solar panel. Thousands of PV cells are used in a solar power generator to create a required voltage level since individual PV cells are capable of generating only potential difference of the order of 1V [19]. These solar cells are usually connected in series to maximize the voltage output. To maximize the power output however, control techniques such as Maximum Power Point Tracking (MPPT) are used that controls the load on the generator in order to run it at the most optimum voltage for the maximum power [21].

One of the problems with renewable sources such as solar power is unreliable input conditions that lead to varying power output. Shading conditions are often a problem in cases such as roof top power generator where even a few shaded cells could lead to drastic power reduction. Partial shading not only affects the shaded PV modules but all the modules that are connected in series and control techniques such as MPPT cannot tackle this issue. By modifying the connectivity between PV modules, a more reliable output can be obtained but at the cost of a reduced potential difference. Topology optimization has been shown to optimize the overall power generation especially under partial shading conditions [22]

Previous work on topology optimization has been done as a comparative study between two different approaches - using graph grammar and genetic algorithm (GA) [12]. Results of the study show that genetic algorithm approach does not converge on a global optimum and performs worse than a graph grammar approach. Using GA, the problem is represented as a bit string and transformation of the problem to and from the bit string is not stored explicitly. Problem constraints are represented as penalty functions or explicit and implicit constraints which are much harder to encode and prove the validity. In graph grammars, the problem is represented as seeds and rules that encode the problem and the constraints and the graphical and intuitive nature can easily help understand and breakdown the complexity of the problem. Graph grammar techniques can also be combined with complex search techniques for an efficient search process.

Representation

A solar panel power system consists of several interconnected photovoltaic modules and a PV module consists of several interconnected PV cells. PV modules are arranged in an array and connected in a configuration that maximizes the power output such as the one shown in Figure 7. The cells within a PV module are connected in a predefined fashion for mass production purposes but the connectivity between PV modules can still be customized.

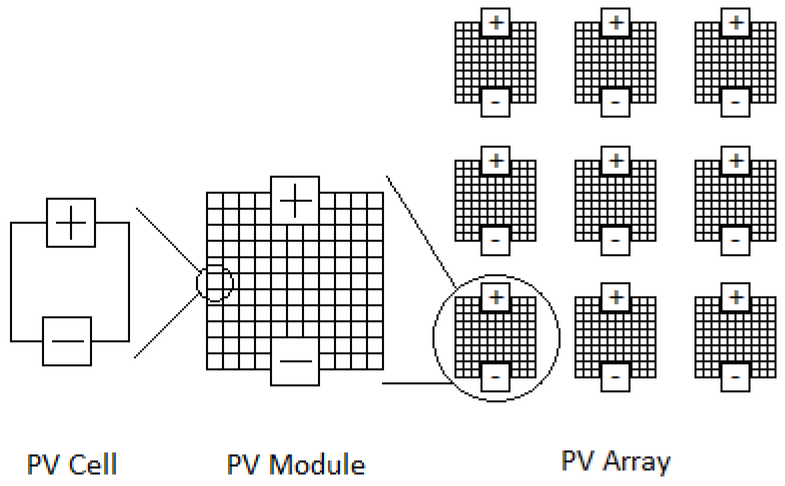


Figure 7 Construction of a PV array. Connectivity between modules of PV array is open to be optimized

Each PV module is modeled as a voltage source. All modules can be connected in series to maximize the voltage or in parallel to maximize current or reliability. A combination of series and parallel connections is usually preferred to obtain an optimum considering different objective functions. As the possible topologies could be very high in number, the problem is limited to the following scope

1. Nodes are connected only to neighboring nodes and bypassing nodes is not allowed
2. Nodes are connected only in the North, South, West, East directions. Diagonal connections are not permitted.

3. A node is connected to the positive terminal if there are no outgoing connections
4. A node is connected to the negative terminal if there are no incoming connections
5. Cycles between two modules are not allowed.

With these restrictions in place, the number of possible topologies is restricted and the problem representation becomes much more manageable. As node placements are not modified, in both methods, genetic algorithm and graph grammar, the aim is to optimize the connectivity.

In the GA approach the PV array is encoded in a bit string of length $6n$ where n is the number of PV modules. Each bit defines the connectivity of a node to one of its four neighbors or to the global positive or the negative terminals and any bit string of length $6n$ is a possible solution. Explicit constraints, implicit constraints or penalty functions rule out impossible configurations. In the graph grammar representation, the problem is split into seed graph and rules. The seed graph consists of nodes representing each PV modules and the position of the node represents the placement of the module in a 2D space. An array of nodes would therefore represent a PV array that is placed identical to the graph representation. Connectivity between the modules is represented by arcs and labels that connect the nodes in 4 directions (North, East, South and West). Connectivity to the global positive and negative are derived implicitly. Table 3 explains the relation between the label on a node and the connectivity that it implies. A missing connection for a node represents a connection to the global positive / negative terminals as shown in the row 5 of Table 3.

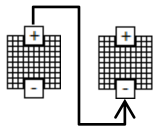
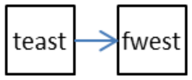
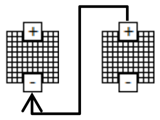
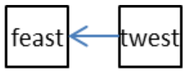
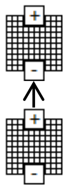
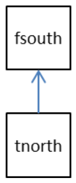
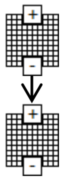
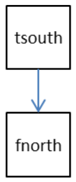
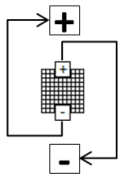
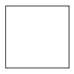
Connectivity	Graph grammar representation
	
	
	
	
	

Table 3 Connectivity information represented using labels and arcs

The connectivity representation can be extended to an entire graph as shown in Figure 8 where the topology on the left are represented as graph grammars in the right side. A node can have more than one label indicating more than a single connection to it and which is represented in Figure 8 e and Figure 8 f.

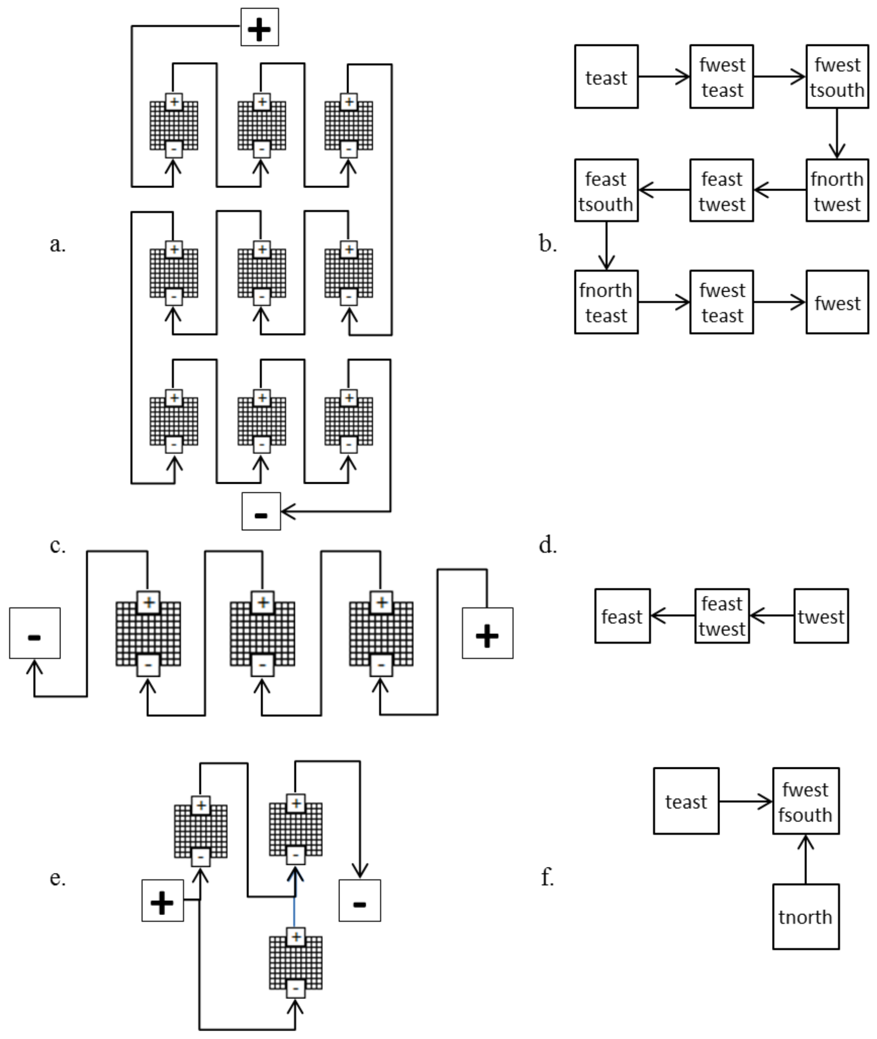


Figure 8 Graph grammar representations of certain connectivity patterns

To create these topologies 5 rules have been identified that are shown in Figure 10. In the above set the first 4 rules look for nodes that are not connected in a particular direction (N E S W) and create an arc in that direction. In the *L* part of the rule, the first node contains the label "empty". The second node negates the label "empty" and "twest" indicating that the node should not already be connected in the same direction preventing the creation of overlapping arcs. Rule 5 allows the generation of a node that is not connected to any of its neighbors. This is done by removing the label "empty" preventing the node from being recognized by the other rules. Nodes that are not connected to neighboring nodes are considered to be connected to global positive and negative terminals. The recognition of adjacent nodes here is based on a position match as nodes do not have any other distinguishing feature. This implies that cases such as the one shown in Figure 9 cannot be represented by the rules due to the uneven position of the node on the top that prevents it from being recognized by any connectivity rules.

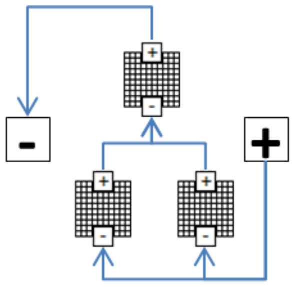


Figure 9 Nodes not positioned on a grid cannot be represented in this approach

To create topology solutions, an empty seed without any connectivity information and the rules in Figure 10 are used. An example seed is shown in Figure 11 that contains 9 nodes. Each node contains a label "empty" indicating that no rule has been applied on that node and it does not have any connectivity

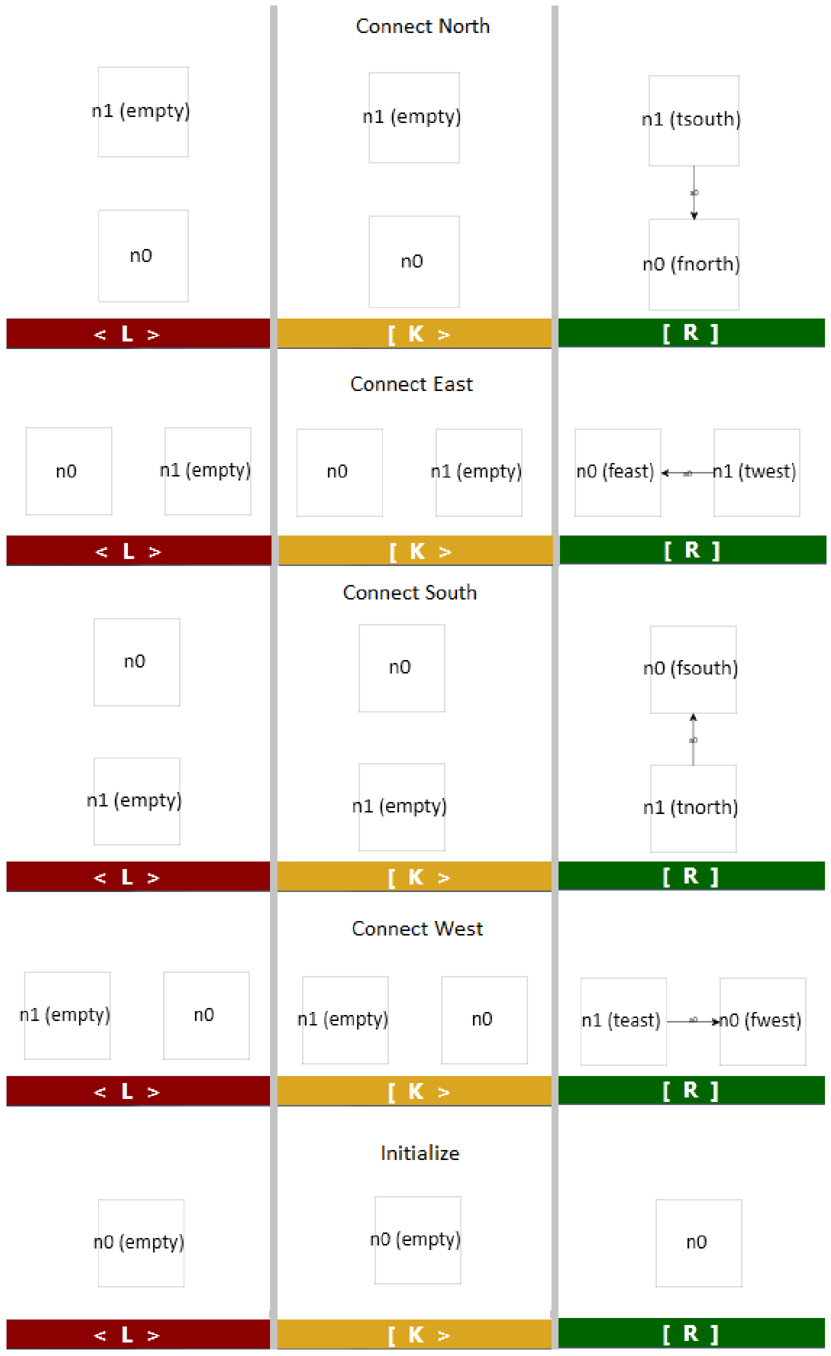


Figure 10 Rules for creating connection between nodes in a seed graph. Fig a, b, c, d show rules to create arcs in N, E, S, W directions. Rule creates arcs to connecto global positive/negative terminals

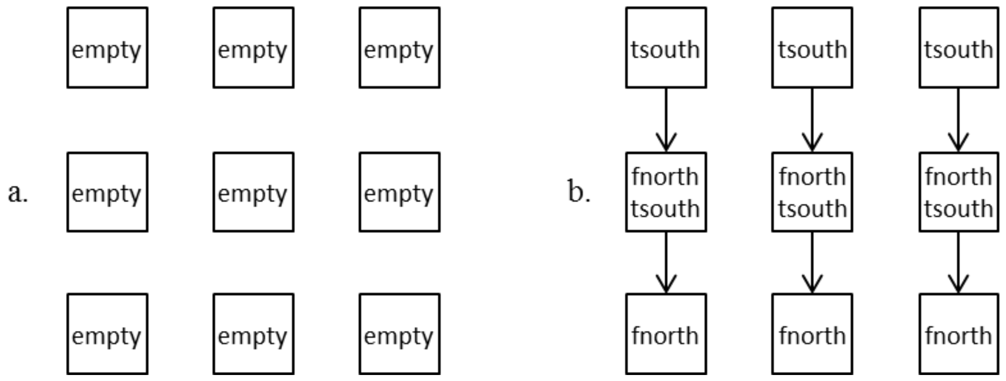


Figure 11 Example seed graph for a 9 node PV array. A possible solution is shown in figure b

The rule set is still highly flexible and allows for a variety of solutions. The approximate number of results for the case of a square grid with 9 nodes (3 x 3 nodes) is around 3^6 different solutions. The accurate number of variations of the solution is to be calculated on a case by case basis due to the restrictions posed by the rules such as preventing the possibility of forming cycles etc. The rules are highly generalized and this is helpful to validate the search process's effectiveness. The representation has reduced the PV optimization problem as a topology optimization problem. The problem is to find the best configuration of arcs that would prove most optimal. As this optimal configuration can be unpredictable, an arbitrary configuration is selected and experiments are conducted to see if the search converges to this solution.

Specifications of the problem

The size and complexity of the problem is determined more by the number of PV modules and less by the shape of the array. The problem is not restricted to a square or a rectangular grid of PV modules but at the same time, the layout of the problem can be such that even with a large number of nodes, the variety in the solution set can still be reduced. For example, in the case shown in Figure 9 the nodes are placed at locations such that none of the rules can recognize the presence and create a connection thus reducing variation in the solutions. When the nodes are placed in a square or rectangular grid, the number of different design solutions depends exponentially on the number of nodes and the number of rules applicable. In the experiments, the number of rules is kept a constant (5 rules) and the size of the seed graph is changed to modify problem complexity.

Evaluation mechanism

Evaluation of a solar panel array is complex. The evaluator must convert the graph that is generated into an electrical circuit that modifies each PV module into a cell, resistor and a diode triplet representing a single PV module. Calculation of effectiveness for a particular configuration must consider different shading patters to find the power generated. In a no-shading case, the best power is obtained by the configuration shown in Figure 12a where all nodes are connected in parallel. However, under an unknown shading condition any arbitrary topology could be obtained. The optimality of the solution cannot be determined preemptively based on heuristics and no assumption about the nature of the solution is made such as one shown in Figure 12b.

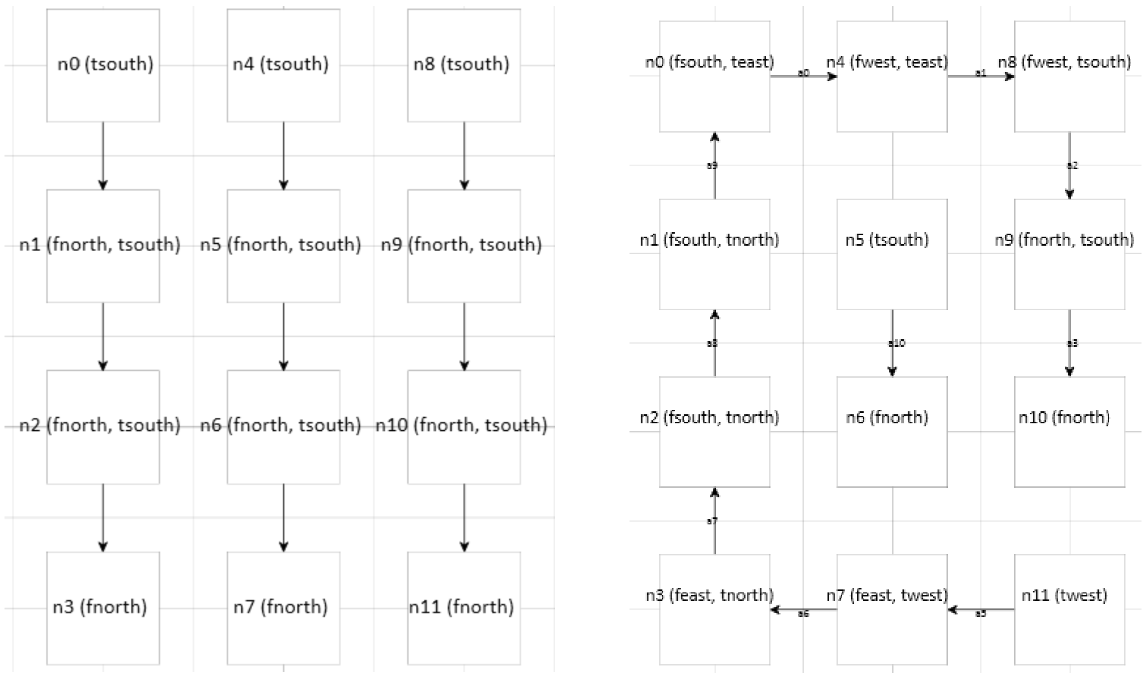


Figure 12 Different possibility of target solution for solar panel optimization problem

The search process is not given any information about the nature of the shading condition. This saves the need to build complex evaluators. To test the effectiveness of the new method, we create an evaluator that assumes a predefined solution topology to be target and rate other solutions based on its distance from this target solution and this is done by computing the distance between their corresponding vector representations. The vector representation must be such that distance between two identical graphs is zero and as the number of modifications done to the graph in terms of labels and arcs increases, the distance between the graphs must increase; distance between two vectors is always a positive number. For this problem, the graph is converted to a vector of length six times the number of nodes in the graph is used where each node is allocated 6 elements to store the connectivity information. The first 4 elements represent the element connectivity to its immediate 4 neighbors. The elements represent the neighbors N, E, S, W in order and take values of 0 or 1 indicating whether it is connected or not. The fifth and sixth element represents that it the node is connected with the global positive and negative terminals. The vector is therefore a series of 0s and 1s that represents the graph and the use of this form of a representation is that any change to the graph is reflected as change of 0s and 1s to the corresponding elements. Removal of an arc for example, will result in one of the cells set to 0. It is important to note that in this representation, a vector of the right size with arbitrarily selected 0s and 1s need not represent a valid design. In order to get a valid design, the candidate must be generated from the seed graph by applying the sequence of the required grammar rules and then converting to the vector representation. The distance between two vectors is calculated by obtaining a vector difference of the two vectors and reduced to a single value using a 2-norm or in other words, the Euclidean distance between the vectors is used.

Optimization problem

Due to the Euclidean nature of the distance metric, the objective function becomes unimodal where the magnitude of the slope is 0 at the optimal solution. This is done to test the validity of the search process and establish metrics on convergence rate. Traditional optimization techniques work by selectively scanning several points in this multidimensional space. In this problem, each point is a vector representation of the candidate design and from the argument made in the previous section, selection of an arbitrary point by the optimization technique need not result in a valid design. Thus methods such as steepest descent do not work here and we use a process that directly works at the level of graph grammars.

Results

Convergence of an optimization problem is often indicated when the objective function does not change significantly from small perturbations about a given point. The point is returned as the best solution. Convergence can also be correlated with convergence of the state of the knowledge repository. Ideally a final candidate is returned when the repository has attained enough maturity and contains accurate information about the fitness of the options. The best candidate for a given repository can then be generated by starting from the seed graph and applying the options with the highest fitness at each level with no regard of popularity (i.e. $B = 1$) and no randomness (Q = 1). This represents a deterministic and exploitative search of the space. After the repository has converged, adding additional candidates to the rule knowledge will yield only a marginal change in the option fitness values and therefore the best candidate generated would not change significantly. This can be used to detect the convergence of the process. A candidate can be generated with values B and Q as 1 at every step and evaluated. Once the rating of this candidate stabilizes (i.e. remains within a certain tolerance) it can be assumed that the knowledge repository has attained a stable state and the process has converged.

Figure 13 shows the variation in the rating for a seed of size 8 nodes and B and Q values of 0.2 and 0.8 respectively. It can be seen that the search algorithm eventually converges to find the candidate of rating 0 implying the distance of the candidate from the ideal solution. To prove the reliability of the process despite the stochastic nature, the same graph is recreated in multiple runs as shown in Figure 13b and Figure 13c. To obtain a convergence curve that better summarizes the effectiveness of the process, similar graphs are created for 20 runs and they are averaged over the iterations. The resulting graph is shown in Figure 13d. Several conclusions can be made from studying this graph. As it represents the average of several runs of the search process, some unfortunate runs that do not converge at all or converge on a solution far from the ideal solution will have a negative impact on this. It measures the convergence while considering the stochastic nature of the process. Our aim is to solve the solar panel problem for different sizes and complexities and compare the results. However, before we do this, we seek to optimize the values of parameters B and Q such that we can get the best convergence graph. To do this, the experiment is run with different sets of values for Ba and Q. Value of B is changed from 0 to 1 with a step size of 0.2 and Q takes the values 0.0, 0.3, 0.5, 0.7, 1.0. The reason for Q to take a slightly different set of value is that Q=0.5 is a significant point at which the process becomes stochastic. The following sections describe the effect of modification of these two parameters with the results of the experiment.

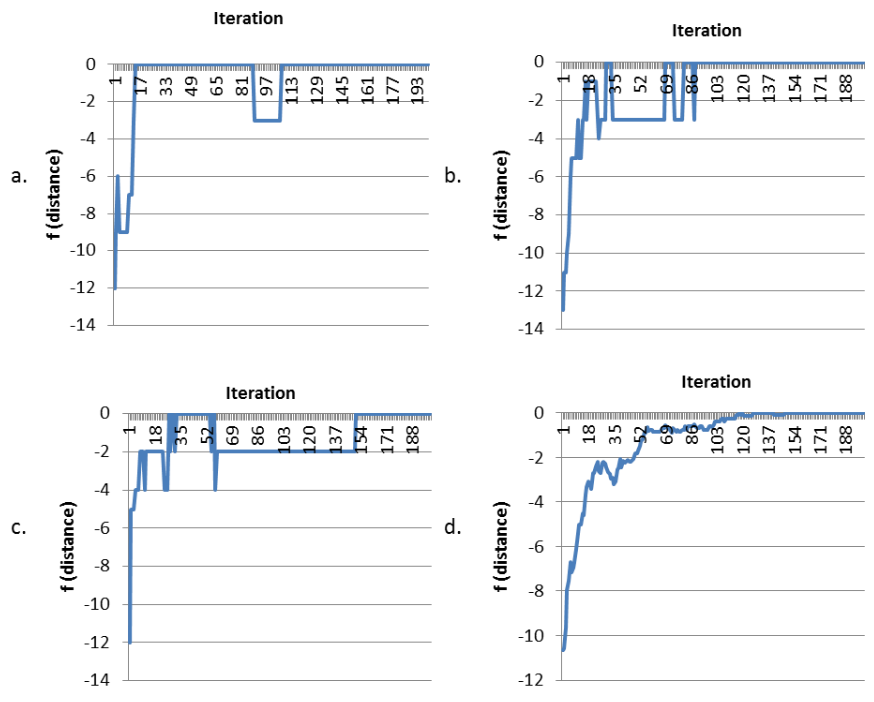


Figure 13 Convergence of rating vs. number of iterations. Graphs are generated with same settings but different stochastic runs produce different output (a, b, c). Plot d shows an average of 20 such runs (a, b, and c plus 17 more).

Experimenting with Q

A Q value of 0 represents a completely random process that does not use the knowledge and instead selects an arbitrary choice at every generation step (recall discussion in page 19). From the point of view of an optimization algorithm, this leads to an uninformed search where the process tries different candidates without making an intelligent decision on the direction of exploration. Evaluating these candidates will still add to the knowledge repository but the quality will be low as repeated options will be added. However, as more candidates are explored, eventually all options will be rated and their fitness values will be stored in repository. This implies that convergence can be arbitrarily delayed and there is no guarantee on number of evaluations. We see this in the case shown in Figure 14a, that within 200 iterations the process does not converge on a single candidate and even within the 200 iterations, the progress is not smooth compared to the other graphs.

Q value of 1 implies a deterministic process. In this case, the process is highly dependent on the knowledge repository and uses that to make a choice of option at every step during candidate generation. Although this might still lead to a search process that converges very quickly, the problem is that the process shows a very slow learning curve and it is highly dependent on the starting point which affects the quality of the final solution. The options in the first candidate selected by the process get the highest rating due to normalization. Since there exploration is minimal, the process repeats the same options repeatedly and only occasionally creating candidates using a new set of options. The slow learning curve is shown in the convergence graph in Figure 14 The convergence value that we see is from the average of arbitrary starting value that was obtained in each of the 20 runs from which we can conclude each of the process has been highly deterministic

Q value of 0.5 is neither deterministic nor random and from Equation (2), this implies that the option probabilities are preserved. In this case, the process considers the rule knowledge as a valid source of information for predicting future direction of search but assigns a level of error to the knowledge and explores the design space arbitrarily. This leads to more unexplored options being included in the knowledge repository within a lesser number of repeats in exploring existing rule knowledge. In this case, the chooser uses the popularity value in addition to the fitness value. The result is a space explored more uniformly thus increasing the chance of a better candidate chosen at a much earlier iteration.

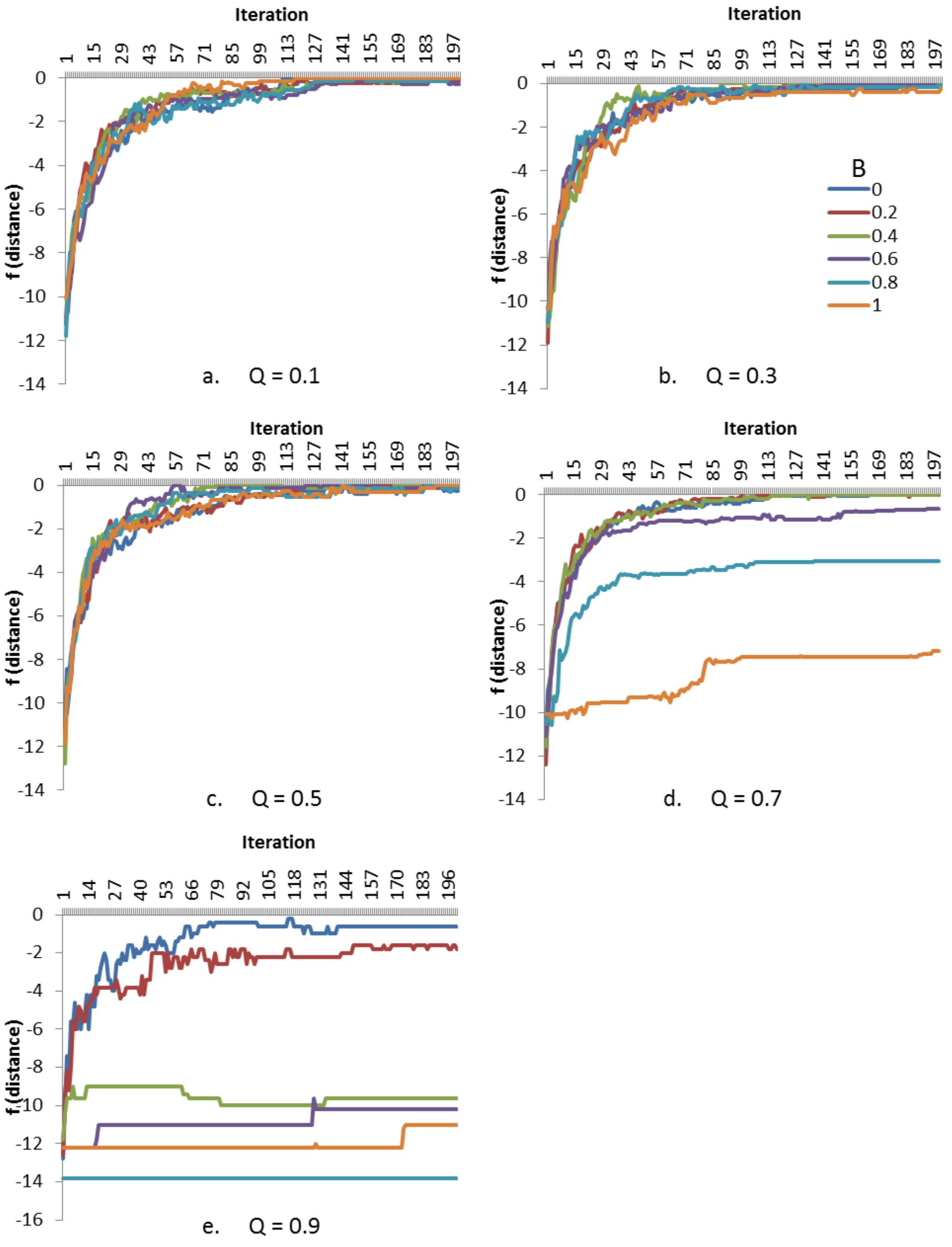


Figure 14 Progress of average rating vs. number of iterations with different sets of B and Q values grouped by Q

Experimenting with B

Q determines the stochastic nature of the process. B determines the exploration vs. exploitation of each option while choosing a new candidate. The search process characteristics come out when Q values ranges in the middle where the differences between the option fitness and popularity are taken into account. Using a B value of zero implies that we are wholly interested in exploring as opposed to exploiting what has been learned. Regardless of what fitness value each option has, only the popularity of an option is considered for determining the search direction. An option that has not been explored will always be explored at the earliest. Once all options have been explored equally, the process will try to continue by repeating options Due to the normalization that is done on the popularity rating, all other options will now have a relatively less popularity and the process continues to explore these less popular options until all options have same popularity and the process continues.

Whenever an option is explored again, the fitness rating potentially changes. As an example, we consider a case where the search process encounters the two candidates as shown in Figure 15, both of which contain the rule: Connect West applied at nodes n0, n4 and n8. When the search process, encounters the first candidate, it will assign a rating +5 for this rule-location pair. Due to exploration, the search process encounters the second candidate that also contains the same option and now the Connect West rule at location n0, n4 and n8 gets a rating of -5 giving a total of 0. The knowledge repository gets updated every time a candidate is explored thus increasing the accuracy and reliability of the fitness ratings of all options. Eventually the repository would converge on the fitness values and therefore converge on the best candidate generated. Even in the case of solving the solar panel problem, though there are only 5 rules to be evaluated, the number of options for each rule based on location is still a high number. For problems with more options (which can due to more number of rules or more applicable locations), this leads to more evaluations for the search process to explore all options till convergence is reached

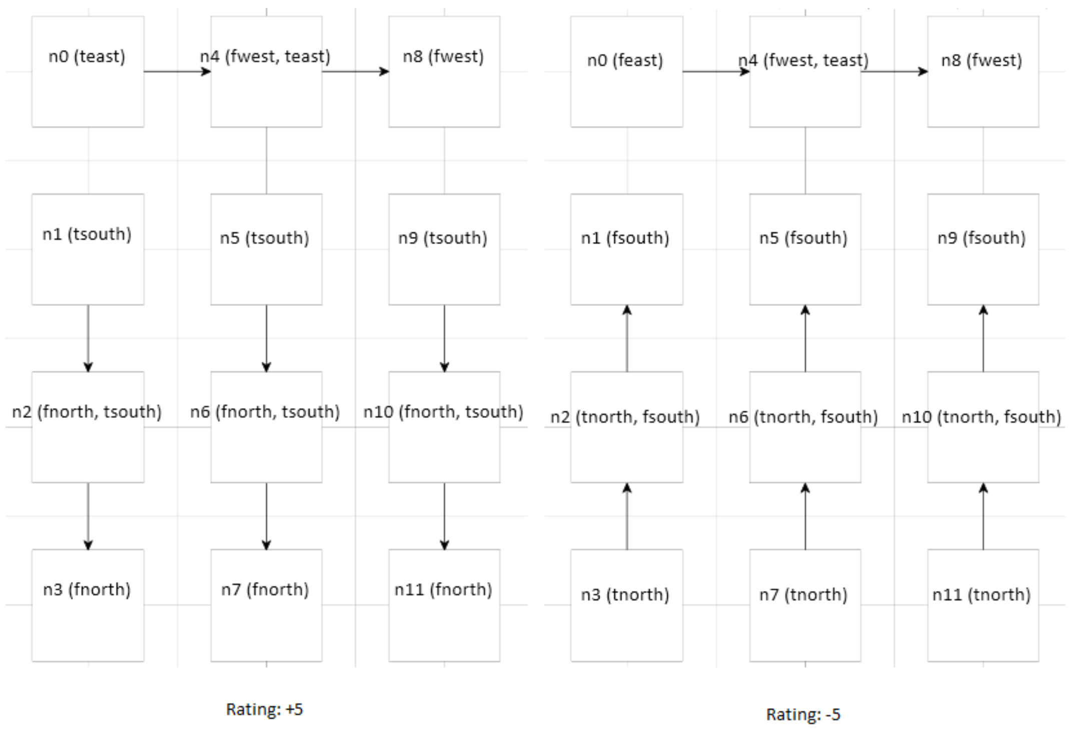


Figure 15 Candidates that contain some same options but have different rating

The parameter B is therefore increased to exploit more, meaning it does not waste computations on re-examining options that have low fitness rating. Arguing along these lines, if the value B is selected at the other extreme as 1, we get a pure exploit search process. The process now tries to exploit the set of best options found in the repository to get a better accuracy on their ratings. The set of options that are rated low in the repository are not considered as it is assumed they lead to a worse candidate. However, after each evaluation especially during the initial stages, the fitness rating for options might change drastically such that it completely changes overall the picture of the repository and previously unconsidered options are now used for generation in the next iteration.

Although exploration is still done, there is a possibility that options are still left unexplored. The process therefore need not find the best candidate and converges on a best candidate that can be found with the set of options that it exploited more. Results of this can be seen in Figure 16f where we find that search process converges but on candidates that that are far from the ideal solution.

If parameter B is chosen to be an intermediate value such as 0.6, the process assumes that the knowledge repository contains fitness ratings of moderate accuracy. In this case, options of high fitness are definitely exploited more and there is a low probability that options of low fitness values are not explored. The process neither completely explores nor exploits and therefore is better suited to finding a global optimum. The results can be seen in Figure 16c where the process converges to the ideal solution (global optimum) and converges in a reasonable number of attempts. Compared to graphs Figure 16 f and Figure 16a, this value of B attains a balance.

Selecting best values of B and Q

From the graphs that were obtained, the number of iterations to of convergence and the distance of the final candidate to the optimal are considered. Table 4 shows the variation of these two parameters with B and Q values. In order to converge quickly on the best solution, it can be inferred that the best values of B and Q are at neither extreme. Values of distance to optimal are more uniform even at low values of Q but take a long time to converge. The best values of B and Q can thus be chosen as 0.4 and 0.5.

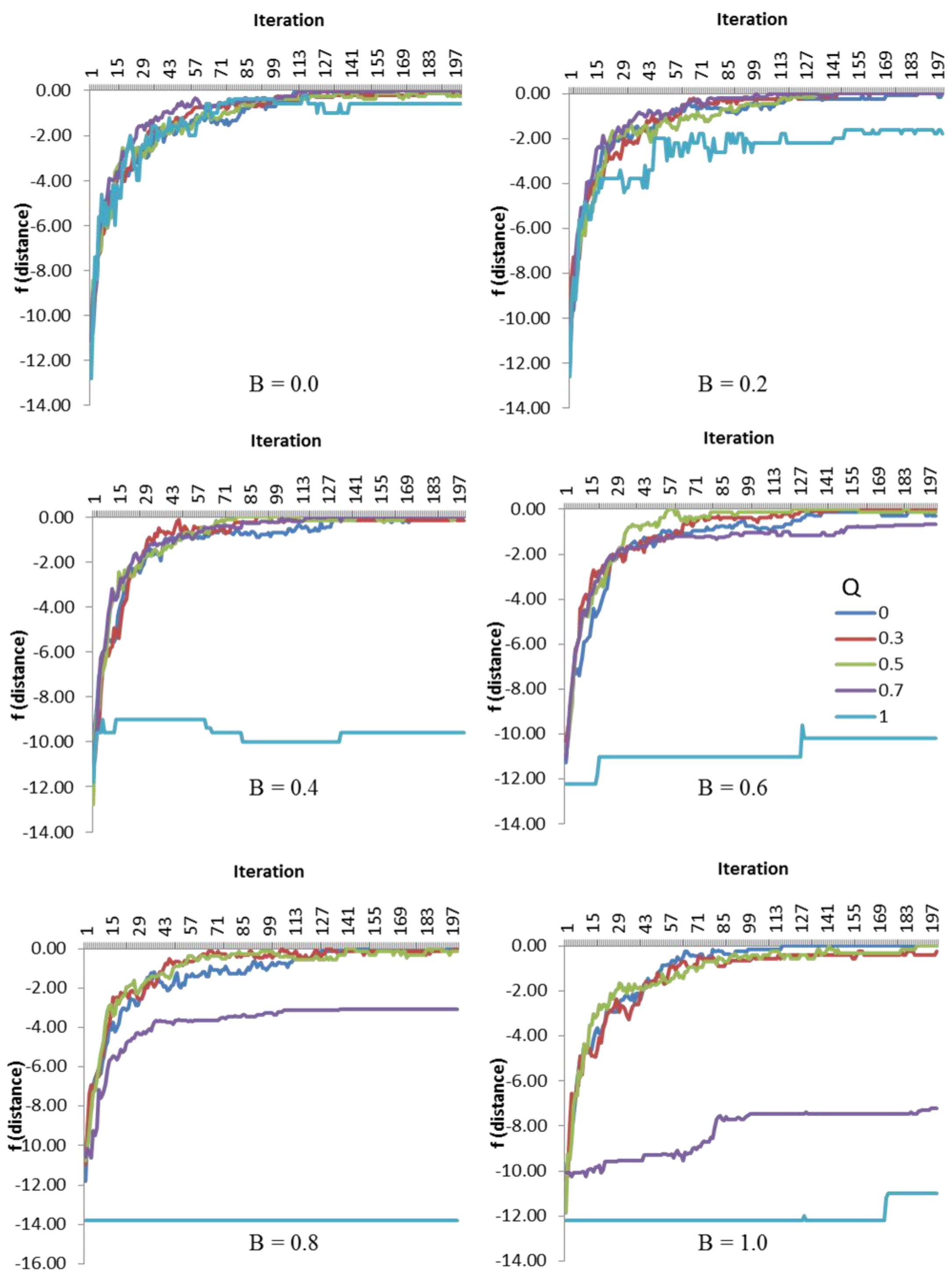


Figure 16 Progress of average rating vs. number of iterations with different sets of B and Q valuesgrouped by B

Iteration of convergence		Q				
		0	**0.3**	**0.5**	**0.7**	**1**
B	**0**	114	118	120	116	139
	0.2	168	129	120	98	165
	0.4	131	93	76	118	133
	0.6	131	131	83	156	127
	0.8	130	134	161	108	1
	1	104	105	185	194	1
Rating		**Q**				
		0	**0.3**	**0.5**	**0.7**	**1**
B	**0**	-0.11	-0.10	-0.25	0.00	-0.6
	0.2	0.00	0.00	0.00	0.00	-1.6
	0.4	0.00	-0.15	0.00	0.00	-9.6
	0.6	-0.20	0.00	-0.15	-0.65	-10.2
	0.8	0.00	0.00	-0.15	-3.05	-13.8
	1	0.00	-0.25	0.00	-7.20	-11

Table 4 Variation of Convergence and Rating with B and Q

Application to problem of higher complexity

Problems with larger number of nodes converge slower due to more information required to assess the complexity in the problem. There are more locations where rules can be applied and therefore for each additional location there are 4 possibilities included, one for each direction of connectivity. The approximate number of candidates in the problem with 8 nodes is 3^8 which is 6,561 and so the problem does not represent a high degree of complexity. In order to demonstrate the effectiveness of this process, we use an initial problem with 24 nodes and a target optimal graph that contains an arbitrary connectivity between the nodes. The number of possible configurations grows exponentially and the approximate number of variations in the case of 24 nodes is 3^{24} which is 282,429,536,481. The size of the problem is significantly larger and convergence on a problem of this size will show the effectiveness of this method in searching over a large solution space.

The search process runs for 200 iterations and to offset the stochastic nature of the process, 20 identical runs are executed and the results are averaged. The best value of B and Q are 0.4 and 0.5 as determined form the results of the previous experiment. The convergence graph is shown in Figure 17. Since the evaluation mechanism is simplified, the main consumers of time during the initial stages of the search process are the graph manipulation steps: recognize, choose and apply. One run of the search process takes with the seed graph containing 8 nodes takes less than 10 minutes to complete 200 iterations. The incremental time taken to complete successive iterations increases as more candidates are evaluated and more nuggets are added to the knowledge. Creating a candidate solution for evaluation requires selection of option at each stage of the graph transformation and every selection requires consulting this knowledge database. Time consumed for generating a candidate increases proportionally with the number of nuggets. However, with regard to the experiments conducted in this work, the search process was run 20 times for each experiment with a different value of B and Q taking less than 5 hours to complete.

With higher number of nodes, the process takes longer to complete. The reason is that with higher number of nodes, more rule-location pairs can be tested with each candidate that has been evaluated. The knowledge database thus increases at a faster rate (in this problem, it is approximately 3 times bigger as the number of options in a solution graph is proportional to the number of nodes where rules can be applied) and therefore the process takes longer to complete and in this case it takes around 20 minutes for a single run. However, it is very important to note that this time taken is purely because of the search process and not because of the evaluation. An evaluation mechanism for this problem as mentioned earlier would take in the order of 5 to 10 minutes for a single candidate. In this case, the process evaluates one candidate per iteration and would therefore consume around 10 to 15 hours for completion.

Even with such a large space of possible solutions, the search process has converged within a distance of 2 from the target solution within the first 100 iterations. The process also continues to improve by smaller amounts after the 100th iteration and at the end of 200 iterations has converged on a better solution.

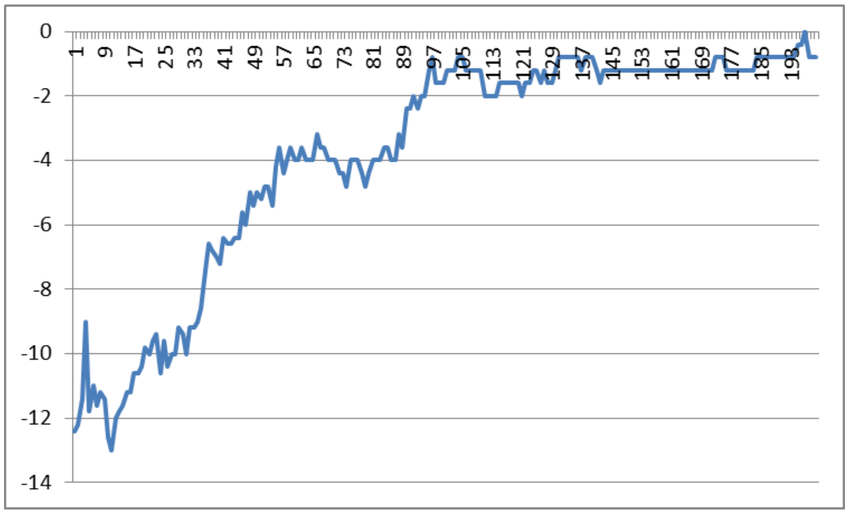
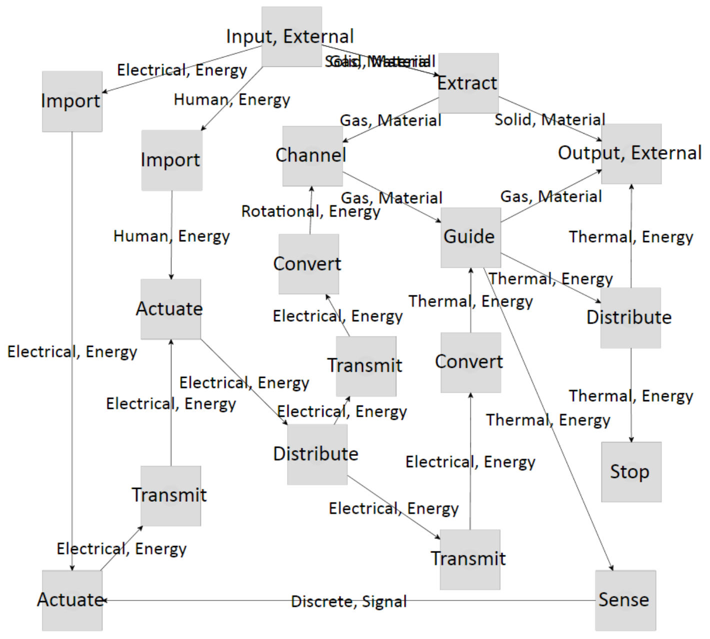


Figure 17 Progress of candidate rating vs. iteration number for a candidate of size 24 nodes

Product assemblies

The PV optimization problem falls under the category of problems with a small set of rules that apply to many locations in the host graphs. A second category of problems would contain many rules in a relatively fewer set of locations in the graph.

Electromechanical products such as fans, leaf blowers and staple guns are popular in the consumer market and many different companies have their unique designs that offer advantages in terms of packaging, efficiency, ergonomics and styling. Conceptual designs of such products often use graphs such as Function Structures (FS) that show the interaction between different functional modules of the product. The FS is realized by actual designs that replace these functional modules with components such as a motor, switch, indicator lights, battery etc. A Component Flow Graph (CFG) represents a solution to the function structure where there are no function nodes and instead contains components that can be used to build the product. FS and CFGs are built by disassembling products, studying the interaction between the components. An example of FS and CFG are show in Figure 18 which were obtained by disassembling a hairdryer. Twelve different products were identified based on functionality, complexity, engineering domain and nature of use which are listed in Table 5. Each of these products were disassembled and their function structure and CFG were derived. These consumer products range from 19 to 46 components and are easily disassembled without destruction. Their functionality always involves transfer of mass and energy such as air, dust, heat, mechanical force or transformation between these energy domains. For example, the function of a fan inside a hair dryer is to move air while adding energy to it in the form of increase in velocity while the heater in the fan does not move material but transforms electrical energy to heat energy. Interactions such as these are captured in function structures by using a generalized terminology that works across these products.



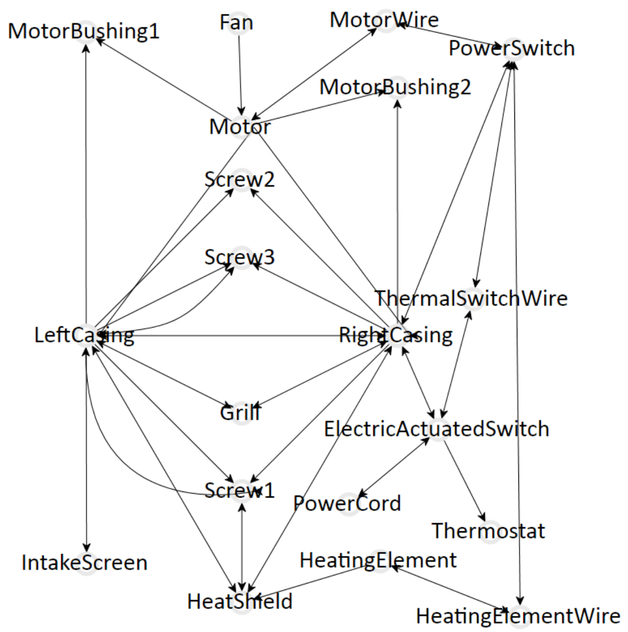


Figure 18 Screenshots of Function Structure (top) and Component flow Graph (bottom) of a common hair dryer

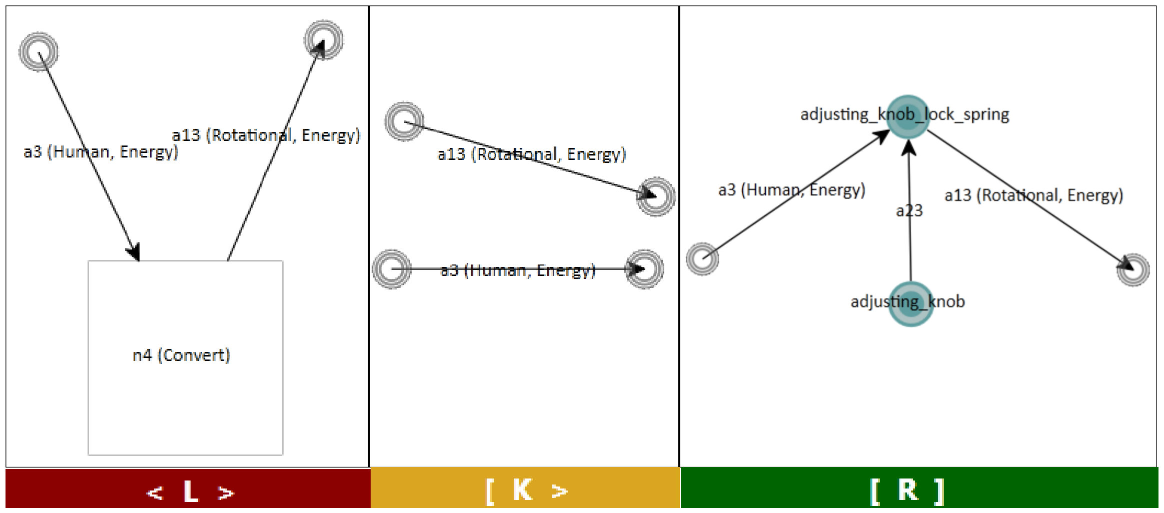


Figure 19 Rules that transform a single function node to multiple component nodes

ID	Product Name	# functions in FS	# components in CFG
A	Common Alternator	13	36
B	Hair Dryer	18	20
C	Hydraulic Jack	18	64
D	Lycoming T53 Turbine	14	20
E	Portable Air Compressor	25	64
F	Proctor Toaster	17	54
G	Squirt Gun	21	63
H	Stanley Staple Gun	26	60
I	Troy Bilt Leaf Blower	10	19
J	VW Bug Carburettor	20	58
Average		18.2	46
Min		10	19
Max		26	64

Table 5 Candidates from which rules were generated

The next step for a generative grammar process is to create grammar rules. The graph grammar representation allows for complicated component-function relation. In these products, there is not always a one to one mapping between the components and the function. A set of components might be responsible for achieving a single function such as the grammar shown in Figure 19. In this rule, a conversion between human energy to rotational energy is accomplished by a knob and a spring pair. Each grammar rule represents a way of realizing a function or a group of functions using a set of components. The interaction between the components and the function is also captured by representing them as graphs inside the rule. In the same example, the rule captures the connectivity between the knob and the spring and indicates how energy flows through the system. Rules thus capture a more complicated form of knowledge than a components database that stores a list of components and the function they satisfy.

It is better to create as many different rules as possible for a larger variety in the possible designs. Grammar rules are generated for this problem using an automated rule generator that attempts to extract rules from these products by comparing their Function Structures and CFG. The process works by correlating the information contained in the arcs from the two graphs matching those components and functions that work between a particular energy transformation or mass transfer. These components and functions are captured and stored as the L and R of a grammar rule. Rules were generated from each of the products above using this method and a total of 90 rules were generated and grouped together to form a rule set. By setting the seed graph as a function structure and by running a generative process using this rule set, a variety of solutions can be created and each solution represents a method showing how concepts and components from other products can be reused to better redesign a product at hand. By including rules from other products, the original CFG of the product is just one solution in a large search space.

Experiment

Similar to the previous problem, a preset solution is identified as the best solution and the aim of the search process is to converge as close as possible to this target. The best solution could be arbitrarily selected and for this experiment, it is selected as the CFG of the original product that was disassembled. The search process has no idea of the nature of the target solution and similar to the previous case works by using a distance metric that determines how close a given solution is to the target. To compute the distance metric, the graph is converted to a vector representation. For the case of this problem, an adjacency matrix, which is similar to a Design Structure Matrix, is created from the original graph. This is a $n \times n$ matrix where n is the number of nodes in the graph and each row and column represents a node of the graph. The matrix consists of 0s and 1s where a 1 represents connectivity between the nodes represented by the row and column of the element. This matrix is converted to a linear vector of size $n^2 \times 1$ by appending all the rows. The distance between the two graphs is thus found by calculating the Euclidean distance between the two vectors.

To find the best values for the search parameters B and Q, experiments are run with B values ranging from 0 to 1 with step size of 0.2 and Q taking values 0, 0.3, 0.5, 0.7, 1.0. And similarly, each run of the search process is run 20 times to compensate for the stochastic nature of the process. The seeds *hydraulic jack* and the *portable air compressor* were chosen as they have a large number of nodes in their function structure and are therefore the more complex cases. The rule set used here contains the 90 rules created from disassembling all the products.

The results of the experiment are captured in the form of convergence graph for each different values of B. The convergence graphs are shown in

Figure 20 and Figure 21 where each graph shows the convergence with different values of B and Q. Since the process converges early enough for the case with the hydraulic jack, only 100 iterations are shown in the figure. Similar trends are observed as in the case of the PV array optimization problem. For lower values of B where the search process exploits more, the process has a lot of variation as shown in

Figure 20a and Figure 21a. As B and Q value increase, the search tends more towards exploit. This can be seen in the case when B= 1.0 where for all graphs with Q values > 0.5 the search process converges on the first value it chooses. In the mid-range, the process attains a balance between exploration and exploitation and converges on a good solution while the variation stays within a reasonable limit.

The maximum rating is the rating of the final solution that the process converged to. The point of convergence is calculated as the iteration at which the process attains stability within 1% of the max rating. For the case of the hydraulic jack*,* Table 6 tabulates the point of convergence and max rating values for each case of B and Q values as derived from the graph. Best values of B and Q are chosen as those with minimum iterations to convergence and the highest maximum rating. For values of B and Q for which rating is the highest, the number of iterations to convergence is large. For example the case with B and Q as 0.0, it takes 57 iterations to converge when compared to the case with B and Q as 1.0 where it converges in the first step. In cases where the convergence is fastest, for example, with B and Q as 1.0, the maximum rating of the candidates is much lower than the ideal obtained in other cases (-2714 compared to -2042). To strike a balance between the two, the best values obtained are when B and Q values are 0.4 and 0.5 respectively. This trend is also observed in the case of the PV array optimization problem.

The search process has converged within 30 iterations for the case for hydraulic jack and around 120 iterations with the seed as the portable air compressor. The number of evaluations done is far minimal when compared to the number of possible solutions. Previous work show that the possibility for number of candidates is in the order of thousands and the reduction in the number of evaluations shows a significant reduction in the time complexity of the search method and shows the effectiveness of this method.

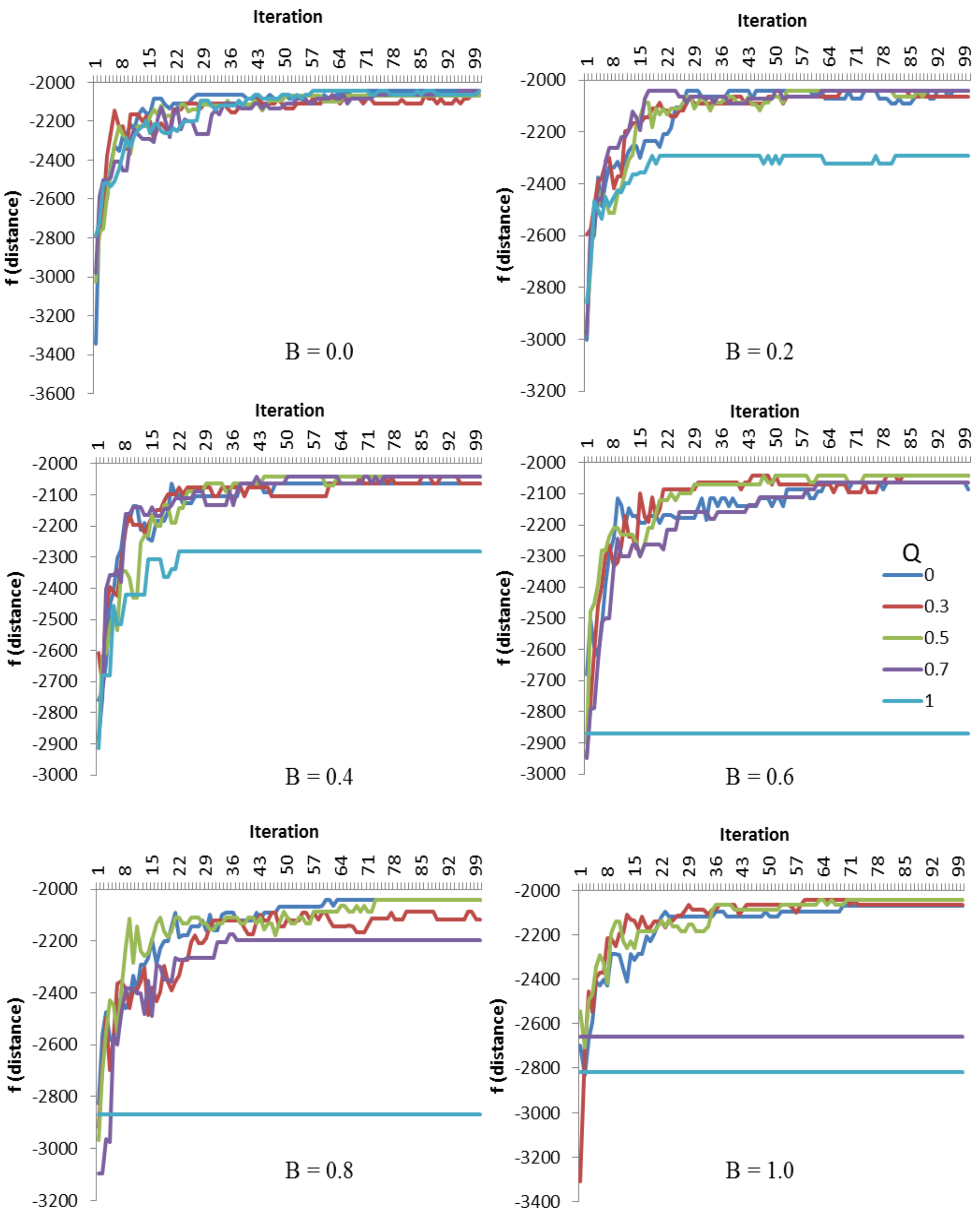


Figure 20 Convergence of search process when used with seed graph as hydraulic jack

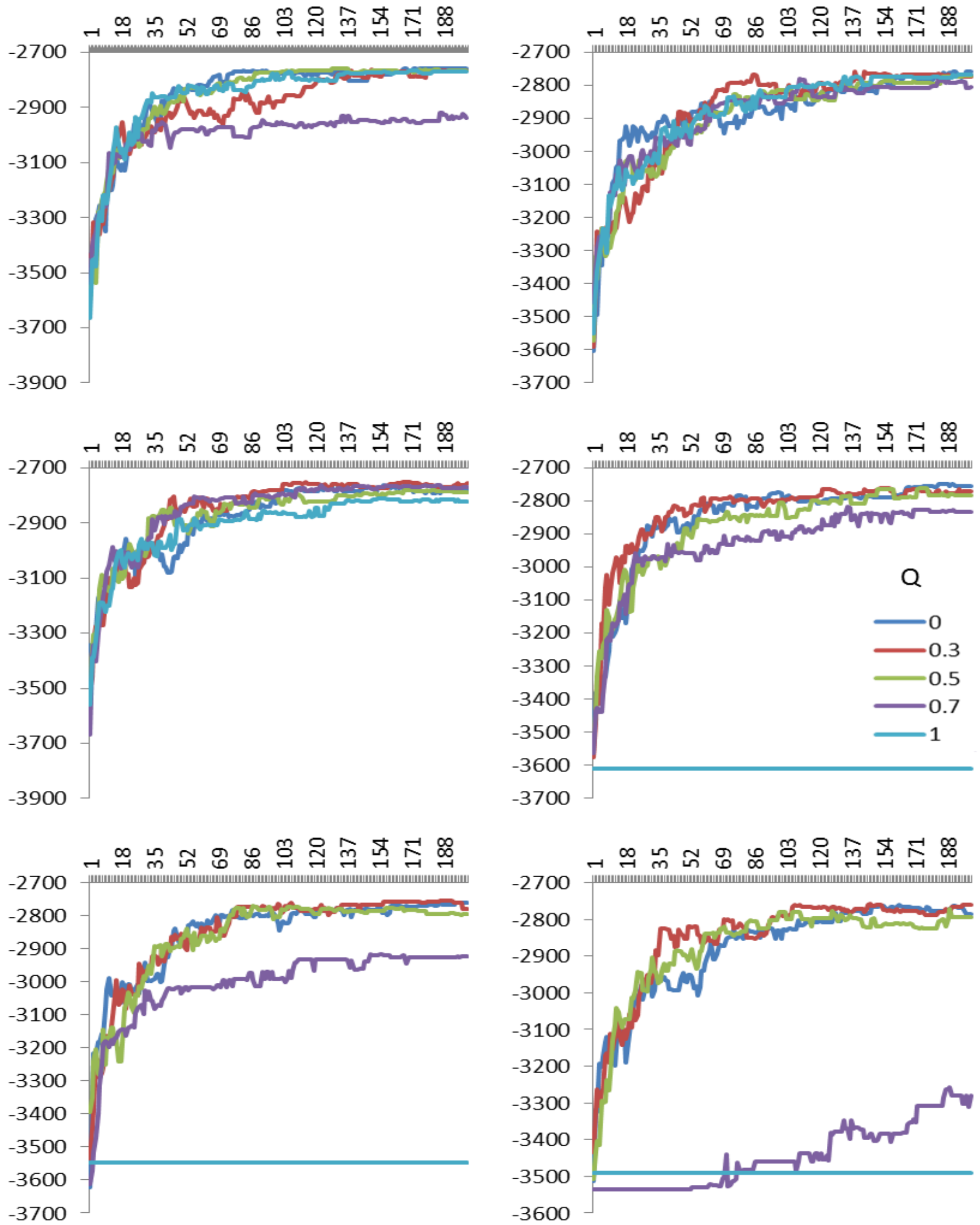


Figure 21 Convergence of search process using a portable compressor seed of size 25 nodes

Max Rating		Q				
		0.0	0.3	0.5	0.7	1.0
B	0	-2042	-2064	-2069.8	-2042	-2064.2
	0.2	-2042	-2064	-2042	-2042	-2293
	0.4	-2064.2	-20642	-2042	-2042	-2281.8
	0.6	-2064.2	-2042	-2042	-2064.2	-2870.6
	0.8	-2042	-2116.2	-2042	-2198.8	-2869.2
	1	-2069.8	-2064.2	-2042	-2657.8	-2817

Convergence		Q				
		0	0.3	0.5	0.7	1.0
B	0	62	98	45	72	59
	0.2	30	27	48	43	21
	0.4	42	60	29	37	21
	0.6	62	76	28	57	1
	0.8	58	70	72	34	1
	1	57	57	52	1	1

Table 6 Max rating and average convergence point of search process for B and Q values from 0 to 1 and 0.1 to 0.9 respectively with seed of hydraulic jack FS.

Conclusions and future work

The work presented here shows how a topology optimization problem can be solved using a generative grammar approach. The results show that achieving a near optimal solution for a problem space containing unmanageable number of solutions (more than a million) is possible within a few hundred iterations. This is possible only in a framework that uses the generative grammar rules as the fundamental building blocks. An optimal value for the search parameters B and Q were determined from experiments. The method has been shown to work on two kinds of problems, one with a few rules that can be applied at several location in the graph and one with several rules of which only a few are applicable in few select locations in the graph. Many problems in graph grammar involve a similar situation where the question is to select the location of each rule to be applied.

The methodology shows a lot of potential for future improvement. The objective function used here is a monotonically decreasing distance function to a known optimal making it a unimodal problem. The methodology will likely require modification to be effective on a multimodal problem. For example, in the PV array problem, two configurations could exist with a 4 x 4 grid of nodes, one having all arcs pointing south and the other having all arcs pointing east. Although these two configurations are equivalent, they will have a different set of labels for each node. If both these solutions are rated high and the distance metric is such that there are two local minima in the search space, the process will then give equal fitness for arcs pointing south and east. The final solution will therefore contain a mix of both arcs and not be an optimal configuration. To solve a problem of this kind, we will need to use a context measure that assigns a fitness for a rule based on other rules and options that have been applied on the graph. This can open up the possibility for the search process to consider more than a single solution.

However, detailed study needs to be done before a context measure can be made up by examining the effect it will have on both unimodal and multimodal problems.

Another optimization that can be made is to modify of B and Q values of the search process as a function of time. In processes such as simulated annealing, the exploration is set very high at the beginning in order to explore the search space more thoroughly. The process gradually moves towards exploitation towards the end when a reasonable amount of scanning has been done. In this search process, B and Q values determine the exploration and exploitation of the search process. It can been seen in Figure 16f that even after the search process has converged on a value after around 100 iterations, small variations are still seen. This is because the process still tries to explore to see if there were any options left out from being added to rule knowledge or reexamine options that do not have enough popularity. Although this results in a more comprehensive evaluation of the design space, it also leads to a slower convergence of the search process. Thus there is a question of when the process should be made to completely exploit and this would depend on the nature of the problem.

There are some other issues that could be investigated to study the effectiveness of the process. First, the same problems need to be solved using other techniques such as genetic algorithm to benchmark the performance and provide insight for further improvement.

In this thesis, the RBITS method was extended.to use an accurate simulator. The above work demonstrates that this can be used effectively to solve real engineering problems. Several tweaks can be easily performed to control the nature of the search process such as explore vs. exploit, storing and retrieving rule knowledge etc. to suit the nature of the problem. Graph grammar is an efficient representation technique is being increasingly used in several domains and a search process that efficiently uses this representation can prove to be very useful in solving these problems.

Bibliography

[1] J. Nelson, *The physics of solar cells*. London: Imperial College Press, 2003.

[2] W. Beitz, G. Pahl, and K. Wallace, *Engineering design: a systematic approach*. Springer, 2003.

[3] G. Rozenberg, *Handbook of graph grammars and computing by graph transformation: Foundations*, vol. 1. World Scientific, 2003.

[4] P. Radhakrishnan and M. I. Campbell, "A Graph Grammar Based Scheme for Generating and Evaluating Planar Mechanisms," in *Design Computing and Cognition '10*, J. S. Gero, Ed. Dordrecht: Springer Netherlands, 2011, pp. 663-679.

[5] A. Swantner and M. I. Campbell, "Automated Synthesis and Optimization of Gear Train Topologies," 2009.

[6] K. Shea and A. C. Starling, "Virtual Synthesizers for Mechanical Gear Systems," *Proceedings of ICED"05 International Conference on Engineering Design, Melbourne, Australia.*

[7] P. Sridharan and M. I. Campbell, "A study on the grammatical construction of function structures," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 19, no. 3, pp. 139-160, 2005.

[8] M. Agarwal and J. Cagan, "A blend of different tastes: the language of coffeemakers," *Environment and Planning B*, vol. 25, pp. 205-226, 1998.

[9] K. Shea, J. Cagan, and S. J. Fenves, "A shape annealing approach to optimal truss design with dynamic grouping of members," *Journal of Mechanical Design*, vol. 119, p. 388, 1997.

[10] A. C. Starling and K. Shea, "A grammatical approach to computational generation of mechanical clock designs," 2003.

[11] "GraphSynth | GraphSynth2: software for generative grammars and creative search." [Online]. Available: http://graphsynth.com/. [Accessed: 15-Nov-2011].

[12] C. Koenigseder, K. Shea, and M. I. Campbell, "Comparing a graph grammar based to a classical optimization approach for computationally designing PV arrays," *Proceedings of the 22nd CIRP Design Conference, Bangalore India, 28-30 March*, 2012.

[13] M. S. Hundal, "A systematic method for developing function structures, solutions and concept variants," *Mechanism and Machine theory*, vol. 25, no. 3, pp. 243-256, 1990.

[14] A. C. Ward and W. P. Seering, "The Performance of a Mechanical Design 'Compiler'," *Journal of Mechanical Design*, vol. 115, p. 341, 1993.

[15] M. Kumar and M. I. Campbell, "Organizing a Design Space of Disparate Component Topologies," *Design Computing and Cognition'10*, pp. 465-485, 2011.

[16] L. Qian and J. S. Gero, "Function-behavior-structure paths and their role in analogy-based design," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM*, vol. 10, no. 4, pp. 289-312, 1996.

[17] J. Pearl and R. E. Korf, "Search techniques," *Annual Review of Computer Science*, vol. 2, no. 1, pp. 451-467, 1987.

[18] L. C. Schmidt and J. Cagan, “Recursive annealing: a computational model for machine design,” *Research in Engineering Design*, vol. 7, no. 2, pp. 102-125, 1995.

[19] J. Patel and M. I. Campbell, “An Optimization Approach/Technique for Solving Graph Based Design Problems,” 2008.

[20] M. I. Campbell, R. Rai, and T. Kurtoglu, “A stochastic graph grammar algorithm for interactive search,” in *Proceedings of ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE*, 2009.

[21] T. Esram and P. L. Chapman, “Comparison of photovoltaic array maximum power point tracking techniques,” *Energy conversion, IEEE transactions on*, vol. 22, no. 2, pp. 439-449, 2007.

[22] L. Gao, R. A. Dougal, S. Liu, and A. P. Iotova, “Parallel-connected solar PV system to address partial and rapidly fluctuating shadow conditions,” *Industrial Electronics, IEEE Transactions on*, vol. 56, no. 5, pp. 1548-1556, 2009.