AN APPROACH TO AUTOMATE AND OPTIMIZE CONCEPT GENERATION OF SHEET METAL PARTS BY TOPOLOGICAL AND PARAMETRIC DECOUPLING

Jay Patel
Department of Mechanical Engineering
The University of Texas at Austin
Austin, TX 78712-0292
pateljk@mail.utexas.edu

Matthew I. Campbell
Department of Mechanical Engineering
The University of Texas at Austin
Austin, TX 78712-0292
mc1@mail.utexas.edu

ABSTRACT

This paper describes an approach to automate the design for sheet metal parts that are not only novel and manufacturable but also satisfies multiple objective functions such as material cost and manufacturability. Unlike commercial software tools such as Pro/SHEETMETAL which aids the user in finalizing and determining the sequence of manufacturing operations for a specified component, our approach starts with spatial constraints in order to create the component geometries and helps the designer design. While there is an enormous set of parts that can feasibly be generated with sheet metal, it is difficult to define this space systematically. To solve this problem, we currently have 108 design rules that have been developed for five basic sheet metal operations: slitting, notching, shearing, bending and punching. A recipe of the operations for a final optimal design is then presented to the manufacturing engineers thus saving them time and cost. The technique revealed in this paper represents candidate solutions as a graph of nodes and arcs where each node is a rectangular patch of sheet metal, and modifications are progressively made to the sheet to maintain the parts manufacturability. They are presented in the form of Standard Tessellation Language files (.stl) that can be transferred into available modeling software for further analysis.

This paper also discusses a new topological optimization technique to solve graph based engineering design problems by decoupling parameters and topology changes. This paper presents Topological and Parametric Tune and Prune (TP^2) as a topology optimization method that has been developed specifically for domains representable by a graph grammar schema. The method is stochastic and incorporates distinct phases for modifying the topologies and modifying parameters stored within topologies. Thus far, with abovementioned sheetmetal problem, (TP^2) had proven better than genetic algorithm in terms of the quality of solutions and time taken to acquire them.
Keywords: sheet metal, graph, rule-based systems, topology, graph transformation, grammar, optimization, parametric tuning, design automation

1 INTRODUCTION

Sheet metal is used in various applications from body parts of vehicles such as airplanes, cars, helicopters, and motorcycles to small and customized artifacts such as furniture, cabinets, machine bases and covers. Due to the extensive use of the sheet metal parts, any enhancements made in the field of their design and manufacturing could have a large impact on the modern engineering artifacts. In comparison to forging or machining components, the sheet metal can produce lightweight and inexpensive design solutions. The main shortcomings of sheet metal design are that resulting components have a limited rigidity, and the feasibility of the parts is constrained by the inherent two-dimensionality of the initial sheet. Various structural configurations can be incorporated in the designs to increase the stiffness which would overcome the above limitation. Good design is also based on how the design engineers manage the trade-offs between the manufacturing process and the design specifications. Smart solutions to the design of sheet metal components can reduce both manufacturing time and energy, and result in high quality components. The concurrent efforts between the manufacturing engineers and the design engineers are complicated for even the simplest sheet metal components, resulting in an iterative and time-consuming design process. Usually designing of the sheet metal components is left to experienced designers, who must understand both the functional requirements that a specific part must accomplish and the constraints involved in fabricating the components. The conventional process of sheet metal design is a very monotonous and time consuming design process. There is modeling software such as Pro/SHEETMETAL [1] and Hyperform [2] that can assist the design engineers to form a three dimensional CAD model of a part and analysis it using finite element analysis respectively, but this would not assist designer in the concept generation process which currently is accomplished by passing concepts back and forth between design and manufacturing engineers. This process is not only tedious but it also lowers productivity resulting from multiple redesign of the product.

Designing of the sheet metal components is perhaps one of the most challenging issues for the design and manufacturing engineers. To ease the design process, a method presented in this paper has been developed to encapsulate the spatial constraints of sheet metal parts, at the same time avoiding certain locations that would impinge upon other parts. This paper presents a novel way to automate the synthesis of the sheet metal parts while taking into account the overall manufacturability of parts during the design process. A design routine developed here consists of four elements namely representation, generation, guidance, and evaluation. The approach formulates the problem using designer's input and the library of grammar rules (representation) which synthesize a wide

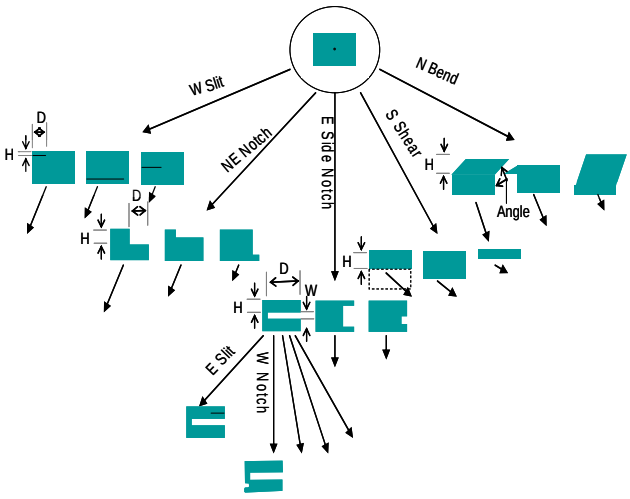

Figure 1: Tree formation leading to different designs depending on the rule and parameter choice.

range of feasible solutions (generation). These are evaluated based on the user-defined specifications (evaluation), and the search process navigates towards progressively better solutions via a stochastic search process (guidance). The resulting designs are presented in form of candidate solutions which are represented as a graph of nodes and arcs, and modifications are progressively made to the sheet in order to maintain a part's manufacturability. A search process is established that can produce the designs, which are optimal in both performance and manufacturing specifications. Hence, this approach offers a different perspective for the designers by providing a number of solutions, and helps them create novel designs. These designs can be then modified or taken as they are, depending on the design requirements and customer needs. A "recipe" can be provided at the end of the search process that captures the sequence of the manufacturing operations that are used to produce the design as well as the design variables chosen to perform the operations. Knowing this information would help the manufacturing engineer maintain a lean production of indefinite components.

For the most part, computational tools in engineering design that are centered on aiding or automating the synthesis of engineering artifacts are built on either numerical optimization [3] or tree-search methods [4]. Numerical optimization models the design problem as a fixed set of decision variables and with one or more objective functions. However, synthesis in engineering design often involves making many difficult decisions that are beyond the representational power of optimization. Early in the design process one cannot often reduce the design problem to a set of fixed decision variables. This fixed set of decision variables in optimization is typically viewed as a vector. But vectors and matrices have shortfalls since only a limited amount of connectivity information can be stored and their size is fixed in nearly all relevant mathematical techniques, thus limiting one to a uniform amount of complexity

A better modeling concept for synthesis would be not to look at design as a fixed set of design variables but as a mathematical graph. A mathematical graph is essentially a collection of two basic elements: nodes and arcs. The physical sciences and engineering have used graphs to great effect. In fact, upon examining several engineering examples (e.g. electric circuits, chemical process diagram, control block diagrams, function structures, etc.) it becomes clear that representing such examples in the form of graphs results in a richer more insightful representation. Graphs afford a level of abstraction which is unrivalled by any fixed array or vector format, and their visual representations are meaningful and efficient to their human observers.

The construction of these complex graphs is subject to certain rules or guidelines. In many cases these are not explicitly stated and taught through examples only. But in theory, one can view the set of; say chemical processing diagrams, as artifacts of an underlying set of grammar rules that govern how process nodes are connected and the valid set of labels and connection types that exist. Defining a language by a set of formal grammar rules not only eases the learning process for people but also affords computational creating of valid artifacts.

In this paper, a search method is developed for engineering systems defined by a graph grammar. The method has been successfully applied to some small example problems and to the creation of sheet metal component. The method is known as Topological and Parametric Tune and Prune, or $(TP)^2$. As the name implies the method can address problems that have topological changes – changes in the number and connection of discrete elements – as well as parameters stored within the discrete elements. As a simple example, the creation of sheet metal design configurations from discrete operations in shown in Figure 1. Given a small set of rules, such as 'westerly slit', 'easterly notch' and 'northern bend' an infinite set of configurations or topologies can be created. Within each topology a number of parameters may be adjusted for example the height and depth value of the slitting operation. Within this space of solutions each state in the tree is a potential solution. It is unclear at the onset how far down the tree one needs to go to solve a particular design problem successfully. This type of design problem in which varying complexity exists within the design space and each configuration carries its own parameters has not been solved computationally. However, it is clear that such situations exist often in the engineering design process. The development of $(TP)^2$ is motivated and created by these types of problems.

The division of tasks within the search process follows the flowchart shown in Figure 2. This flowchart has been used through a number of computational design synthesis project [5] as a way to break down the synthesis problem into four main challenges of representation, generation, evaluation and guidance. Essentially, the new elements of $(TP)^2$ only affect the guidance and generation tasks. Representation and the recognize-choose-apply cycle are described in last year's DTM paper [6]. This is accomplished within a computational framework, known as GraphSynth [7] which provides the fundamental objects for defining specific problem domains by grammar rules. The evaluation task is always a domain specific and is accomplished by either closed form objective functions or by invoking external computational simulations.

2 BACKGROUND

Traditionally, related research has focused more on the manufacturability or assembly of the sheet metal parts rather than design synthesis. Process planning being a time consuming and tedious operation, is optimized using the knowledge of the features and their corresponding heuristics [8]. To make this process planning more efficient, Wang and Sturges [9] came up with a representation that eliminates the ambiguities of wireframe and flat panel models. This research project was focused on only bending operations since they play a vital role in manufacturability of parts. A method for automated process planning of sheet metal bending operations using the A* algorithm was presented by Gupta et al [10]. Manufacturability analysis in the above-mentioned research is done by decomposing the final design into simpler parts to find its interference with the bending tool.

Our research, unlike all the previous research, starts with just the user-defined specifications and comes up with designs while concurrently taking into consideration the manufacturability of the parts. We use shape grammars to formulate the design space based on manufacturing operations. Shape grammars have been previously used for formulating and representing design rules and thus the space. Padhye and Campbell [11, 12] incorporate an evaluation method for the topological synthesis of the sheet metal components. This evaluation method uses 17 shape grammar rules[11, 13] to represent the sheet metal components, and provides an approach towards the automated design of the sheet metal parts through a user-computer interface. Similar to many engineering problems, the sheet metal design problem can be represented in form of a graph consisting of nodes and arcs. Graph grammars are a technique that transforms an initial seed or specification into a final design. The basic concept of combining the manufacturing constraints and graph grammar has been seen in the lathe grammar of Brown and Cagan [14]. Though the concept of graph grammars has existed for the last couple of decades in artificial intelligence literature, researchers in the field of design automation have now begun to realize the advantage of using them to exploit both the knowledge and heuristics of a particular problem domain. Recent contributions indicate an increasing pace in graph grammar developments such as designing innovative epicyclic gear trains by Li and Schmidt (Li Schmidt}, and Emdanat solving form-making problems [15] using shape algebras. This paper presents a graph grammar that formulates the manufacturing based design rules to define the feasible space.

Graph transformation systems or graph grammars formally exist in graph theory research as a way to rigorously define mathematical operations such as addition and intersection of graphs [16]. The foundations of graph grammar research lies within theoretical computer science and mathematics, but it is only recently that researchers in engineering design have assimilated these concepts as a way to formalize the creation of complex engineering systems [17].

These grammar rules need to be constructed prior to the start of the design cycle as an encapsulation of the heuristics for designing within that domain. The graph transformation process in the engineering design context can be viewed as a distinct three step process: *recognize*, *choose* and *apply*. At any given stage in the synthesis of a design, it is desirable to know all possible design perturbations or options. These options are comprised of both recognized grammar rules and their locations. Once the recognition process is done, it presents a list of valid options. Next, a particular rule is selected within the *choose* step by some decision-making agent such as a human designer or a computer search algorithm. Such an agent should be able to predict the benefits of making a particular rule choice over others – this is essentially an area where decision-making research can play a vital role. The final step is the *apply* step. The chosen option is now applied to the graph in question and a new resultant graph is created. This apply step may seem rather obvious and rote but since it may introduce or remove nodes and arcs, or alter the properties of a node or arc within the current graph, a comprehensive apply method is difficult to fathom. Fortunately, a number of advances from theoretical computer science have established clear transformation methods [18, 19]; these have provided the basis for GraphSynth's apply functions.

A key feature of using a graph-based *modeling concept* in design is that the space of possible designs can be viewed as a large tree starting with a seed or initial state. As shown in Figure 1, the various branches of the tree correspond to grammar rule applications. The "seed" graph or starting point for the method is a simple signal source with leads coming out of it. After the recognize process rules 1, 2 and 3 are seen to be applicable. Imagine a scenario where rules 2, 1, and 3 are chosen in sequential fashion to arrive at the final design. The figure however indicates that at any given point in the state tree there are often multiple options to choose from and that the current position is dictated by the decisions we have made thus far.

Traditionally, tree searching algorithms such as A* [4] search or Uniform Cost [20] search have been used to search such trees for optimal solutions. However, in most engineering design applications the space is large, multi-modal and non-monotonic. Classical search techniques such as branch-and-bound [21] cannot be utilized as there is no clear upper bound that can be calculated or predicted. Various approaches have been developed to optimize design based on existing optimization methods. For example, a computational design method to generate structural truss topologies has been developed by Shea [22] that combines grammars with simulated annealing. A recursive search method for creating simple mechanical block diagrams is developed by Schmidt and Cagan [23] that builds structures from a graph grammar. Hornby has demonstrated a wide range of solutions for table designs using a Lindemayer generative representation [24]. Additionally, the agent-based approach, known as A-Design [25] builds electromechanical configurations using an agent-based strategy. All of these methods were created ad-hoc for a particular problem domain, and while they were successful, their applicability to other problems is not clear. The method discussed here is the first to attempt to generalize these problem types and arrive at a general search method.

3 DESIGN APPROACH

A rectangular piece of sheet metal is cut from a larger roll after determining the amount of material that would be needed to create a particular part. The initial sheet metal blank (starting seed for the approach) as revealed in Figure 3 is represented by a rectangular patch with attributes like width, height and uniform thickness. Unlike most design automation research for sheet metal, where a

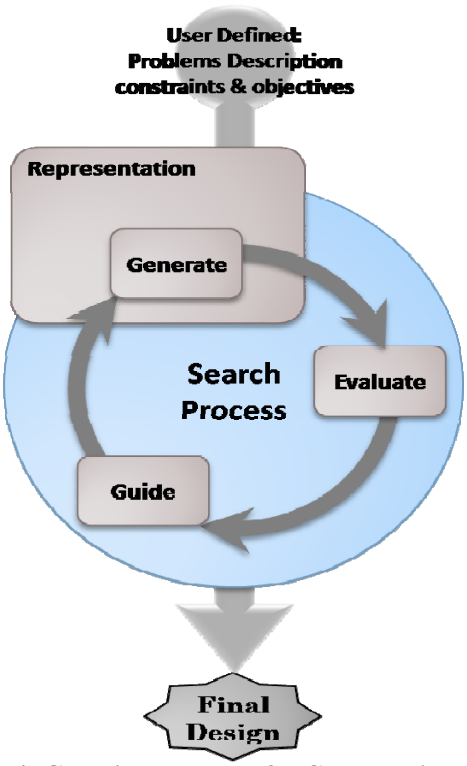

Figure 4: Generic Flowchart for Computational Synthesis has four divisions: a representation of the design space, a method for generating new solutions, a method for evaluating solutions, and a method for guiding the search process.

final shape of a part is known, the transformation of an initial blank to a final creative design is done using the GraphSynth platform. This approach of searching the design space for novel design is intended to create a set of unique solution that are then

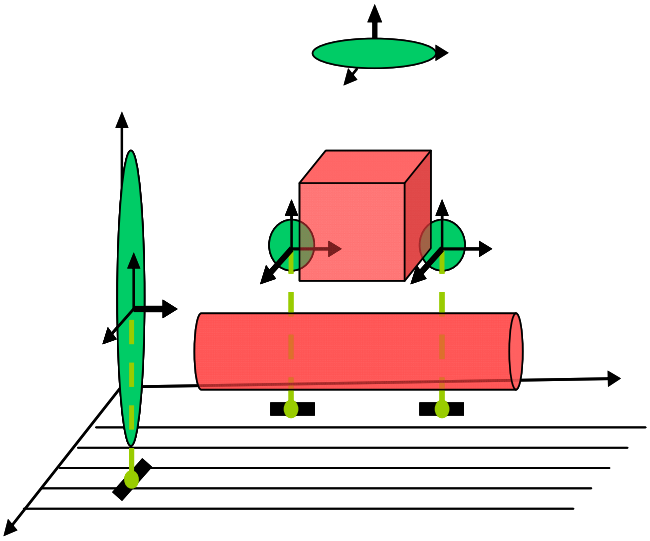

Figure 2 : The design specifications for a sheet metal component is embodied as spatial constraints, where green circles represent locations through which a sheet should pass, and red solids represent objects to be avoided.

optimized for cost and time. GraphSynth is the tool developed in the Automated Design Lab within the Department of Mechanical Engineering at the University of Texas at Austin for creating graph grammars. Within GraphSynth, one can design, implement, test and automatically invoke grammar rules that govern the design process.

Figure 2 presents a generic flowchart that is used for general design synthesis methods such as search strategies or optimization routines. The process uses the designer's input to define the problem which includes defining the objective functions and constraints discussed in more detail in Section 3.1. It is done by the users of the tool who they apply their understanding about the

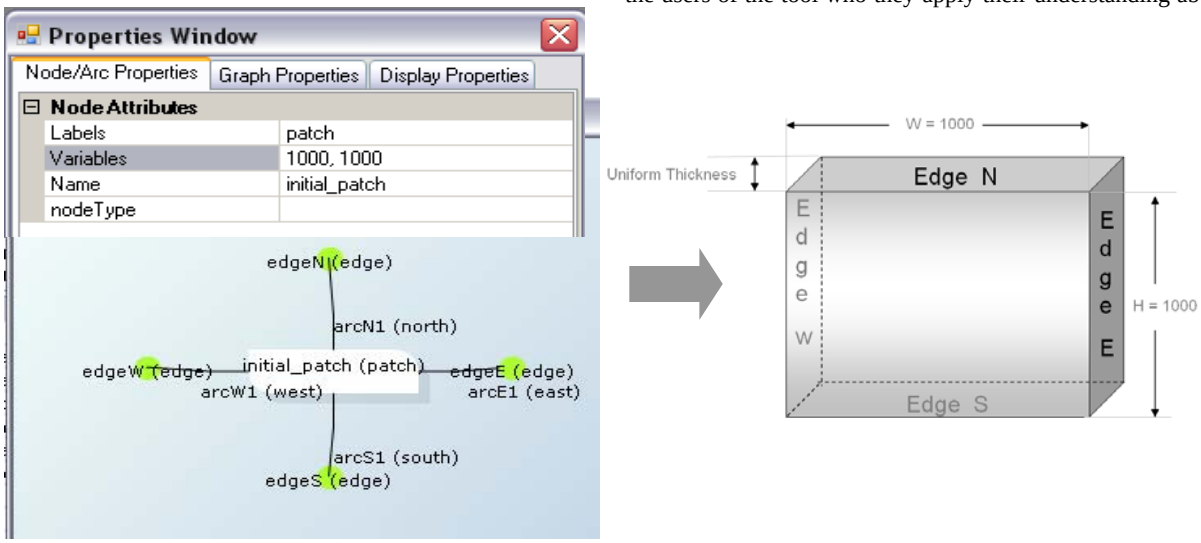

Figure 3: Initial seed representation as a sheet metal blank to adhere with production.

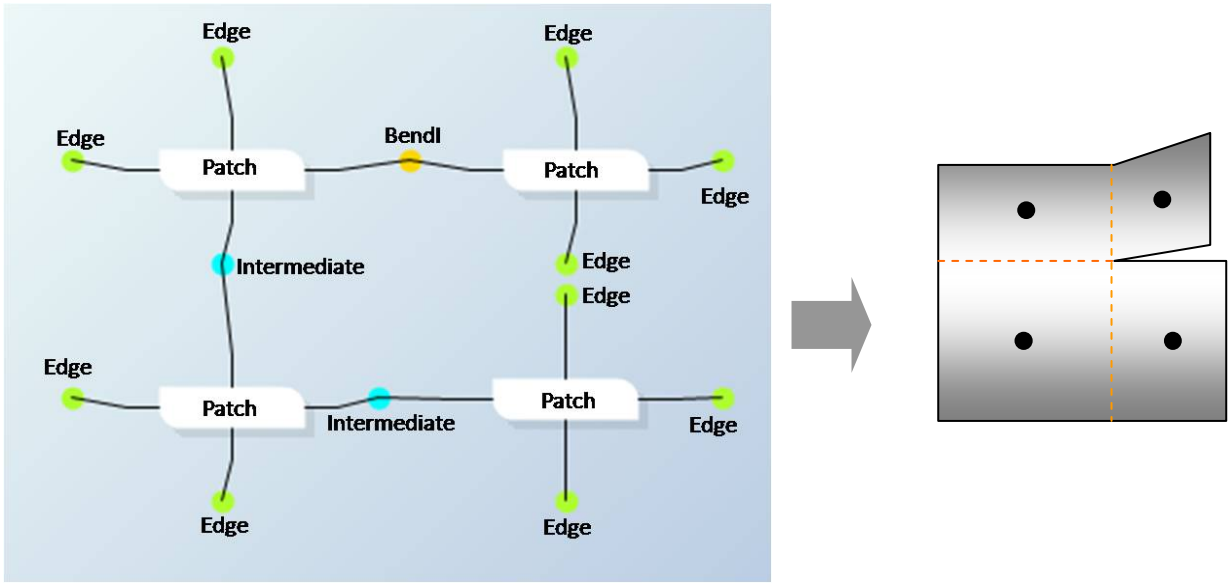

Figure 5 : Graph showing different kinds of nodes and arcs along with a sheet metal part resulting from it.

design problem to position the constraints in such a way that better solutions are resulted. Within the execution of the general design synthesis method, there are typically four common challenges which are depicted as representation, generations, evaluation, and guidance as seen in Figure 2. Representation (Section 3.2) is the formulation of all design variables which leads to all the possible design solutions. Using this representation, candidate solutions are created through the generation (Section 3.3) task. These candidate solutions depend on the choices made in terms of the design rules and the parameters.

As shown in Figure 1, if the first chosen rule is to perform westerly slitting operation, there are infinite possible designs that can be generated based on the values for depth and height of the slit. Hence, a wide range of the feasible solutions can be created transitioning through the levels of the design tree. Each one of the generated candidates is then analyzed in the evaluation step (Section 3.4) to determine how well it satisfies the objectives and the constraints of the design problem. A fitness value is then assigned to each candidate based on its evaluation. This value of the evaluation is passed to the guidance strategy (Section 4) so that better designs can be generated using the resulted direction.

Problem Description

The sheet metal parts due to their two dimensional nature are usually utilized to support light objects and/or provide enclosure for the objects. In this project, the spatial constraints are represented as green circles and red solids in three dimensional space as shown in Figure 4. Green circles represent the location where the designer wants the sheet metal to be present, which can result in providing support to the objects that are enclosed in the design and also material required for the fixtures/welds. These circles are used to emphasize on two-dimensional structural nature of sheet metal blank due to uniform thickness. A circle also simplifies the calculation to determine if the newly created design meets the user requirements, details of which are discussed in [12]. One of several ways of evaluating a spatial design to meet the requirements is to find out if the design covers area where the user intends to have material present and to avoid situations where it (design) is interfering with other existing parts. An example of the above mentioned evaluation is a car's hood design, a good design would have material present where it hinges to the car's frame as well as covers the engine, while avoiding interfering with engine as well as other components that's covered. Our approach would represent the spatial constraints like hinging and covering with circles and the existing parts including engine with solids. Red solids in the figure 4 exemplify other parts, which the designed sheet metal part must not interfere with. Hence, a good design would need to avoid the solids and pass through circles for making it feasible. Given the complex design process of the sheet metal components, evaluating candidates is a mixture of several additional objectives. These objectives include minimizing the material used to make the design and the fabrication energy, and maximizing the manufacturability.

3.1 Representation of a feasible space

Our approach formulates the design problem as grammar rules that define the design space. A design state is comprised of a graph of nodes and arcs. There are four basic nodes as shown in Figure 5 that are used in this approach, labeled as patch, edge, intermediate and bendI (an intermediate that is also a bend location). A Patch (white color node in Figure 5) can be viewed as a rectangular section of the sheet metal having certain properties such as length, width and uniform thickness. Every rectilinear patch must be bordered by four nodes one in each of the four compass directions. These nodes can be an Edge (green color), indicating that the patch has no other patch connected to it on that side, or an Intermediate (light blue color), showing the connection between a patch and another patch without bend applied on them, or a BendI (golden color), representing patch connectivity to another patch with a bend between them.

Currently, we have 108 design rules that govern the candidate solution and 36 auto rules that propagate the information of how the design affects the rest of the graph. These grammar rules are stored as XML files, loaded into the program, and then instantiated as defined rules with left-hand side and right-hand

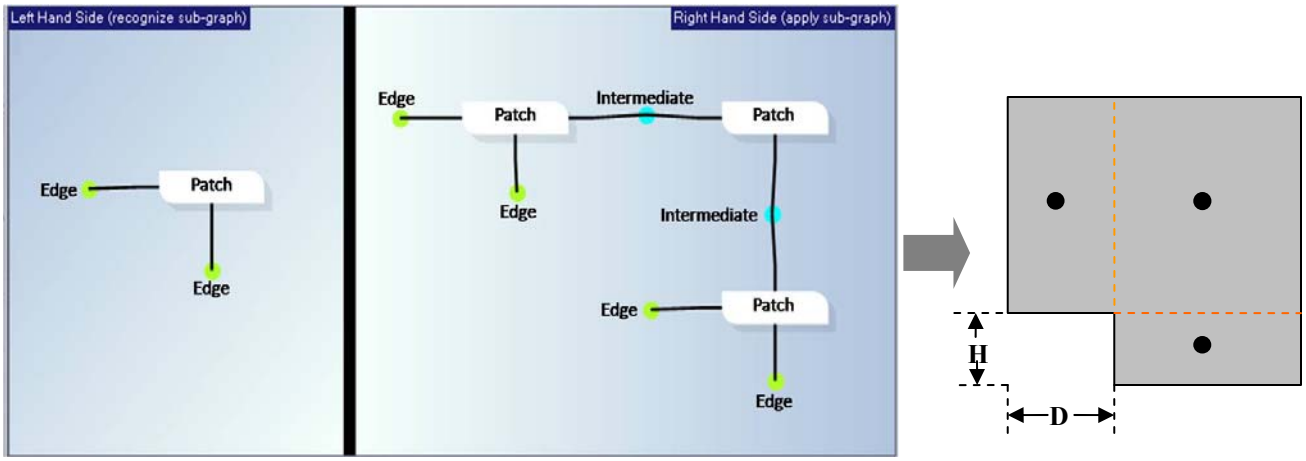

Figure 7: Corner notching rule showing left-hand side and right-hand side of grammar rules which requires 2 variables namely height and depth of the notch.

side graphs. The design rules are developed for five basic sheet metal operations: slitting, notching, shearing, stamping and bending. A recipe of the operations can thus be outputted to indicate which operations were executed, where they were implemented and what parameters were used. This recipe helps the manufacturing engineer to prepare and sequence these operations.

A grammar rule is essentially constructed of two elements: application conditions, and application instructions. These two elements are each represented as a graph: the conditional recognition or left-hand side graph, and the application or right-hand side graph. Grammar rules can be constrained, so that independent of how the rules are applied; one can develop a large set of designs, which are guaranteed to be within a feasible design space. Figure 6 shows a corner notch rule with 2 degree of freedom (variables), namely height at which the notch is formed and the depth of the notch, required performing the rule. The left hand side shows the condition which is searched for in the candidate graph, that is a patch connected to 2 edges on east and south direction. The application of this rule would replace the patch with three patches as shown in right hand side of the rule. On the right of the figure is the sheet metal part resulting from application of this rule on the blank patch. Every rule has different parametric requirements that lead to different designs, permutation of the degree of freedom and orientation for each operation leads to the current assortment of the grammar rules. All the manufacturing operations mentioned above require information such as the orientation or location of where they are applied and design parameters that would govern the resulting topology. The number of parameters that are required for these operations

	0 Parameter				1 Parameter				2 Parameters				3 Parameters				4 Parameters			
	E	N	W	S	E	N	W	S	E	N	W	S	E	N	W	S				
Slit	17	18	19	20	5	6	7	8	1	2	3	4								
	21	22	23	24	9	10	11	12												
					13	14	15	16												
Corner Notch	33	34	35	36	29	30	31	32	25	26	27	28								
Side Notch	65	66	67	68	53	54	55	56	41	42	43	44	37	38	39	40				
					57	58	59	60	45	46	47	48								
					61	62	63	64	49	50	51	52								
Shear	73	74	75	76	69	70	71	72												
Bend					81	82	83	84	77	78	79	80								
					85	86	87	88												
Stamp	105	106	107	108	101	102	103	104	97	98	99	100	93	94	95	96	89	90	91	92

Figure 6: Classification of grammar rules based on their operations and degree of freedom

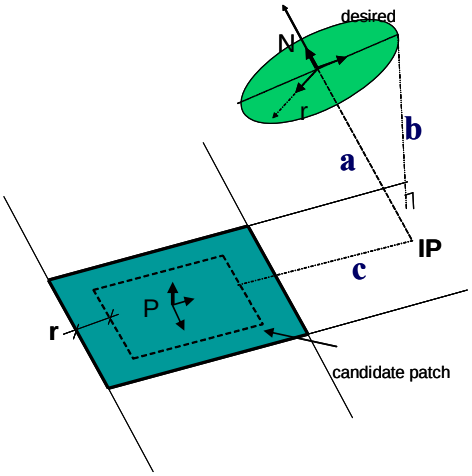

Figure 8: Evaluation of sheet metal patch compare to green circle by measuring distances a, b and c.

depends on the rule choice as well as the complexity of the candidate. As these operations are performed orthogonally to the candidate, there are four possible direction options (namely north, south, east and west) for each operation given the specific design parameter, hence number of rules resulting from shearing are 8 (2 * 4 orientations). The table in Figure 7 shows the classification of the resulting 108 design rules into two main criteria, manufacturing operations and parametric requirements. Due to the limitation on the paper size, it would not be possible to describe these rules in detail. The details can be found in author's dissertation [40] and are also available online [41]. The rows in the tables indicate rules based on their manufacturing operations and the columns represent the number of parameters required to perform.

3.2 Generation of topologies

Generation uses the representation mentioned above to generate possible designs using the graph grammar. Candidate solutions are created in generation using a recognize, choose and apply cycle. Based on the constraints for each rule, recognition makes a list of applicable rules or options. Each option is a valid execution that can be performed on the given design state and includes the rule number, where it is applied, and its orientation. This option is then selected depending on the recipe (list of options and parameters associated with those choices) which is passed along with the variables required to perform the chosen operation. The options lead to different branches on the design tree as shown in Figure 1 creating a pool of design. Upon initiation of the rule, application of the rule involves replacing the matched left-hand side with the right-hand side of the grammar rule.

The two dimensional nature of the sheet metal is exploited by ensuring all the cutting operations are done first to minimize the cost of fixtures required to perform these operations. It also reduces the time and effort required to transport the piece from one operation station to another as well as the stocking space [26] that accumulates in the production area. Therefore in accordance with the real manufacturing plants, all the cutting operations on a part must be completed before the bending is performed. The approach achieves this by taking into account the global variables named "BendX" / "BendY". Through the application of bending rules a graph is assigned a global variable which is then compared in subsequent rules to prevent any more cutting operations from being recognizing.

3.3 Evaluation

Candidates created in generation are evaluated against the multi-objective function that includes minimizing spatial constraints (Section 3.4.1), material used (Section 3.3.2), patch to patch interception (Section 3.3.3), solid to patch intersection (Section 3.3.4), manufacturing energy spent, and maximizing manufacturability.

3.3.1 Minimize Spatial Constraint

As indicated by Figure 4 the user supplies spatial requirements in the form of (green) circles that represent locations where the sheet metal is required and (red) solids for locations where the designed part must avoid contact. One of the possible approaches of comparing a candidate sheet metal's patches with the goal is to add the straight-line distance from a node relative to a goal in three dimensional space to the difference in the angles of the two planes. The disadvantages of using this kind of comparison are that it does not unify the units of the parameters, and may penalize a current candidate when the exact centers of the patch and node are not the same. There are many instances in which a sheet metal would meet the goal perfectly but not share center-points with patches and nodes.

We have come up with a method to calculate the distance between patches and goals using three distances. The value of the spatial penalty function (Ps) is just the addition of these three parameters and is thus linearly dependant on a, b and c as shown in Figure 8. In case of the green circle residing inside the boundary of the patch, all a, b and c would have a value of zero and hence no associated penalty. The higher the values for a, b and c are for a node, the farther is the design from the specified constraints. The penalty function is thus represented as:

$\mathbf{P_s = a + b + c}$

where, a is the distance from center of circle to candidate's plane, b is the distance from farthest edge of circle to candidate's plane and c is the distance from intersection point to candidate patch.

3.3.2 Minimize Material used

The overall material used in the design can be reduced either by reducing the size of initial patch that would still satisfy the constraints or by removing the material from the design by using the cutting rules. Currently the tool calculates the total volume of the design by summing the volumes of each patch and then assigns a volume penalty (P_v). This calculation assumes that the design is completely created using a uniform thickness rectangular blank. Fitness of the overall design depends on a combination of above mentioned penalties and penalties resulting from interfering with user specified constraints. The objective function is given by a weighted sum of fitness (penalty) of the design with respect to user defined spatial constraints and volume penalty. These individual weights can be then adjusted by the user depending on the application requirements.

$$\mathbf{P_v} = \sum (\text{Patch}_{\text{length}} * \text{Patch}_{\text{width}} * \text{Patch}_{\text{thickness}})$$

$$\textbf{\textit{Objective Function}} = \mathcal{W}_s^o * P_s + \mathcal{W}_v^o * P_v$$

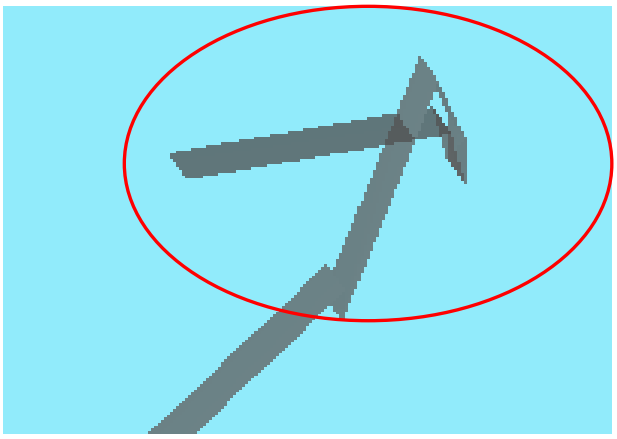

Figure 9: Design with patch to patch intersection making it impossible to manufacture.

where: $\mathcal{W}_s^o$ = weight on spatial penalty
$\mathcal{W}_v^o$ = weight on volume penalty

3.3.3 Patch to Patch Interception

Previously during our test runs, there were designs that were presented which met the spatial constraint, P_s well but were not manufacturable. Figure 9 demonstrates a topology that has the sheet impinging with itself making it impossible to manufacture and hence a bad design. To avoid such topologies from getting created and pursued further, we evaluate each topology to check for such intersection. A patch as mentioned in Figure 3 is a rectangular structure, which is divided into 2 triangles. These triangles are compared with other triangles from the candidate's patches using a fast triangle-triangle intersection algorithm [27]. If there is a patch-to-patch interception, then that candidate is assigned a heavy penalty.

3.3.4 Solid to Patch Interception

Solids are the real objects that the design supports, circumvents, or encloses. Intersection of the design with the existing objects would make it undesirable. To discourage such design, the solids are segregated into triangles which are compared using the algorithm mention in section 3.3.3. Similarly, with solid to patch intersection, candidate is assigned a heavy penalty value.
The remaining objectives of the minimizing energy and maximizing manufacturability are shown in detail in Padhye [12]. As indicated in the above equation, users can weigh these objective functions depending on their experience and requirements.

4 GUIDANCE VIA $(TP)^2$

It is our observation that most engineering problems can be divided into two main categories: one which distinguishes problems based on the accessibility of their evaluation and another category with degrees of coupling between topological changes and design parameters. These two categories form the rows and columns of Figure 10. Those under the first category are 1) search trees ending in solutions and having evaluable intermediate states, 2) search trees in which all states are potential candidates and are

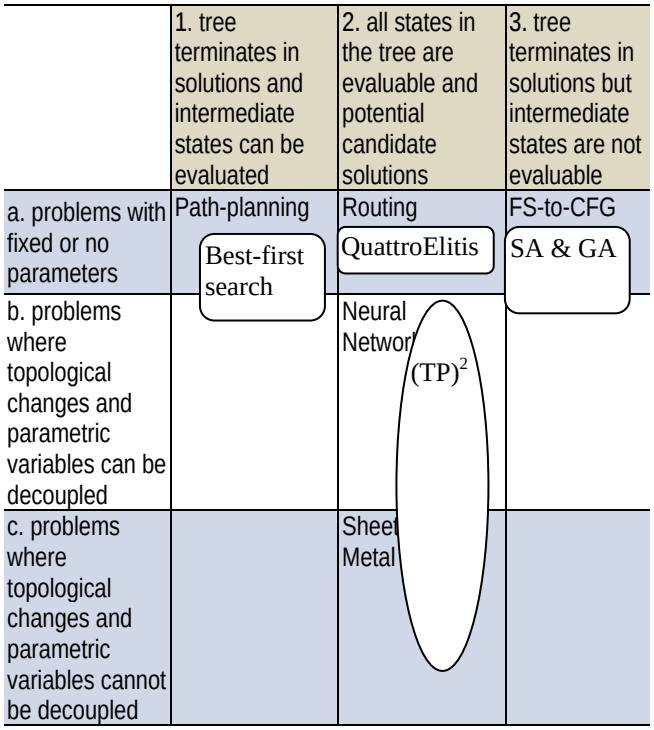

Figure 10: Topological engineering design problems can be divided six categories.

evaluable, and 3) search trees which terminate in solutions but the intermediate states are not evaluable. Those under the second category include: A) problems with fixed or no parameters, B) problems where topological changes and parameters can be decoupled and C) problems where topological changes and parameters cannot be decoupled. By plotting these on axes as shown in Figure 10, nine different problem types are created. The cell in first row and column (1A) is the simplest problem types. Problems in this area are traditionally solved by tree search algorithms such as A*. The basis of these methods is nearly forty years old but no other methods have definitively progressed into the other regions. For the most part, stochastic optimization methods such as simulated annealing [28] and genetic algorithms [29] have been attempted on these problems but have shown little success outside of the 3A region. This is because these methods are designed to solve a fixed set of discrete decision variables. It is easy enough to apply these to continuous variables, but it is not clear how to overcome the dependence on a fixed number of variables. The method discussed in this paper occupies the 2A and 2B cells of Figure 10. A previous method known as Quattro Elitism [30] was shown to solve problems in cell 2A. However, its effectiveness was not thoroughly tested and development has ceased in favor of this more general method.
Topological and Parametric Tune and Prune is a unique method that is able to solve these problems by decoupling the parametric tuning and the topological changes. $(TP)^2$ intelligently utilizes the knowledge built in the tree structure. The search process starts with a seed (user defined starting point) which can lead to a full spectrum of design solutions forming the tree like structure as shown in Figure 1. As shown in Figure 11, $(TP)^2$ is a novel

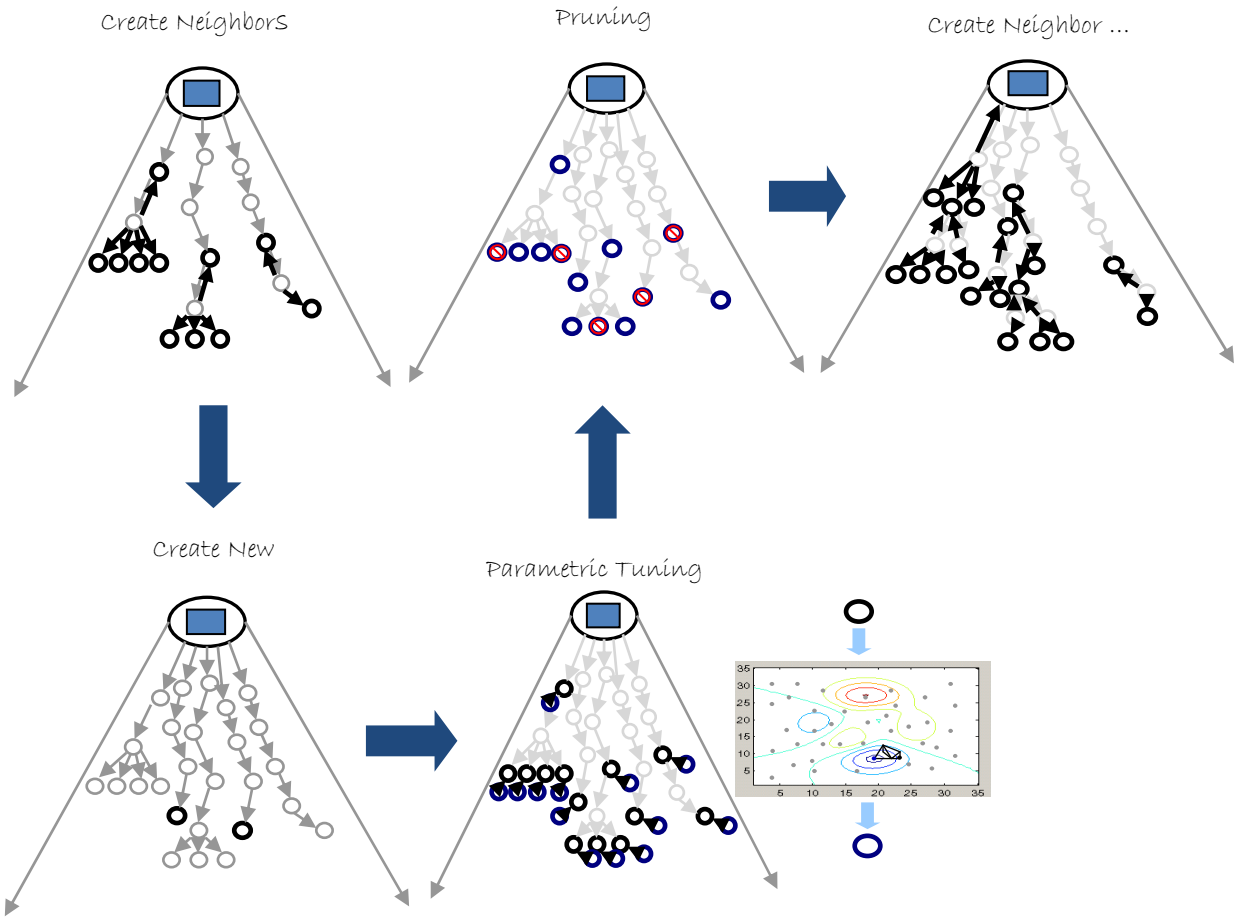

Figure 11: Topological and Parametric Tune and Prune $(TP)^2$ has four phases including create neighbors, create percentage new, parametric tuning and pruning.

approach that consists of creating all possible neighbors and new candidates followed by parametric tuning and pruning to store the best candidates.

The uniqueness of $(TP)^2$ lies in its ability to traverse through different branches of the search tree at varying levels to avoid getting stuck in local minima while exhaustively (to the extent possible due to computational limits) searching a particular branch to find the best downhill solutions for that branch. A level of the tree is defined by the number of rules that were applied to the seed to form the candidate.

The user inputs to this process are in form of following parameters: *P*, *N*, *m*, and *MaxI* (or *MaxTime*). The number of candidates that are generated in each iteration is defined by the variable P which is the overall allowed population. The population size determines tradeoff between exploration versus exploitation of the search space, with increasing values of the population size, the resulting search is more exploratory. Currently for the sheet metal design problem, we have tested values of P from 1000 to 7000 candidates in a particular run. The value of P is restricted by the computation memory and overall time allocated for evaluation and tuning of the candidates. The size s of the list L_c and the value of P determine the limit of neighbors 'c' that a particular candidate can produce (Section 4.1).

The flowchart in Figure 12 illustrates overall guidance and generation process that is involved in searching the space. The search starts with a user-defined seed that is add to the list of candidate L_c. Each candidate in the list is then expanded to include all possible child candidates to that state by invoking all valid rule options that are recognized on that candidate. These newly formed neighbors along with their parent candidates are added to L_c. The average level of L_c is computed to determine the average number of rules that are used to form candidates in the list.

To continuously explore new branches in the tree a percent of new candidates 'N' are created. N is currently set to 25% for the problem in Section 4. The average level of L_c is used to determine the number of random rules application on the seed. The newly formed random candidates using the 'create new' phase (Section 4.2) are integrated with L_c. The newly-formed are assigned random values for their included parameters.

These design variables for each candidate are optimized in 'parametric tuning' phase as mentioned in Section 4.3. Candidates with optimized parameters are reinserted into the list L_c. Each

candidate in L_c is evaluated based on the user defined objective function. The underperformed candidates are prune as mentioned in Section 4.4 to number of solutions to keep defined by variable m The candidates surviving into the next generation are used to create the next set of adjacent candidates. The process stops if either of convergence criteria currently defined by the maximum number of iterations, *MaxI* or overall run time of the process (Max Time) is met.

4.1 Create Neighbors

In the 'create neighbors' phase as shown in top left corner of Figure 11, every candidate in the population generates all adjacent solutions that are allowed considering the computational limitation. Adjacent solutions are those that are one grammar rule application away from the original state. This phase allows the search to find the best possible downhill solution/direction from a particular point in the search space while maintaining the varying complexity of the space. The generation of candidates consists of the recognition of all possible options. The maximum number of options c is determined by the ratio of P and s. The initial size s of the list is equivalent to m for all iteration except for the first iteration where it is one (just for the seed). If the number of options available is more than c, then a randomly chosen subset is found. This happens more often as we traverse deep into the tree as there are more numbers of nodes within a candidate. All possible neighbors in this context refer to states that not only are created by applying a grammar rule but also by undoing the last rule to recreate the parent that led to this candidate. The effect of this step is to exploit the region around each candidate within the population regardless of the position of the candidate within the search tree.

4.2 Create New

In create neighbor phase as shown in Figure 11, almost all the possible options for a particular candidate gets evaluated later and only the few best are selected for further search. This step cannot guarantee that there are no good solutions possible from the discarded options. To avoid this influence of initial choices of rule selection made in 'create neighbor' phase and to explore the search space more thoroughly, during 'create new' candidates are created by following a new sequence of rule choices applied to the initial patch. The average of all the levels of the candidates is computed and used to generate random candidates in this 'create new' phase. This means that if, for example, the average level for the set of candidates at a particular iteration is four then new candidates are generated by executing four random rules from the initial seed. The number of new candidates N is a fraction of the overall population, as the number of iterations goes up so does the number of new solutions that are created.

4.3 Parametric Tuning

The list of candidate solutions that are created from above mentioned steps are tuned for best possible parametric values in this step as indicated at the bottom center of Figure 11. As seen in Figure 1 each operation based grammar rule requires a set of parameters to define a topology. These parameters govern the resulting topology, and also the values of parameters for subsequent rules, and in some cases the rules themselves. The resulting design is dependent as much on the parameters that are passed to the operations as the grammar rules choices. Hence, it is very essential to optimize or tune these parameters in order to continuously find

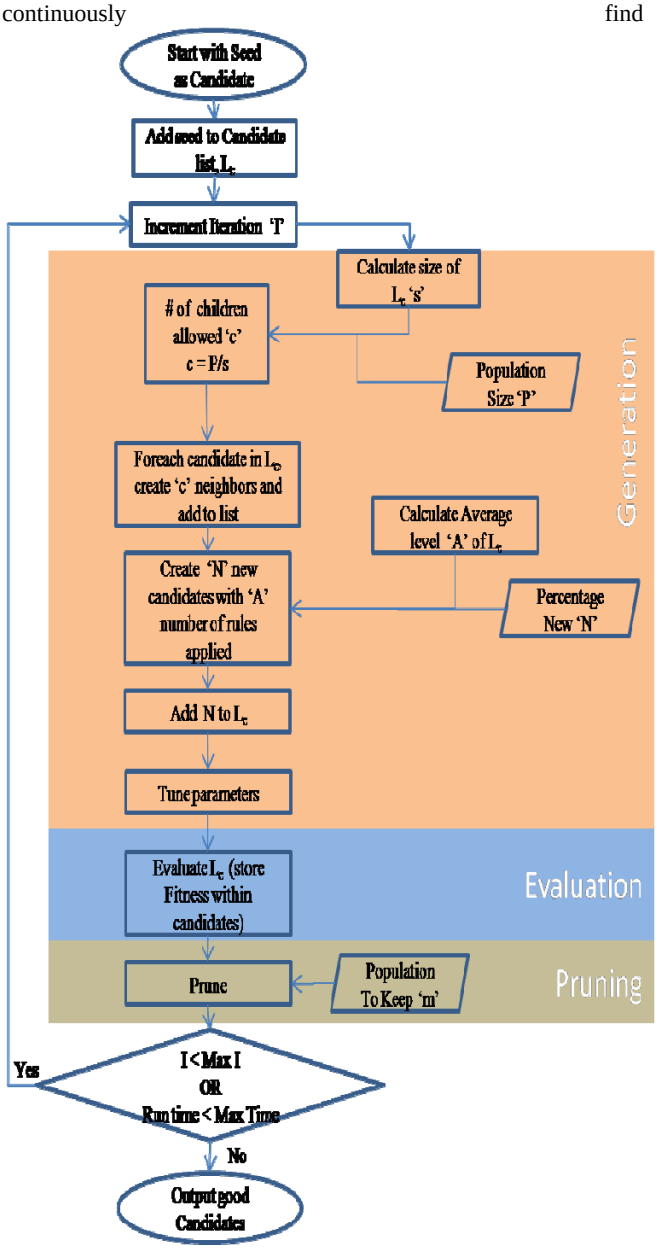

Figure 12: Flowchart detailing the search process.

better designs. For instance, designing bi-leg bracket has parameters such as angle of the bend (that can vary from 0 to 360) as well as length and width of the legs. These parameters can be changed / tuned to better meet the requirements mentioned in Section 3.1.

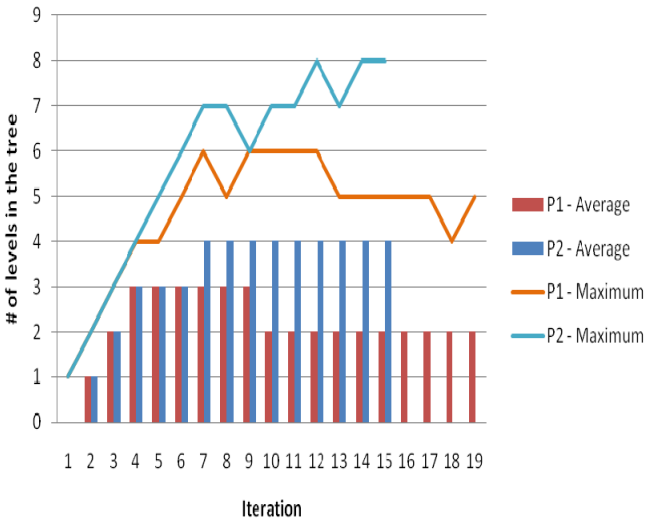

Figure 13 : Traversing through a tree during a run.

Our approach involves finding out the unique parameters that were used to form the candidate. These "n" variables are inputted to the following optimization technique. A Latin Hypercube sampling [31, 32] is used to generate a distribution of plausible parameter sets to cover the complete search space. When sampling a function of n variables, the range of each variable is divided into M equally probable intervals. This interval determines how fine or coarse is the grid that covers the search space. All these sample points are evaluated and a fitness value is assigned to each of the points. The sample with the best fitness is used as a new starting point for a Nelder Mead optimization [33]. This downhill gradient-free method leads to the local minima for the chosen candidate topology and replaces that candidate in the list. This combination of Nelder-Mead and Latin Hypercube sampling is chosen for its robustness and scaling behavior. Other optimization methods were considered, but what is needed is method that can be invoked automatically by the process and that works on problems of varying numbers of variables. With Latin Hypercube, we can avoid scaling issues as the number of variables increases (i.e. it is always M points). Since Nelder-Mead does not require gradients, we avoid the costly and error prone finite-difference gradient formulations that require 2n additional function evaluations per iteration.

4.4 Pruning

A probabilistic selection method built directly from the proportional selection method [34] is used to *prune* away unpromising solutions based on the fitness values of the candidates. This pruning method shares similarity to the proportional selection method more commonly used in evolutionary computational methods such as Genetic Algorithm and Simulated Annealing. This unique approach is capable of changing the selection process from random, through stochastic, to deterministic. Using the value of the fitness of the candidates, the selection method prunes and rearranges the list to the number of solutions to keep, m. Candidates that survived are again used to create all possible neighbors. As shown in Figure 11 this cycle goes on to create solutions by traversing through different levels of the tree.

One key benefit of this algorithm lies in finding the optimum level of the tree to search depending on the complexity of the problem.

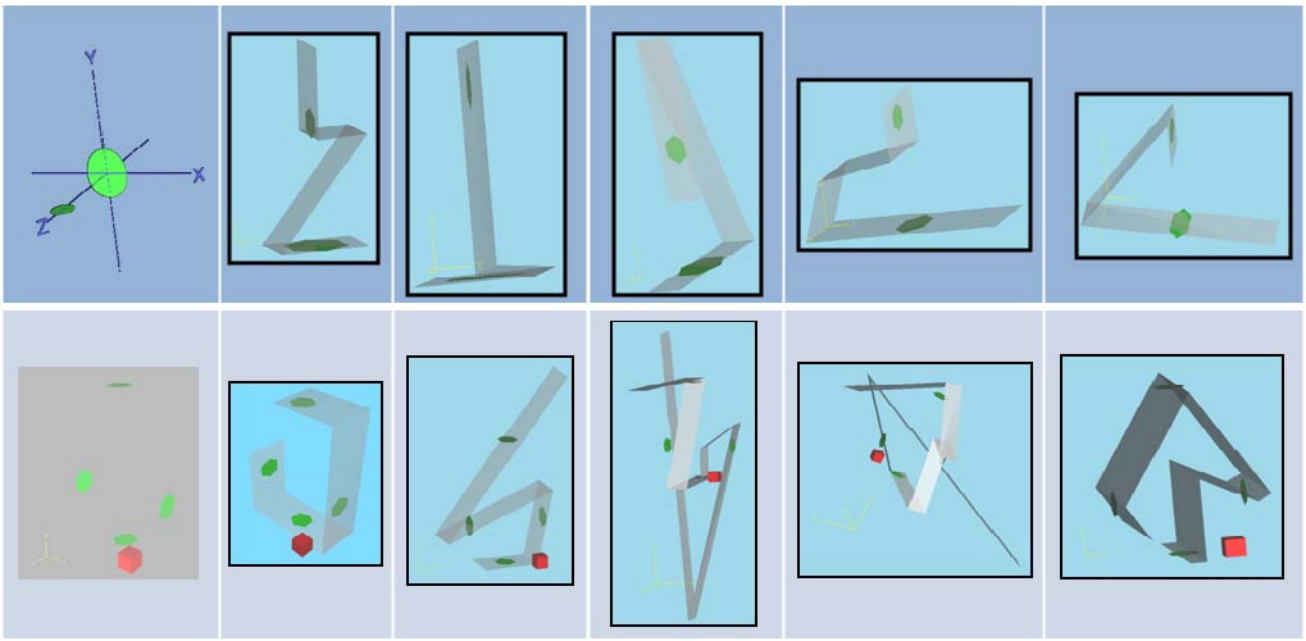

Figure 14: Test problem showing the spatial constraints and the resulting topologies using the universal guidance method.

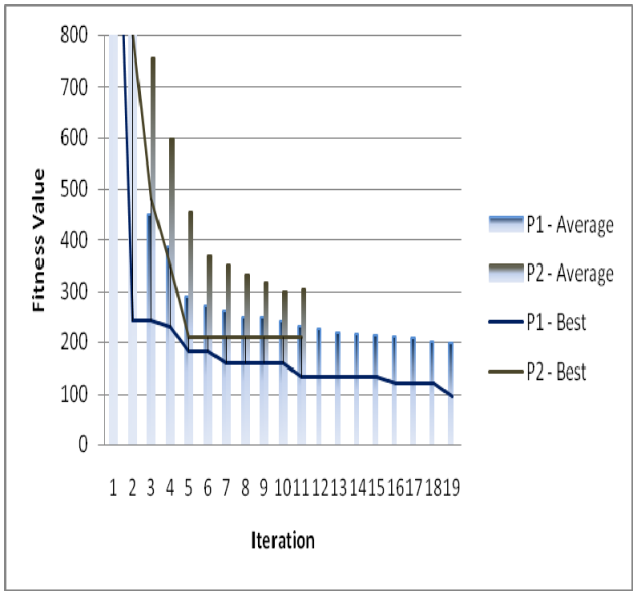

Figure 15: Bars represents the average fitness and lines indicates the best fitness value for each fitness.

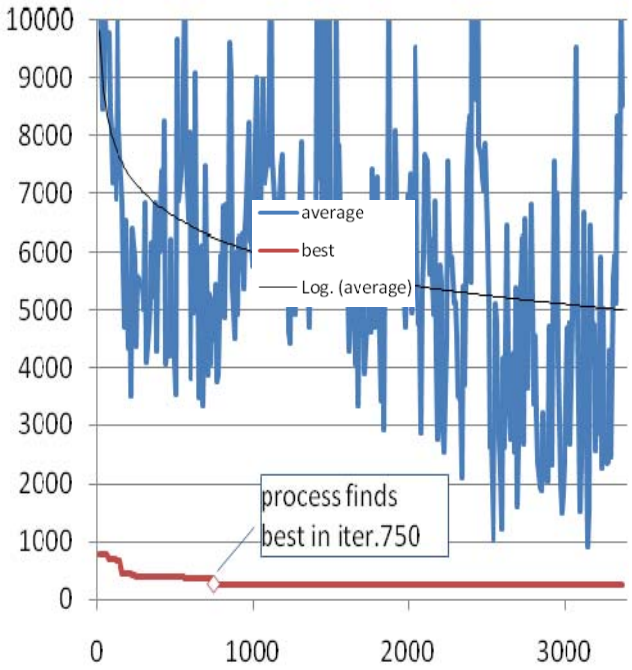

Figure 16: Average and the best fitness per iteration for problem 1 (P1) using Genetic Algorithm.

Figure 13 shows average and maximum number of rules performed in each iteration for the two test problems mentioned in Section 4. The problem, P1, is simpler in terms of the number of spatial constraints than P2. The reflection of this is seen in the figure too as more number of rules are required to be performed to find optimal solutions for P2 as indicated by line graph representing maximum levels in the figure. The maximum number of levels for both the problem fluctuates as the search is not restricting itself from going to different levels of the tree to find the optimum solution. Another thing worth noting is that the average of the run stagnates, indicating that the search has found a good level of the tree to search for better solutions. The average for both the problem is represented as bars in the figure.

5 IMPLEMENTATION AND RESULTS OF EXAMPLE PROBLEMS

The preceding guidance strategy has been implemented in synthesizing sheet metal parts. An approach [35, 36] has been developed to automate the synthesis of novel sheet metal parts while taking into account their manufacturability. The design process starts with a sheet metal blank that is represented by a rectangular patch with attributes like width, height and uniform thickness. The candidate solutions depend on the choices made in terms of the design rules and the parameters. As shown in Figure 1, if the first chosen rule is to perform westerly slitting operation, there are perhaps infinite possible designs that can be generated based on the values for depth and height of the slit. Hence, a wide range of the feasible solutions can be created transitioning through the levels of the design tree. This graph grammar approach to sheet metal design not only encapsulates a wide range of feasible sheet metal solutions but it also affords us the ability to create automated search tools to find an optimal part for a set of user-defined specifications.

Topological and Parametric Tune and Prune method discussed above is used to provide the guidance for searching the design space more efficiently. Hence, a wide range of the feasible solutions can be created transitioning through the levels of the design tree. Test problems resulted in lots of novel solutions to satisfy the user-defined constraints, some of them are listed in Figure 14. Here the user defines spatial constraints in form of green circles and red solids. The green circles represent the location where the designer wants the sheet metal to be present, which can result in providing support to the objects that are enclosed in the design and also material required for the fixtures/welds. The red solids represent other parts, which the designed sheet metal part must not interfere with. Feasibility is thus defined by avoiding the red solids and passing through green circles. Two distinct problems as mentioned in the first column of Figure 14 are tested using $(TP)^2$. P1 is a simple problem with 2 goals arranged in a 'T' and P2 represents the one with four goals and a red solid.

Figure 15 shows the average and best fitness values for a 15 hours run for both the problems. As seen the average as well as the best fitness value per iteration follows the same trend of improvement. The population size P for this particular run was 5000 candidates per iteration. Few of the many solutions that were produced for these problems are shown in Figure 14. Before running these test problems, we derived a few topologies that would satisfy the goals; interestingly the approach came up with much different topologies than what we created. These results from the experiment are encouraging as our research intent is to provide

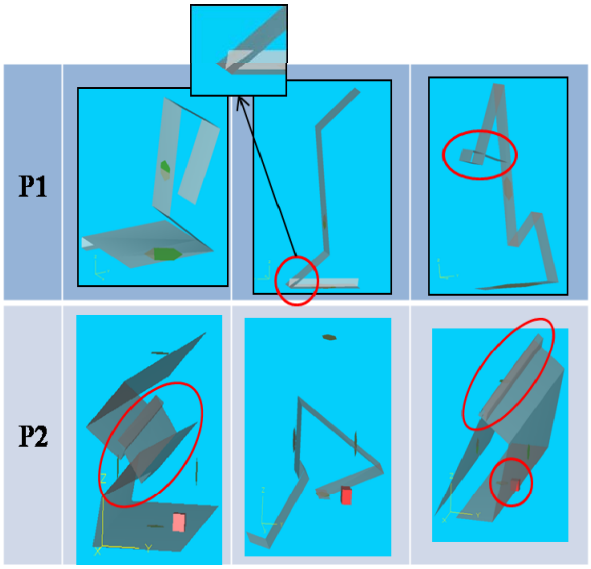

Figure 17: Best possible topologies resulting from using Genetic Algorithm.

designer with as many possible designs that can satisfy the spatial constraints.

6 COMPARING $(TP)^2$ WITH GENETIC ALGORITHM

The above mentioned two problems are tested using a genetic algorithm as a guidance strategy. Many experiments were run to determine the best values for population size, mutation and crossover rates. All the other parameters and settings were kept the same as $(TP)^2$ to give it a fair comparison. The best topologies resulting from using the genetic algorithm are shown in Figure 17. Even though some of the designs did satisfy the user defined constraints to an extent, they are infeasible (impossible to manufacture) due to the impingement of the patches with each other (circled in red in Figure 10). After a run of 15 hours, using a genetic algorithm, fewer distinct designs were produced in comparison to what $(TP)^2$ produced.

The best (red line) and average (blue line) fitness values for each iteration with a population size of 20 candidates is shown in Figure 16. This is created by averaging three separate 15 hour runs. The genetic algorithm provides the same trend of improvement in terms of the best value as $(TP)^2$; however, in comparison to $(TP)^2$, the genetic algorithm fell short in values. The best fitness for P1 is 260 compare to 98 using $(TP)^2$ and for P2, it is 1632 compare to 210 using $(TP)^2$. While the best fitness follows the same trend as $(TP)^2$, the average is fluctuates significantly for the GA. This shows randomness in the Genetic Algorithm and its inability to take advantage of the information that is built in the problem. The GA was able to find the solution with fitness value of 260 (the best that Genetic Algorithm could find for P1) at 750^{th} iteration and after that there was no sign of improvement for the next 2750 iterations.

7 DISCUSSION AND FUTURE WORK

In this paper, we discuss a method to automatically synthesize the sheet metal components for a set of spatial constraints. The method is unique in that it combines a graph based optimization with a shape grammar to create only the feasible sheet metal components. Unlike commercial available software which aids the user in determining the sequence of manufacturing operations for a specified component, our approach only starts with spatial constraints in order to create component topologies. The results currently obtained are not only solving the design problem but they are also promising in terms of uniqueness and can be improved to incorporate the complex design problems. Previous attempts to use a genetic algorithm were unsuccessful as the process took over 48 hours run, developed far inferior solutions to those shown above, and all had the same level of complexity (i.e. same level of the tree). This issue is resolved by creating a new method that leverages the inbuilt knowledge of the design problem in terms of representation and uses it to glean better designs and shorten the search time. The tree structure (Figure 1) comprises a lot of information which can be exploited to increase the efficiency of the exploration process.

At the time of writing, this research has only addressed simple problems but in the coming year the research will be much more beneficial when it is capable of solving more complex design problems with all the objective functions including minimizing spatial constraints, material and fabrication energy, and maximizing the manufacturability. In order to achieve better designs in less computational time, new search strategies such as $(TP)^2$ need to be developed that can take advantage of grammar rule spaces and the included parameters. Next, the 'generate all neighbors' step of generation needs to be re-evaluated. It is possible that certain rules may perform better at specific times on the candidate and thus the creation of a to-do or taboo list to assist further choices may improve the process. Manufacturability constraints are inherently part of the design synthesis process as the grammar rules. Although manufacturability is considered throughout the approach, the main objective here is to create solutions in a challenging design space constrained by manufacturing issues. This, to us, is more intriguing as the design space for such engineering problems is extremely large and complex. Another issue with such problems is that it is very difficult to decouple the parameters from the topologies. Hence, much of our research is focused on tackling these issues. Manufacturability of the part is considered in formulating the grammar rules and during the assignment of parameters and the evaluation of the resulting candidates. The complexity of the design is determined by the number of bends and the angles of the bends. Incorporating such additional manufacturing complexity into the evaluation process would increase the possibility of the created designs being manufacturable.

A unique approach for optimizing problems that includes topology changes and parametric tuning is developed and discussed in this paper. Topological and Parametric Tune and Prune in its 'Create Neighbors' and 'Create New' phases capture various ways to explore and exploit the search space simultaneously. The 'Create Neighbors' phase is used to exploit the search space locally by finding the best possible branches to follow in the subsequent iteration while the 'Create New' phase randomly follows different branches of the tree to create new

solutions, hence exploring the search space. For the engineering problem that $(TP)^2$ has been tested on so far, it has proven more efficient and effective compared to Genetic Algorithm. By using graphs to represent a design as opposed to a vector (or other genotype representation such as bit-string), it becomes possible to encapsulate more information and thus generate a richer set of candidates. This representation of information is utilized in $(TP)^2$, to provide better guidance strategy for the search process.

The $(TP)^2$ results are promising there are a number of new improvements that are being pursued. Currently, if the number of allowed neighbors is less than the number of recognized options, the process randomly choose a set of options to apply. Otherwise, it applies all the recognized options to the candidate to create all neighbors. Due to the randomness associated with the current choice method, there is no control on the selection of the branches to follow. As a next step to $(TP)^2$, a method needs to be developed to learn and target specific branches that are likely more fruitful. This can be done by creating a Taboo [25] list for rules that did not performed well and assigning less probability of it been chosen. Also similarly, the good rules can be favored by forming a TODO [37] list. Also, the authors believe that stochastically choosing the number of rules to be applied in 'Create New' phase instead of averaging the overall level of population would increase the variability in the solutions. It is anticipated that these changes to generation will improve the efficiency of $(TP)^2$ (and that these changes will be completed and tested for presentation at this conference).

The graph based optimization technique mentioned in this paper will be applied to multiple domains to find its effectiveness over a varied range of engineering problems. Routing problems [30], resistive networks [38], and neural networks [39] are application where graph grammars are already in place, so implementing $(TP)^2$ on these problems can be easily completed. Currently, these applications are using optimization techniques such as QuattroElitism and Genetic Algorithm with BFGS for providing guidance to the search space. Implementing $(TP)^2$ for these problems will provide a good comparison to other optimization techniques other than Genetic Algorithm.

$(TP)^2$ is unprecedented in its approach to traverse through the search space. Unlike other optimization techniques that initially explore and later exploit the search space, $(TP)^2$ simultaneously explores the space by creating new candidates in each iteration and exploits the local neighborhood for each candidate. As describe in the Figure 10, the area under 2B and 2C are most challenging for stochastic based algorithms such as SA and GA. These algorithms are unable to decouple topological and parametric changes which lead to a weak search, as some initial decisions made in the search would affect the final outcome. $(TP)^2$ has overcome this problem by tuning the topological changes first, followed by optimizing the parameters for each iteration leading to a creamier set of candidates.

8 ACKNOWLEDGMENTS

The authors would like to thank the National Science Foundation for supporting this work under grant award number DMI-044880
.

9 REFERENCES

[1] Pro/Sheetmetal, "The Official Pro/Sheetmetal Site," in *http://www.ptc.com/products/proengineer/nc-sheetmetal*.

[2] A. HyperForm, "The Official Altair HyperForm Site."

[3] D. P. Bertsekas, *Nonlinear Programming*. Cambridge, MA: Athena Scientific, 1999.

[4] S. J. Russell, *Artificial Intelligence: Modern Approach*. Englewood Cliffs, N.J.: Prentice Hall, 1995.

[5] J. Cagan, M. I. Campbell, S. Finger, and T. Tomiyama, "A Framework for Computational Design Synthesis: Model and Applications," *Journal of Computing and Information Science in Engineering*, vol. 5, pp. 171-181, 2005.

[6] M. I. Campbell, S. Nair, and J. Patel, "A Unified Approach to Solving Graph Based Design Problems," in *Proceedings of the ASME Design Engineering Technical Conferences and Computers in Engineering*, Las Vegas, 2007.

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

[8] C. H. Wang and D. A. Bourne, "Design and manufacturing of sheet-metal parts: using features to aid process planning and resolve manufacturability problems," *Robotics and Computer-Integrated Manufacturing*, vol. 13, pp. 281-294, 1997.

[9] C. H. Wang and R. H. Sturges, "BendCad: a design system for concurrent multiple representations of parts," *Journal of Intelligent Manufacturing*, vol. 7, pp. 133-144, 1996.

[10] S. K. Gupta, D. A. Bourne, K. H. Kim, and S. S. Krishnan, "Automated Process Planning for Sheet Metal Bending Operations," *Journal of Manufacturing Systems*, vol. 17, pp. 338-360, 1998.

[11] A. Soman, S. Padhye, and M. I. Campbell, "Towards an Automated Approach to the Design of Sheet Metal Components," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 17, pp. 187-204, 2003.

[12] S. Padhye and M. I. Campbell, "Evaluation Method for the Topological Synthesis of Sheet Metal Components," in *Proceedings of the ASME Design Engineering Technical Conferences and Computers in Engineering*, Salt Lake, UT, 2004.

[13] A. Soman and M. I. Campbell, "A Grammar-Based Approach to Sheet Metal Design," in *Proceedings of the ASME Design Engineering Technical Conferences and Computers in Engineering*, Montreal, Quebec, Canada, 2002.

[14] K. N. Brown and J. Cagan, "Optimized process planning by generative simulated annealing," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 11, pp. 219-235, 1997.

[15] S. Emdanat, E. G. Vakalo, and W. Birmingham, "Solving Form-Making Problems Using Shape Algebras and Constraint Satisfaction," *Architectural Computing from Turing to*, pp. 15-17, 2000.

[16] H. Ehrig, Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series). library: Springer, 2006.

[17] *Formal Engineering Design Synthesis*. Cambridge, UK ; New York: Cambridge University Press, 2001.

[18] H. Ehrig, H. J. Kreowski, A. Maggioloschettini, B. K. Rosen, and J. Winkowski, "Transformations of Structures - an Algebraic

Approach," *Mathematical Systems Theory*, vol. 14, pp. 305-334, 1981.
[19] M. Nagl, "Bibliography on Graph Rewriting-Systems (Graph-Grammars)," *Lecture Notes in Computer Science*, vol. 153, pp. 415-448, 1983.
[20] L. Bolc and J. Cytowski, *Search Methods for Artificial Intelligence*. London: Academic Pr, 1992.
[21] W. D. E. Lawler E.L., "Branch-And-Bound Methods: A Survey," *Operations Research, JSTOR, ,* 1996.
[22] D. H. Myszka, *Machines and Mechanisms: Applied Kinematic Analysis (3rd Edition)*. Upper Saddle River, NJ: Prentice Hall, 2005.
[23] L. C. Schmidt and J. Cagan, "Optimal Configuration Design: An Integrated Approach Using Grammars," *Journal of Mechanical Design*, vol. 120, pp. 9-Feb, 1998.
[24] G. S. Hornby, "Functional scalability through generative representations: the evolution of table designs," *Environment and Planning B: Planning and Design*, vol. 31, pp. 569-587, 2004.
[25] M. I. Campbell, J. Cagan, and K. Kotovsky, "The A-Design approach to managing automated design synthesis," *Research in Engineering Design*, vol. 14, pp. 12-24, 2003.
[26] M. D. McKay, R. J. Beckman, and W. J. Conover, "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output From a Computer Code," *TECHNOMETRICS*, vol. 42, pp. 55-61, 2000.
[27] T. Möller and B. Trumbore, "Fast, minimum storage ray/triangle intersection," 2005.
[28] S. Kirkpatrick, C. Gelatt Jr., and M. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-679, 1983.
[29] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison Wesley, 1989.
[30] S. Nair and M. I. Campbell, "A Graph Topology Optimization Method for the Synthesis of Robust And Efficient Routes," in *International Conference on Engineering Design, ICED'07*, Paris, France, 2007.
[31] R. L. Iman, J. E. Campbell, and J. C. Helton, "An approach to sensitivity analysis of computer models. I- Introduction, input, variable selection and preliminary variable assessment," *Journal of Quality Technology*, vol. 13, pp. 174-183, 1981.
[32] R. L. Iman, J. M. Davenport, and D. K. Zeigler, *Latin Hypercube Sampling (Program Users's Guide)*: SAND-79-1473, Sandia Labs., Albuquerque, NM (USA), 1980.
[33] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, pp. 308-313, 1965.
[34] M. Campbell, Swantner, A., Patel, J.,, "A Unified Approach to Selecting and Ordering Candidates in Stochastic Search," *AAAI*, 2008.
[35] J. Patel and M. I. Campbell, "Automated Synthesis of Sheet Metal Parts by Optimizing a Fabrication Based Graph Topology," in *1st AIAA Multidisciplinary Design Optimization Specialist*, Austin, TX, 2005.
[36] J. Patel and M. I. Campbell, "An approach to automate concept generation of sheet metal parts based on manufacturing operations," in *Proceedings of the ASME 2008 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Brooklyn, New York, 2008.
[37] M. I. Campbell, J. Cagan, and K. Kotovsky, "Learning From Design Experience: Todo/Taboo Guidance," in *Proceedings of the ASME Design Engineering Technical Conferences and Computers in Engineering*, Pittsburgh, PA, 2001.
[38] S. Nair, "A Graph Theory Approach to the Synthesis and Optimization of a modified Transportation Network," in *Dept. of Mechanical Engineering*. vol. Master of Science Austin: University of Texas, 2007, p. 84.
[39] C. Vempati and M. I. Campbell, "A Graph Grammar Approach to Generate Neural Network Topologies," in *Proceedings of the ASME Design Engineering Technical Conferences and Computers in Engineering*, Las Vegas, 2007.
[40] J Patel, " *"Automated Synthesis of Sheet Metal Parts by Optimizing a Manufacturing Based Graph Grammar Rules* " in *Dept. of Mechanical Engineering*. Doctor of Philosophy Austin: University of Texas, 2008, p. 35 and appendix A.
[41] Graphsynth Sheet metal Website, "http://www.graphsynth.com/sheetmetal"