Bicriteria Product Design Optimization: An Efficient Solution Procedure Using AND/OR Trees

S. Raghavan,[1] Michael O. Ball,[2] Vinai S. Trichur[3]

[1] *The Robert H. Smith School of Business, Van Munching Hall, University of Maryland, College Park, Maryland 20742*

[2] *The Robert H. Smith School of Business and Institute for Systems Research, University of Maryland, College Park, Maryland 20742*

[3] *I2 Technologies, One Cambridge Center, Cambridge, Massachusetts 02142*

Received 31 October 2000; revised 15 January 2002; accepted 15 January 2002

Abstract: Competitive imperatives are causing manufacturing firms to consider multiple criteria when designing products. However, current methods to deal with multiple criteria in product design are ad hoc in nature. In this paper we present a systematic procedure to efficiently solve bicriteria product design optimization problems. We first present a modeling framework, the AND/OR tree, which permits a simplified representation of product design optimization problems. We then show how product design optimization problems on AND/OR trees can be framed as network design problems on a special graph—a directed series-parallel graph. We develop an enumerative solution algorithm for the bicriteria problem that requires as a subroutine the solution of the parametric shortest path problem. Although this parametric problem is hard on general graphs, we show that it is *polynomially solvable* on the series-parallel graph. As a result we develop an efficient solution algorithm for the product design optimization problem that does not require the use of complex and expensive linear/integer programming solvers. As a byproduct of the solution algorithm, sensitivity analysis for product design optimization is also efficiently performed under this framework. © 2002 Wiley Periodicals, Inc. Naval Research Logistics 49: 574–592, 2002; Published online in Wiley InterScience (www.interscience.wiley.com). DOI 10.1002/nav.10031

1. INTRODUCTION

Manufacturing firms today are faced with an increasing array of choices and decisions when designing a product. Furthermore, competitive imperatives are causing firms to shift their strategic focus from one of excellence on a single front (for instance, being a low cost producer, or being a high quality producer) to one where different objectives are prioritized and traded off, in an effort to better fit narrower market niches. This shift in focus has resulted in the need to explicitly incorporate an increasing number of typically downstream product life cycle considerations (such as design costs and manufacturing yields) into the decision-making process at the design stage. Thus, the product design problem, which was traditionally concerned only with the functionality of the product, is now more accurately modeled as a multicriteria discrete optimization problem.

Correspondence to: S. Raghavan

In this paper we describe a modeling framework for product design that permits product managers to efficiently approximate the set of *Pareto optimal* solutions for the *bicriteria* product design problem. In the context of bicriteria optimization the term *efficient,* or Pareto optimal, solutions refers to the set of solutions that are not *dominated* by any other solution in both criteria. As an example in the manufacturing application described in this paper, the two criteria under consideration are cost, which we wish to minimize, and manufacturing yield, which we wish to maximize. In this situation a solution S with cost C_S and yield Y_S is Pareto optimal to the bicriteria problem if there is no other feasible solution R to the problem with both a lower cost and a higher manufacturing yield [i.e., $C_R \leq C_S$ and $Y_R \geq Y_S$ and $(C_R, Y_R) \neq (C_S, Y_S)$]. Such solutions are important in multicriteria analysis due to the fact that irrespective of how a decision-maker trades off various criteria, one of these Pareto optimal solutions will be preferred. In bicriteria optimization one is interested in presenting decision makers the set of Pareto optimal (or efficient) solutions.

Bicriteria optimization problems are often solved by modeling them as parametric (objective) optimization problems. This is achieved by setting up a parameter λ, which varies from 0 to 1, that combines the two criteria into a single objective function. When $\lambda = 0$, the objective function represents one criterion, and, when $\lambda = 1$, it represents the other criterion. Values of λ between 0 and 1 represent convex combinations of the two objectives. For the example involving cost and yield, the parametric objective (that we wish to minimize) is setup as $\lambda C_S - (1 - \lambda)Y_S$. In parametric optimization problems one is interested in identifying the set of (nondegenerate) optimal solutions as λ varies from 0 to 1. It is well known that the optimal solutions to the parametric optimization problem are Pareto optimal solutions to the bicriteria problem. If the decision maker trades off the various objectives linearly, i.e., has a linear utility function (this is a reasonable assumption in many instances), then the solutions to the parametric problem and the Pareto optimal solutions coincide. Otherwise the solutions to the parametric problem are a subset of the Pareto optimal solutions and serve as an approximation to the efficient frontier.[1]

We begin our presentation by reviewing a simple model, the AND/OR tree, first introduced in the context of product design by Trichur and Ball [26]. This model makes explicit the decisions involved in designing a product, without specific reference to either the consequences of these decisions or the interactions between them. We then show an equivalent network representation of the AND/OR tree, by transforming the AND/OR tree into a directed series-parallel graph. Further, we establish a one-to-one correspondence between feasible solutions to the product design optimization problem modeled using the AND/OR tree and paths between two specified nodes on the directed series-parallel graph. As a result, we show how product design problems utilizing this framework can be cast as a network design problem—either a shortest path problem or a more complex variant of it where the path cost is a function of the arcs on the path.

The shortest path connection allows one to devise computationally efficient solution techniques for bicriteria product design problems modeled via this approach. The solution procedure models the bicriteria design problem as a parametric optimization problem. It is based on the observation that fixing some of the choice variables in the product design problem results in a shortest path problem. Thus, it enumerates these choice variables (there are a small number of

[1]In certain cases, like bicriteria linear programming problems, there is a one-to-one correspondence between the solutions to the parametric problem and the Pareto optimal solutions. However, for discrete optimization problems, like the product design optimization problem or even the shortest path problem, this does not generally hold.

them), and so the core of the solution procedure calls for the repeated solution of the *parametric shortest path problem.*

Several researchers [2, 11, 21, 28] have studied network flow problems on directed series-parallel graphs, and described polynomial time dynamic programming algorithms for solving them. Since the shortest path problem is a special case of the minimum cost network flow problem, it can be solved in $\mathcal{O}(|A|)$ time, where $|A|$ is the number of arcs, using the algorithms described therein. The parametric shortest path problem is known to be *hard* on general graphs in the sense that there may be up to $\mathcal{O}(|V|^{\log|V|})$ paths that must be found to solve the parametric problem [9], where $|V|$ is the number of vertices in a graph. Thus, for general graphs, it is not polynomially solvable irrespective of whether $\mathcal{P} = \mathcal{NP}$ (since the size of the output is not polynomial). In this paper, we generalize the dynamic programming algorithm and show that the parametric shortest path problem can be solved efficiently in $\mathcal{O}(|A|^2)$ time on a directed series-parallel graph. (Note, however, the algorithm for the bicriteria product design problem has exponential worst-case complexity, since it is an enumerative algorithm). Importantly, our result also shows that the parametric value function for the shortest path problem on a directed series-parallel graph *has at most* $|A| + 1$ *breakpoints* (for general graphs Gusfield [9] shows that the value function can have up to $\mathcal{O}(|V|^{\log|V|})$ breakpoints). Consequently, we show that an alternate technique for parametric optimization, due to Eisner and Severance [5], when applied to this problem also has an $\mathcal{O}(|A|^2)$ running time.

Our approach has several advantages. First, it provides a systematic way to describe explicitly the decisions involved in designing a product. Second, the AND/OR tree and its corresponding network formulation provide additional insight into the problem structure, allowing for the development of efficient algorithms for product design problems that are modeled using our framework. Third, bicriteria analysis of the product design problem can be efficiently performed within this framework (either exactly under the assumption that product managers have linear utility functions or as an approximation otherwise). Fourth, since sensitivity analysis is essentially a parametric optimization problem, (objective function) sensitivity analysis for product design optimization can be efficiently carried out under this framework. Finally, our approach allows for the development of algorithms that do not require the use of commercial LP or IP solvers. Avoiding the use of commercial LP or IP solvers can, in many instances, significantly drive down the cost of a software product.

The rest of this paper is organized as follows. At the conclusion of this section we provide a brief literature overview. In Section 2 we describe the AND/OR framework for product design and establish the correspondence between AND/OR trees and directed series parallel graphs. In Section 3 we describe the application of this framework to the design of printed circuit board assemblies at a FORTUNE 100 company. We also outline the solution procedure that requires, as a subroutine, the solution to a parametric shortest path problem. In Section 4 we present our polynomial time solution algorithm to the parametric shortest path problem on a directed series-parallel graph, thereby providing the core algorithm for the solution procedure to the bicriteria product design problem. We conclude in Section 5 with a discussion on some suggested extensions and directions for further research. In particular we discuss the extension of the solution procedure to multicriteria (i.e., with more than two objectives) product design problems.

1.1. Literature Overview

Krishnan and Ulrich [14] give a comprehensive survey of the product development literature. Other useful survey articles are by Finger and Dixon [7, 8] on detailed design decisions, and by

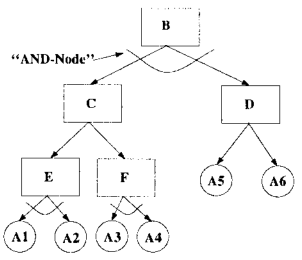


Figure 1. Modeling a hierarchical system via an AND/OR tree.

Papalambros [18] on using mathematical programming approaches, typically involving nonlinear programming models, to solve parametric design problems.

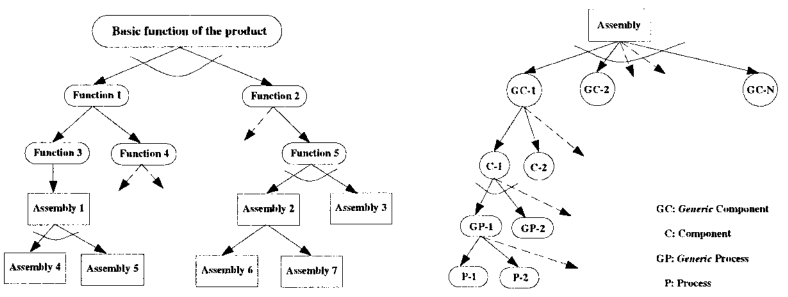
Several researchers [3, 10, 12, 15] have proposed algorithms for the bicriteria shortest path problem. In particular, Mote, Murthy, and Olson [15] describe an approach for the bicriteria shortest path problem that involves the use of a (simplex based) parametric shortest path algorithm to identify a subset of the Pareto optimal solutions. It then uses a label-correcting procedure to identify the remaining Pareto optimal solutions. Recall that, under the assumption of linear tradeoffs, the Pareto optimal solutions are solutions to the parametric problem. Thus one may use use a parametric network simplex algorithm [23, 24] to solve the parametric shortest path problem discussed in this paper. However, the approach described in this paper has much lower complexity.

Many network design problems that are hard on general graphs admit polynomial time algorithms on directed series-parallel graphs. Ward [28] contains a nice summary of results on polynomial-time dynamic programming algorithms for network flow problems on directed series-parallel graphs.

2. MODELING PRODUCT DESIGN PROBLEMS: AND/OR TREES AND DIRECTED SERIES PARALLEL GRAPHS

In order to develop the basic model of a generic product, we utilize a structure known as an AND/OR tree. This is a special case of more general structures, AND/OR graphs, that have been studied in the computer science literature (see Nilsson [17]). An AND/OR tree is a natural starting point for representing a hierarchical system (one that can be decomposed in a top-down fashion into subsystems, subsubsystems, and so forth) in the presence of alternatives for some/all of the subsystems/atomic elements (elements that cannot be decomposed further). Figure 1 illustrates this concept. Here, the system B contains subsystems C and D (indicated by the "AND-node"). C can be decomposed further in two alternative ways; thus, C contains either subsystem E, *or* subsystem F. E contains atomic units A_1 and A_2, F contains A_3 and A_4, and D contains either A_5 or A_6.

Observe that in Figure 1, each node is either an AND-node, whose selection necessitates the selection of *all* of its child nodes, or an OR-node, whose selection necessitates the selection of *exactly one* of its child nodes—B, E, and F are AND-nodes, while C and D are OR-nodes. Notice that the OR, in the AND/OR tree is an exclusive OR. In the rest of the paper, we use OR to denote an exclusive OR. It might appear that AND and OR nodes are insufficient to model logical conditions such as (A_1 AND A_5) OR A_6. However, it is easy to see that by decomposing this expression into its constituent AND and OR parts, we may represent this on an AND/OR



A: Function Block Representation B: Decomposition of an Assembly Block

Figure 2. Decomposition of a product.

tree by using an AND node, say N_i, to represent (A_1 AND A_5), and then an OR node to represent N_i OR A_6. We refer to this case as the standard form, and henceforth assume that the AND/OR tree is of this form.

In order to use AND/OR trees to model a product, we note that any product is designed to satisfy a certain *function;* this basic function can then be recursively decomposed into subfunctions. These function blocks are, thus, abstract representations of what a product must do in order to accomplish its function. For example, in Figure 2(A) the product must do Function 1 *and* Function 2 to accomplish its function. With a function block representation of a product, designers can postulate alternate function blocks that achieve the same function. For example, in Figure 2(A) Function 1 may be achieved by either providing Function 3 *or* Function 4. The decomposition process continues until the function blocks become "concrete" enough, i.e., until it becomes possible to map a function block on to an assembly/component that can be manufactured/purchased. For example, in Figure 2(A) Function 3 is accomplished using Assembly 1, which in turn is composed of Assembly 4 *and* Assembly 5.

Consider the manufacture of each of the terminal assembly nodes in Figure 2(A). An assembly can be decomposed into its constituent components. Each of these components will have alternatives; moreover, each component will have a set of processes that need to be performed on it, and each of these processes will also have alternatives. Figure 2(B) shows this decomposition of an assembly into its constituent components and processes. The generic component and generic process nodes serve as dummy nodes that cast the AND/OR tree into the standard form.

We note that it is possible for an assembly to satisfy multiple functions and thus occur multiple times on the leaves of the AND/OR tree. We assume that an assembly used on a product may only satisfy a single functionality at a time. In other words, using an assembly that can provide two functionalities requires multiple installations of the assembly on the product. However, if an assembly could simultaneously satisfy multiple functions (such assemblies would typically be few and much more expensive), we can modify the AND/OR tree to model this situation and satisfy our assumption. As an example, suppose a product has two functions F_1 AND F_2. Further, F_1 may be provided by assembly A_1 OR A_2 OR A_3, while F_2 may be provided for by A_3 OR A_4 OR A_5. If using A_3 for F_1 as well as for F_2 requires two installations of A_3, then we leave the AND/OR tree unchanged. If A_3 can simultaneously be used to provide functions F_1 and F_2, and thus only a single installation of A_3 is required, we modify the AND/OR tree as follows. We create a new dummy function F_3 that represents the use of assemblies that can simultaneously provide both functionalities F_1 and F_2. F_3 has a single child

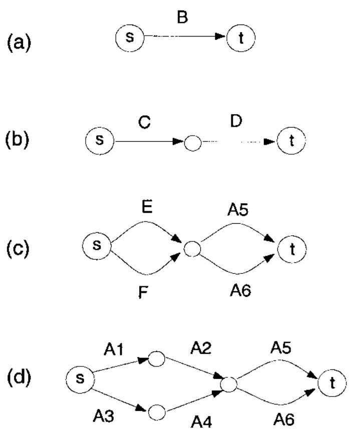


Figure 3. Constructing a Directed Series Parallel Graph from the AND/OR tree in Figure 1. (a) Single arc, (b) a series operation is applied to arc B, replacing it by arcs C and D, (c) parallel operations are applied to arcs C and D, and (d) series operations are applied to arcs E and F.

or assembly, A_3. Our tree then replaces the expression (F_1 AND F_2), by the appropriate creation of AND/OR nodes, by the new expression F_3 OR (F_1 AND F_2). As might be inferred, a systematic way to transform the tree to the required form may easily be obtained by considering assembly blocks that can simultaneously satisfy multiple functionalities.

We now establish a correspondence between AND/OR trees and directed series parallel graphs. A *directed series-parallel graph* is a directed acyclic graph (a graph that contains no directed cycles) that can be constructed solely by series and parallel operations starting from a single arc. A *series operation* replaces a single arc by two or more arcs in a linear chain. A *parallel operation* replaces a single arc by two or more parallel arcs between the two end points of the arc.

Valdes, Tarjan, and Lawler [27] describe a tree data structure, that we refer to as a SERIES/PARALLEL tree, consisting of two types of nodes—SERIES and PARALLEL—to compactly represent directed series-parallel graphs. A directed series-parallel graph is constructed from its SERIES/PARALLEL tree representation as follows. Start the construction by introducing a single arc (s, t) representing the root node (the parent node of the tree, i.e., the node on the tree with no parent) of the SERIES/PARALLEL tree. Next, scan the SERIES/PARALLEL tree in a breadth first search [6] manner: applying a series operation when a SERIES node is encountered, by replacing the arc corresponding to the SERIES node, by arcs in series corresponding to the ordered children of the SERIES node (the order is from the leftmost child to the rightmost child of the SERIES node); and applying a parallel operation when a PARALLEL node is encountered, by replacing the arc corresponding to the PARALLEL node, by arcs in parallel corresponding to the children of the PARALLEL node. By interpreting an AND node as a SERIES node, and an OR node as a PARALLEL node in the SERIES/PARALLEL tree data structure, we obtain a directed series-parallel graph corresponding to an AND/OR tree. Figure 3 illustrates this correspondence.

Observe that by construction the directed series-parallel graph corresponding to the AND/OR tree has a unique node, denoted by s, with no incoming arc that we refer to as the origin; and a unique node with no outgoing arc, denoted by t, that we refer to as the destination. We now show that, by construction, there is a one to one correspondence between the paths from the origin s to the destination t in the directed series-parallel graph and the *feasible choices* of *leaf* *nodes* (or atomic units) in the original AND/OR tree. By a feasible choice, we mean that if the leaf nodes in the choice set are selected, then the conditions represented by the AND/OR tree are satisfied. Consider the following two examples. The *s–t* path $A3$–$A4$–$A5$ corresponds to the selection of atomic units $A3$, $A4$, and $A5$ on the AND/OR tree. $A3$ and $A4$ represent the selection of subsystem F. Notice that since subsystem C is composed of either subsystem E or F, this implies subsystem C is represented. The selection of $A5$ indicates the selection of subsystem D, which together with the selection of subsystem C, represents that system B can be feasibly constructed by the choice of atomic units (leaves) $A3$, $A4$, and $A5$. Similarly $A1$, $A2$, and $A5$ is a feasible choice of leaf nodes on the AND/OR tree represented by the *s–t* path $A1$–$A2$–$A5$.

In what follows let $\mathcal{T}$ denote the AND/OR tree, and $G^{\mathcal{T}}$ denote the directed series-parallel graph constructed from the AND/OR tree. We call the number of nodes on the (unique) path from the root of the tree to a particular node the *depth of the node.* Thus, the root node has depth 1, the children of the root node have depth 2, and so on. We refer to the maximum depth of all nodes in the AND/OR tree as the *depth of the tree.* Additionally, for convenience in the rest of the paper, since we always consider directed series-parallel graphs, we drop the qualifier directed and refer to a directed series-parallel graph as a series-parallel graph.

LEMMA 1: There is a one to one correspondence between *s–t* paths on the series-parallel graph $G^{\mathcal{T}}$ and the set of feasible choices of leaf nodes on the AND/OR tree $\mathcal{T}$.

PROOF: To simplify our proof, without loss of generality, we impose the following structure on the AND/OR tree. Every alternate level of the AND/OR tree, consists solely of OR nodes, or solely of AND nodes. In other words, until we reach the leaves of the tree, children of AND nodes are OR nodes, and children of OR nodes are AND nodes.

The proof of equivalence is by induction on the depth of the AND/OR tree. It is trivially true for a tree of depth 1. Suppose it is true for AND/OR trees of depth k, and consider an AND/OR tree $\mathcal{T}$ of depth $k + 1$. Observe that the AND/OR tree $\mathcal{T}_d$ obtained by deleting the leaves at depth $k + 1$ from $\mathcal{T}$ is a tree of depth k. Consider $G^{\mathcal{T}_d}$ the series-parallel graph constructed from the truncated tree $\mathcal{T}_d$. By the induction assumption the correspondence holds between $\mathcal{T}_d$ and $G^{\mathcal{T}_d}$ the series-parallel graph constructed from it. Now consider the complete tree $\mathcal{T}$. To obtain $G^{\mathcal{T}}$, the series-parallel construction process replaces arcs in $G^{\mathcal{T}_d}$ corresponding to nonleaf nodes of $\mathcal{T}$ with depth k, by arcs corresponding to their children that are leaf nodes with depth $k + 1$.

If the nonleaf nodes at depth k are AND nodes, then an arc corresponding to one of these nodes in $G^{\mathcal{T}_d}$ is replaced by a set of arcs, corresponding to the AND node's children, in a linear chain to obtain $G^{\mathcal{T}}$. Thus any path between nodes s and t in $G^{\mathcal{T}_d}$ that includes an arc corresponding to an AND node of depth k now includes in $G^{\mathcal{T}}$ all of its children. Similarly, any feasible choice in $\mathcal{T}_d$ that contains an AND node of depth k, is now feasible in $\mathcal{T}$ if it includes (instead of the AND node) all of its children. Consequently, if the correspondence holds for AND/OR trees of depth k, it must hold for AND/OR trees of depth $k + 1$, where the nonleaf nodes with depth k are AND nodes.

Now suppose that the nonleaf nodes at depth k are OR nodes. Then the series-parallel construction process replaces the arcs in $G^{\mathcal{T}_d}$ corresponding to nonleaf nodes with depth k by parallel arcs corresponding to their children, that have depth $k + 1$, to obtain $G^{\mathcal{T}}$. Any path from s to t in $G^{\mathcal{T}_d}$ that includes an arc corresponding to an OR node of depth k now includes exactly one of the arcs corresponding to the OR node's children. Similarly, any feasible choice in $\mathcal{T}_d$ that contains an OR node of depth k is now feasible if it includes in $\mathcal{T}$ exactly one of its children. As a result, if the equivalence holds for trees of depth k, it holds for trees of depth $k + 1$, where the nonleaf nodes of depth k are OR nodes. □

As a consequence of Lemma 1, we note that the problem of choosing a feasible set of nodes on an AND/OR tree may be modeled as a problem of finding a path from s to t on the corresponding series-parallel graph. Depending on the costs associated with the choices, and their interactions, the problem may be modeled either as a shortest path problem, or as a more complex network design problem of finding a feasible s-t path with minimum cost.

Before we conclude this section, we make an observation that will prove useful in the development of our solution procedure to the parametric shortest path problem in Section 4. An alternate bottom-up view of the construction process of a series-parallel graph is as follows. The leaves of the AND/OR tree, which we denote by $\mathcal{L}(\mathcal{T})$, represent arcs that are building blocks of the series-parallel graph. Each node i of the AND/OR tree represents a series-parallel graph obtained in a bottom-up construction process. For any $i \in \mathcal{T}$, let $\mathcal{C}(i)$ denote its children on the AND/OR tree, and let $G_i^{\mathcal{T}}$ denote the series-parallel graph it represents. Let $s(G_i)$ denote the origin of series-parallel graph G_i and $t(G_i)$ denote the destination. A parallel composition of two or more series-parallel graphs $G_1, G_2, \ldots, G_k$ is obtained by coalescing the origins $s(G_1)$, $s(G_2), \ldots, s(G_k)$ together and coalescing the destinations $t(G_1), t(G_2), \ldots, t(G_k)$ together. A series composition of two or more series-parallel graphs $G_1, G_2, \ldots, G_k$ is obtained by coalescing in order $t(G_1)$ with $s(G_2)$, $t(G_2)$ with $s(G_3), \ldots, t(G_{k-1})$ with $s(G_k)$.

The bottom-up construction process starts at the leaves $\mathcal{L}(\mathcal{T})$ of the AND/OR tree and works its way up the tree until it reaches the root. At a leaf node $i \in \mathcal{L}(\mathcal{T})$, $G_i^{\mathcal{T}}$ is simply an arc. Elsewhere on the tree if i is an AND node, $G_i^{\mathcal{T}}$ is obtained by a series composition of its children $G_j^{\mathcal{T}}$ for $j \in \mathcal{C}(i)$, and if i is an OR node, $G_i^{\mathcal{T}}$ is obtained by a parallel composition of its children $G_j^{\mathcal{T}}$ for $j \in \mathcal{C}(i)$. Observe that if r denotes the root of $\mathcal{T}$, $G_r^{\mathcal{T}} = G^{\mathcal{T}}$. As an example, Figure 4 shows the series parallel graphs represented by the nodes on the AND/OR tree under this viewpoint.

3. AN APPLICATION: THE T/R MODULE DESIGN PROBLEM

We now review the application [1, 16] of the above framework to the design of transmitter/receiver (T/R) modules—printed circuit board assemblies that are a component of radar systems. The specific metrics that we seek to optimize are a cost metric and a manufacturing yield metric. The basic input to the problem consists of an AND/OR tree description of the T/R module, similar to Figure 2. Given this description, the design problem is to choose among the alternative function blocks, assembly blocks, components, and processes for each component, such that the resulting design is efficient with respect to the cost and manufacturing yield metrics.

In our model, we make a few assumptions that allow us to decompose the product design problem by its constituent subassemblies [the leaves of the AND/OR tree in Fig. 2(A)]. First, for ease of exposition, we assume that the subassemblies are manufactured independently. Second, we do not consider the impact of component commonality between subassemblies. Third, given the first two assumptions, we may now assume that the two metrics, cost and yield, are decomposable by assembly block. That is, the cost/yield contribution of an assembly block is assumed to depend only upon the decisions made within that assembly block. Later in this section we discuss the consequences of relaxing the first assumption.

For any subassembly, Table 1 describes the input data for the problem. Key attributes such as material costs, run times, setup times, process yields, and material defect rates are assumed to be known for components, processes, and component-process combinations.

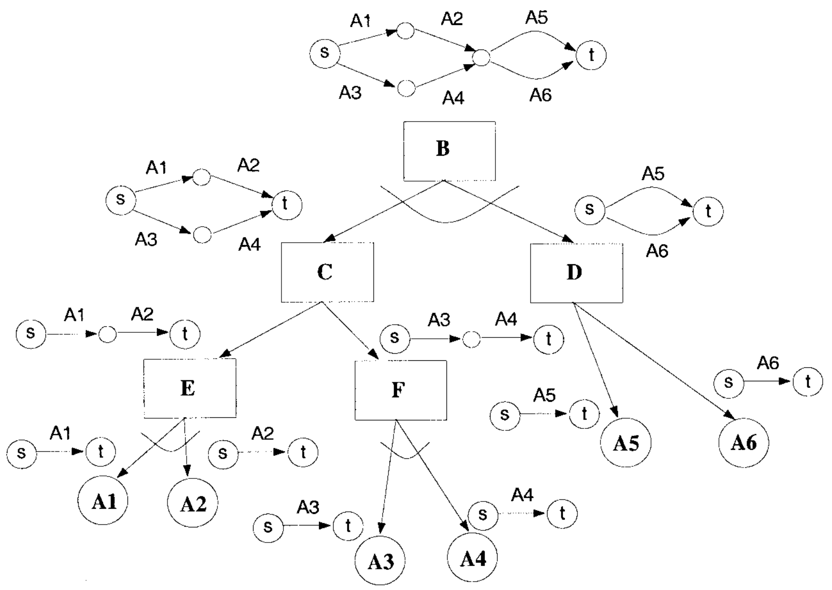


Figure 4. Bottom-up view of AND/OR tree. The series-parallel graph corresponding to each node on the AND/OR tree is located above each node.

We now describe the mathematical formulation of the problem and then show the equivalent network representation. First we define the following decision variables:

$$x_j = \begin{cases} 1 & \text{if component } j \text{ is selected}, \quad j \in \mathcal{V}_k, \quad k \in \mathcal{V}, \\ 0 & \text{otherwise}, \end{cases}$$

$$y_p = \begin{cases} 1 & \text{if process } p \text{ is used in the assembly}, \quad p \in \mathcal{P}, \\ 0 & \text{otherwise}, \end{cases}$$

$$x_{pj} = \begin{cases} 1 & \text{if process } p \text{ is selected for component } j, \\ 0 & \text{otherwise } (j \in \mathcal{V}_k, \quad k \in \mathcal{V}, \quad p \in \mathcal{P}_{ji}, \quad i \in \mathcal{P}_j). \end{cases}$$

Table 1. Notation.

$\mathcal{V}$ =	set of generic components
$\mathcal{V}_j$ =	set of alternatives for the jth generic component
$\mathcal{P}$ =	set of all processes (including alternate processes)
$\mathcal{P}_j$ =	set of set of 'generic' processes related to the jth component
$\mathcal{P}_{ji}$ =	set of alternatives for the ith generic process related to the jth component: $i \in \mathcal{P}_j$
c_j =	unit cost of jth component: $j \in \mathcal{V}_k$, $k \in \mathcal{V}$
t_p =	setup time of pth process: $p \in \mathcal{P}$
t_{pj} =	runtime when pth process is used for jth component: $j \in \mathcal{V}_k$, $k \in \mathcal{V}$, $p \in \mathcal{P}_{ji}$, $i \in \mathcal{P}_j$
α_j =	defect rate of jth component: $j \in \mathcal{V}_k$, $k \in \mathcal{V}$
β_p =	yield rate of the pth process: $p \in \mathcal{P}$
l =	labor cost per unit time
b =	batch size

The expressions for design cost, which we seek to minimize, and manufacturing yield, which we seek to maximize, are as follows:

$$C = \text{unit cost} + \text{runtime cost} + \text{setup cost} = \sum_j c_j x_j + l \sum_{p,j} t_{pj} x_{pj} + \frac{l}{b} \sum_p t_p y_p, \tag{1}$$

$$Y = \prod_p (\beta_p)^{y_p} \prod_j (1 - \alpha_j)^{x_j}. \tag{2}$$

The cost expression (1) is computed as the cost per unit in the batch. Thus, the unit cost and runtime cost do not include the batch size, while the setup cost is spread over the batch. Observe that in the manufacture of an assembly, a process is simultaneously applied to all components requiring that process. Thus, the setup cost for each process is incurred only once per assembly. The yield expression (2) consists of the product of the component defect rates and process yields. We can linearize (2) to get

$$Y' = \log Y = \sum_p y_p \log \beta_p + \sum_j x_j \log(1 - \alpha_j). \tag{3}$$

The problem we wish to solve is the following bicriteria integer program (**P**):

$$\text{minimize} \quad \begin{Bmatrix} C \\ -Y' \end{Bmatrix}$$

$$\text{subject to} \quad \sum_{j \in \mathcal{V}_k} x_j = 1 \quad k \in \mathcal{V}, \tag{4}$$

$$\sum_{p \in \mathcal{P}_{ji}} x_{pj} = x_j \quad \forall j,\ \forall i \in \mathcal{P}_j, \tag{5}$$

$$y_p \geq x_{pj} \quad \forall p, j, \tag{6}$$

$$x_j, y_p, x_{pj} \in \{0, 1\} \quad \forall j, p. \tag{7}$$

Constraints (4) and (5) capture the AND/OR tree structure of the problem. Constraint (4) tells us that we should choose exactly one component among all the components representing generic component k. Similarly constraint (5) tells us that if component j is selected, then, for each generic process that acts on it, exactly one of the alternatives for this generic process should be selected. Constraint (6) tells us that a process may be used on a component (x_{pj}) only if the process has been selected for use in the manufacture of the assembly (y_p). We note that even the single criteria version of this integer program is known to be $\mathcal{NP}$-hard via a reduction from the uncapacitated facility location problem [25].

To obtain Pareto optimal solutions to the bicriteria problem, we solve the parametric problem $\mathbf{P}_\lambda$:

$$\text{minimize} \qquad Z(\lambda) = \lambda C - (1 - \lambda)Y' \tag{8}$$

$$\text{subject to} \qquad \text{constraints (4)–(7),}$$

where the parameter λ ranges over the interval [0, 1]. As discussed earlier, this gives all the Pareto optimal solutions when the decision makers' utility function is linear. Otherwise we have a subset of the Pareto optimal solutions.

We now develop the network representation for problem $\mathbf{P}_\lambda$ that we wish to solve. In order to do so, we first transform the AND/OR tree corresponding to $\mathbf{P}_\lambda$ into an equivalent series parallel graph, using the procedure described in Section 2. This would lead to a graph similar to the one in Figure 3(d). Since the arcs in the series parallel graph correspond to the leaf nodes (i.e., the process nodes) in the AND/OR tree, each arc in the graph represents a specific component-process combination. Thus, the series-parallel graph lists all the possible component-process combinations, and each s–t path in this graph corresponds to a feasible solution to $\mathbf{P}_\lambda$. Each arc will have associated with it an arc weight given by the following expression:

$$W_a(\lambda) = \lambda\left(\frac{c_j}{|\mathcal{P}_j|} + lt_{pj}\right) - (1 - \lambda)\,\frac{\log(1 - \alpha_j)}{|\mathcal{P}_j|}, \tag{9}$$

where j and p are the component and process corresponding to arc a. In other words, the arc weight captures the costs specific to the particular component-process combination. Since component j requires $|\mathcal{P}_j|$ generic processes, and, since for each generic process exactly one specific process alternative is selected (recall that process alternatives are represented by arcs in parallel), we can spread out the cost of component j, c_j, across the component-process arcs corresponding to j in the manner described in Eq. (9); it is necessary to do so in order to avoid incurring the cost c_j multiple times (otherwise we will incur this cost for each component-process arc corresponding to j).

We now turn to the fixed costs associated with each process p, given by $FC(p) = \lambda lt_p/b - (1 - \lambda)\log\, \beta_p$. Since the arcs in the series parallel graph represent component-process combinations, it is clear that each process can be viewed as a set of arcs in the graph. For instance, in Figure 3(d), arcs $A1$ and $A4$ might represent the same process, and so forth. Consequently, $FC(p)$ can be viewed as a *fixed charge* that is incurred (exactly once) should any of the arcs corresponding to process p be selected (irrespective of the number of such arcs selected). The problem of finding efficient solutions to problem **P,** i.e., that of solving $\mathbf{P}_\lambda$ for different values of λ, can thus be viewed as that of finding shortest s–t paths through the corresponding series parallel graph for different values of λ, where the cost of a path includes the fixed charges associated with the processes selected by the path.

3.1. Outline of Solution Algorithm to Parametric Problem $\mathbf{P}_\lambda$

The solution procedure that we propose begins with the observation that the number of processes involved in T/R module design is quite small. Further, selecting a set of processes $\mathcal{P}'$ corresponds to fixing $y_p = 1$ for $p \in \mathcal{P}'$ in the integer program, or may be correspondingly viewed as deleting the arcs corresponding to processes not in $\mathcal{P}'$ from the series-parallel graph. We denote by $\mathbf{P}_\lambda(\mathcal{P}')$ the reduced problem associated with a given set of selected processes $\mathcal{P}'$. Notice that, to solve $\mathbf{P}_\lambda(\mathcal{P}')$, since the set of processes are fixed and all other costs are decomposable by the arcs selected on a path, we need to solve a parametric shortest path problem on the reduced graph.

Our solution procedure, enumerates all process combinations: This is computationally viable since there are a small number of process alternatives. Observe that some process combinations may be infeasible, i.e., there is no path from node s to node t in the series-parallel graph (in practice only a small set of process combinations are feasible). For each feasible process combination $\mathcal{P}'$, it then solves the parametric problem $\mathbf{P}_\lambda(\mathcal{P}')$, corresponding to the selection of processes $\mathcal{P}'$ for the assembly. It then combines these parametric solutions for all process combinations to obtain the overall parametric solution. (Note, as observed earlier, that the solution procedure to the overall product design problem is exponential.)

3.2. Relaxing the Independent Manufacture Assumption

Earlier in this section we assumed that subassemblies are manufactured independently. This was one of the assumptions that allowed us to decompose the product design problem by subassembly. Suppose different subassemblies could be acted upon simultaneously during a single setup of a process. Then, to model the problem, rather than consider each subassembly independently we could consider the AND/OR tree for the entire product. In our model this simply means working on a larger series-parallel graph.

4. POLYNOMIAL TIME ALGORITHM FOR PARAMETRIC SHORTEST PATH PROBLEM

In order to make our solution procedure for bicriteria product design optimization problems work, we need an algorithm to solve the parametric shortest path problem. As we have indicated earlier, the parametric shortest path problem is hard on general graphs. We now describe a polynomial time algorithm for the parametric shortest path algorithm on series-parallel graphs. Our description follows in three parts. First we consider the nonparametric problem (i.e., λ is fixed), and describe an $\mathcal{O}(|A|)$ dynamic programming algorithm. Next, we develop some intuition concerning the parametric analysis, by describing how to combine the parametric analysis of two subproblems. We then modify the dynamic programming algorithm, and solve the parametric problem in $\mathcal{O}(|A|^2)$ time.

4.1. Dynamic Programming Algorithm for Nonparametric Problem

While the shortest path problem is polynomially solvable by Dijkstra, or other alternatives, we describe a linear-time dynamic programming approach that utilizes the AND/OR tree representation. This algorithm is identical to those presented in [2, 28], as applied to the shortest path problem, using the SERIES/PARALLEL tree representation. This approach enables an easy extension to the parametric case.

To simplify the analysis of the running time of the algorithm, we use a *binary AND/OR tree* representation (the running time results hold even if we use the original AND/OR tree representation of the problem). In a binary AND/OR tree each node in the AND/OR tree, except for the leaf nodes, has two children (see Fig. 5). A transformation to a binary AND/OR tree structure may be easily obtained in linear time as follows. Whenever an OR node or an AND node has more than two children, the transformation procedure replaces the poly-ary tree structure by a binary tree structure as shown in Figure 5. The number of operations required to perform this transformation is proportional to the number of nodes in the binary AND/OR tree.

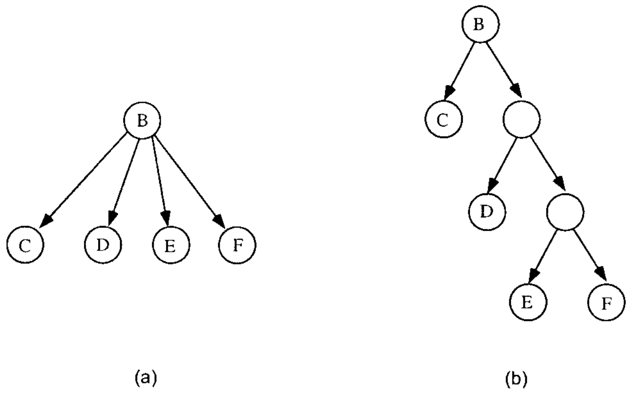


Figure 5. Transformation to a binary AND/OR tree: (a) an OR node with four children and (b) its binary counterpart.

Since the number of leaves in the binary AND/OR tree is identical to the number of leaves in the original AND/OR tree, and a binary tree with $|\mathscr{L}(\mathscr{T})|$ leaves has $2|\mathscr{L}(\mathscr{T})| - 1$ nodes (see [4, 13]), the transformation takes $\mathbb{O}(|\mathscr{L}(\mathscr{T})|)$ time.

For a series-parallel graph G_i, let $R(G_i)$ denote the cost of the shortest path from $s(G_i)$ to $t(G_i)$. For convenience, when obvious, we will drop the superscript $\mathscr{T}$ in the notation $G_i^{\mathscr{T}}$ for $i \in \mathscr{T}$.

The following two simple observations provide the necessary ingredients for the dynamic programming recursion.

OBSERVATION 1: If i is an AND node on $\mathscr{T}$, then the shortest path from $s(G_i)$ to $t(G_i)$ is obtained by taking the union of the shortest paths of the children of i. In particular $R(G_i)$ the cost of the shortest path is obtained as the sum of the costs of the shortest paths of the children of i (i.e., $R(G_i) = \sum_{j \in \mathscr{C}(i)} R(G_j)$).

OBSERVATION 2: If i is an OR node, the shortest path from $s(G_i)$ to $t(G_i)$, is obtained by finding the lowest cost path among the children of i. In particular the minimum cost path is obtained by finding the least cost path among the shortest paths from $s(G_j)$ to $t(G_j)$ for $j \in \mathscr{C}(i)$.

The dynamic programming algorithm starts from the leaves of the AND/OR tree setting $R(G_i)$ equal to the cost of the arc for $i \in \mathscr{L}(\mathscr{T})$ and works its way up to the root using the following equations.

$$R(G_i) = \sum_{j \in \mathscr{C}(i)} R(G_j) \qquad \text{if } i \text{ is an AND node,} \tag{10}$$

$$R(G_j) = \min_{j \in \mathscr{C}(i)} R(G_j) \qquad \text{if } i \text{ is an OR node.} \tag{11}$$

To keep track of the shortest path, we observe that choices among competing alternatives are made solely at OR nodes. Thus to keep track of the shortest path, at OR nodes, we keep track

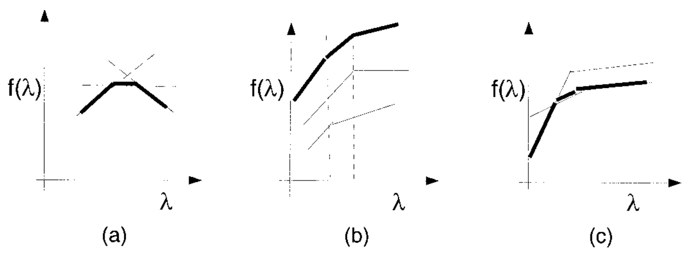


Figure 6. Parametric analysis with linear functions. The (a) minimum of a set of linear functions, (b) sum of two piecewise linear and concave functions, and (c) minimum of two piecewise linear and concave functions are all piecewise linear and concave functions.

of the child (or children, if there is more than one child that gives the minimum) that gives the minimum in Eq. (11).

At each node on the tree the algorithm either finds the minimum of the objective values of the two children or sums the objective values of the two children (since we are using a binary tree representation). Both of these operations take $\mathcal{O}(1)$ time. Thus the total running time, as well as the space required, is bounded by the number of nodes in the tree, and so the algorithm runs in $\mathcal{O}(|A|)$ time, and requires $\mathcal{O}(|A|)$ space (since $|\mathcal{L}(\mathcal{T})| = |A|$).

4.2. Parametric Analysis Preliminaries

In this section we develop some lemmas regarding parametric analysis that are essential to our dynamic programming algorithm. Since some of these results are well developed in the computational geometry literature [19], we provide an informal analysis to motivate the dynamic programming algorithm for the parametric problem.

In the parametric problem, let C_a and Y_a denote the cost and yield terms respectively of arc a in Eq. (9). Thus $W_a(\lambda) = \lambda C_a - (1 - \lambda)Y_a$. Let $\mathcal{Q}$ denote the set of paths from $s(G^{\mathcal{T}})$ to $t(G^{\mathcal{T}})$. For a particular path $Q \in \mathcal{Q}$, let $C(Q) = \Sigma_{a\in Q} C_a$ and let $Y(Q) = \Sigma_{a\in Q} Y_a$. Observe that, for a path $Q \in \mathcal{Q}$ and a fixed λ, its cost is given by $(C(Q) + Y(Q))\lambda - Y(Q)$. The objective function of the parametric problem is obtained by finding the lower envelope of the set of lines $(C(Q) + Y(Q))\lambda - Y(Q)$ (where $\lambda \in [0, 1]$) for all $Q \in \mathcal{Q}$.

LEMMA 2: The *lower envelope* of a (finite) set of straight lines is a piecewise linear and concave function. Furthermore, it contains no more segments (pieces) than the number of lines.

The above observation follows immediately from the definition of a lower envelope, and is illustrated in Figure 6(a). From this lemma we may infer that the objective function corresponding to the bicriteria shortest path problem is a continuous, piecewise linear and concave function (note that concavity implies the function is continuous). Further we may observe, by concavity, that if a path Q is optimal for $\lambda = \lambda_1$ and $\lambda = \lambda_2$, then it is optimal for all $\lambda \in [\lambda_1, \lambda_2]$.

The breakpoints, and slopes and intercepts between breakpoints, completely specify a piecewise linear function. Let $B_i = \{b_i^1, b_i^2, b_i^3, \ldots, b_i^{|B_i|}\}$ denote the ordered breakpoints of a piecewise linear function f_i (i.e., $b_i^1 < b_i^2 < \cdots < b_i^{|B_i|}$). Let m_i^r denote the slope, and d_i^r denote the intercept, between breakpoints b_i^r and b_i^{r+1}. In our analysis, λ varies from 0 to 1, and thus the leftmost breakpoint $\lambda = 0$ and the rightmost breakpoint $\lambda = 1$ are common to all functions. Notice that the number of segments in a piecewise linear function f_i is equal to $|B_i| - 1$.

The following two lemmas provide the necessary ingredients for the parametric analysis at OR and AND nodes in the dynamic programming algorithm.

LEMMA 3: The sum of two piecewise linear and concave functions, say f_i with breakpoints B_i, and f_j with breakpoints B_j, is also a piecewise linear and concave function. Furthermore, the number of breakpoints in the resulting function is at most $|B_i| + |B_j| - 2$, and this summation can be carried out in $\mathcal{O}(|B_i| + |B_j|)$ time.

PROOF: The first part of this lemma follows from observing that a breakpoint of the sum must be a breakpoint of one of the original functions [this is illustrated in Fig. 6(b)]. Thus, the number of breakpoints in the resulting function is at most $|B_i| + |B_j| - 2$ (since the breakpoints $\lambda = 0$ and $\lambda = 1$ are common to both functions). Now, we prove the second part of the lemma.

To obtain the sum f_k of two piecewise linear functions f_i and f_j, we create its ordered set of breakpoints $B_k = B_i \cup B_j$ from the ordered lists of breakpoints B_i and B_j. To determine the slope m_k^t and intercept d_k^t between breakpoints b_k^t and b_k^{t+1}, let b_i^r denote the largest breakpoint in B_i that is less than or equal to b_k^t, and b_j^s denote the largest breakpoint in B_j that is less than or equal to b_k^t. Then $m_k^t = m_i^r + m_j^s$, and $d_k^t = d_i^r + d_j^s$.

Since the lists B_i and B_j are sorted, it takes $\mathcal{O}(|B_i| + |B_j|)$ time to create the ordered list of breakpoints B_k. It then takes $\mathcal{O}(|B_k|)$ time to determine the slopes and intercepts between these points, thus the addition of two piecewise linear function takes $\mathcal{O}(|B_i| + |B_j|)$ time. □

LEMMA 4: The minimum (lower envelope) of two piecewise linear and concave functions, say f_i with breakpoints B_i, and f_j with breakpoints B_j, is also a piecewise linear and concave function. Furthermore, the number of breakpoints in the lower envelope is at most $|B_i| + |B_j| - 1$, and can be determined in $\mathcal{O}(|B_i| + |B_j|)$ time.

PROOF: This lemma might not be so readily apparent. However, it follows when one realizes that a segment in one of the original functions cannot appear at two different locations in the lower envelope, due to the concavity of the original functions. Figure 6(c) illustrates this observation. Since the number of segments in the lower envelope is at most the sum of the number of segments in the two original functions, the total number of breakpoints in the lower envelope is at most $|B_i| + |B_j| - 1$.

We now show how the lower envelope of two piecewise linear and concave functions can be determined in $\mathcal{O}(|B_i| + |B_j|)$ time. The algorithm, referred to as a sweep line algorithm, scans the breakpoints in $B_i \cup B_j$ in increasing order. In doing so, it stores a left sweeppoint (λ_{ls}) and a right sweeppoint (λ_{rs}). Initially, the left sweeppoint is the smallest breakpoint in $B_i \cup B_j$ and the right sweeppoint is the second smallest breakpoint in $B_i \cup B_j$. Let b_i^r denote the largest breakpoint in B_i that is less than or equal to λ_{ls}, and b_j^s denote the largest breakpoint in B_j that is less than or equal to λ_{ls}. Let t denote a counter, with $t = 1$ initially, for the breakpoints B_k of $f_k = \min\{f_i, f_j\}$.

Notice that between the left and right sweeppoints the slope of the piecewise linear functions f_i and f_j are unchanged. The algorithm determines at the left sweeppoint, the smaller of the function values $f_i(\lambda_{ls}) = m_i^r\lambda_{ls} + d_i^r$ and $f_j(\lambda_{ls}) = m_j^s\lambda_{ls} + d_j^s$. For the moment assume that there is no tie. If the slope of the line with the smaller function value has changed at the left sweeppoint (i.e., it is different to the left of λ_{ls}), then the left sweeppoint is added to B_k by setting $b_k^t = \lambda_{ls}$ and adding it to B_k. It also sets m_k^t and d_k^t to the slope and intercept of the line with the smaller function value, and increments t by 1. It then determines the value of λ where the two lines intersect [this is given by $\lambda_{int} = (d_s^j - d_r^i)/(m_r^i - m_s^j)$]. To the left of λ_{int}, the line with the larger slope is lower, and to the right of λ_{int} the line with the smaller slope is lower. If λ_{int} lies strictly between the left sweeppoint and right sweeppoint, then it adds it to the list of breakpoints B_k by setting $b_k^t = \lambda_{int}$. It also sets m_k^t and d_k^t to the slope and intercept of the line with the smaller slope, and increments t by 1. It then makes the right sweeppoint the left sweeppoint, and makes the next point on the combined list $B_i \cup B_j$ the right sweeppoint.

If there is a tie between $f_i(\lambda_{ls})$ and $f_j(\lambda_{ls})$, we observe that λ_{ls} must be a breakpoint in the piecewise linear lower envelope. Further, since both lines intersect at the left sweeppoint, the line with the lower slope must belong to the lower envelope in the interval $[\lambda_{ls}, \lambda_{rs}]$. Consequently, in the case of a tie, the algorithm sets $b_k^t = \lambda_{ls}$ and adds it to B_k. It also sets m_k^t and d_k^t to the slope and intercept of the line with the smaller slope, and increments t by 1. It then makes the right sweeppoint the left sweeppoint, and makes the next point on the combined list the right sweeppoint.

This procedure continues until the left sweeppoint is 1. Observe that the number of additions, comparisons, and updates that the algorithm performs between changes in sweeppoints is bounded by a constant. Since the lists are already sorted it takes $\mathcal{O}(1)$ time to find the next sweeppoint. Therefore, it takes $\mathcal{O}(|B_i| + |B_j|)$ time. □

With these results in hand, we are now ready to develop the dynamic programming algorithm for the parametric shortest path problem.

4.3. Algorithm for Solving the Parametric Shortest Path Problem

The dynamic programming algorithm to solve the parametric problem works using a bottom-up approach on the AND/OR tree. At the leaves of the AND/OR tree the parametric problem is simple. The cost of the shortest path as a parameter of λ is $(C_a + Y_a)\lambda - Y_a$ for $\lambda \in [0, 1]$, and the path is the arc a corresponding to the leaf node.

From our results in the preceding sections, at an OR node, corresponding to a parallel composition, we wish to find the minimum of two piecewise linear and concave functions representing the parametric shortest path for each of the children of the OR node. Lemma 4 shows that this can be done in $\mathcal{O}(|B_i| + |B_j|)$ time, where B_i and B_j are the breakpoints of the two children of the OR node. Similarly, at an AND node we wish to find the sum of two piecewise linear and concave functions representing the parametric shortest path for each of the children of the AND node. From Lemma 3 this can be done in $\mathcal{O}(|B_i| + |B_j|)$ time. The dynamic programming equations are identical to Eqs. (10) and (11), except that $R(G_i)$ now represents the parametric objective function.

We now discuss the running time of the dynamic programming algorithm. Observe that the number of operations at any node $i \in \mathcal{T}$ is equal to $\mathcal{O}(\Sigma_{j\in\mathcal{C}(i)} |B_j|)$. Therefore, the total number of operations at nodes at depth k on the AND/OR tree is

$$\sum_{i\in V^k(\mathcal{T})} \mathcal{O}\left(\sum_{j\in\mathcal{C}(i)} |B_j|\right) = \mathcal{O}\left(\sum_{i\in V^k(\mathcal{T})} \sum_{j\in\mathcal{C}(i)} |B_j|\right)$$

$$= \mathcal{O}\left(\sum_{j\in V^{k+1}(\mathcal{T})} |B_j| + \sum_{i\in\mathcal{L}(\mathcal{T}), i\in V^k(\mathcal{T})} 1\right),$$

where $V^k(\mathcal{T})$ denotes the nodes at depth k on $\mathcal{T}$. Using the recursion between nodes at depth k and depth $k + 1$ shown in the above equation, we find

$$\sum_{i \in V^k(\mathcal{T})} \mathcal{O}\left(\sum_{j \in \mathcal{C}(i)} |B_j| \right) = \mathcal{O}(|\{i : i \in \mathcal{L}(\mathcal{T}),\ i \in V^l(\mathcal{T}) \quad \text{for some } l \geq k\}|).$$

In other words the number of operations at nodes at depth k on the AND/OR tree is bounded by a constant times the number of leaves at depth k or greater, which is itself bounded by $\mathcal{O}(|\mathcal{L}(\mathcal{T})|)$. The depth of the tree is bounded by $|\mathcal{L}(\mathcal{T})|$, since the tree has $2|\mathcal{L}(\mathcal{T})| - 1$ nodes, and each level of the tree, except level 1 which contains the root node, contains at least 2 nodes. This bounds the running time of the algorithm by $\mathcal{O}(|A|^2)$. We can now state the result.

THEOREM 1: The parametric shortest path problem on a series-parallel graph with A arcs can be solved in $\mathcal{O}(|A|^2)$ time.

So far we have discussed how to obtain the parametric objective function. We now discuss how to keep track of the parametric shortest paths. It is important to distinguish whether the product managers are interested in (i) *all* the parametric solutions or (ii) the set of nondegenerate parametric solutions (i.e., a minimal set of parametric solutions that contain an optimal solution for each $\lambda \in [0, 1]$). The distinction is that the latter set does not include any ties. In particular, in the latter set each solution is the unique optimal solution for some value of $\lambda \in [0, 1]$, and collectively contains a set of solutions such that one of them is optimal for any $\lambda \in [0, 1]$. Usually, in parametric analysis, decision makers are interested in the set of nondegenerate parametric solutions.

To ascertain the parametric shortest paths, at each OR node i, in the execution of the algorithm, we keep track of the child (or children in case of ties) that gives the minimum between the breakpoints B_i of f_i. Thus we need $\mathcal{O}(|B_i|)$ space at each node i, or $\mathcal{O}(|A|^2)$ space over the entire tree, to keep track of the paths. To obtain the shortest path for a particular value of λ, we simply traverse down the tree in a top-down (or breadth first search) fashion, following the appropriate child indicated by the OR node for the value of λ to obtain the shortest path. Note that this traversal takes $\mathcal{O}(|A|)$ time since it is bounded by the number of nodes in the AND/OR tree.

To obtain the set of all nondegenerate parametric shortest paths, observe that if a path is optimal for λ, such that $b_r^i < \lambda < b_r^{i+1}$, where r is the root of the tree, then the path is optimal for all $\lambda \in [b_r^i, b_r^{i+1}]$. Thus to enumerate a set of nondegenerate parametric solutions we simply repeat the procedure discussed above for fixed λ repeatedly for $b_r^1 < \lambda < b_r^2$, $b_r^2 < \lambda < b_r^3, \ldots, b_r^{|B_r|-1} < \lambda < b_r^{|B_r|}$. In other words, we repeat the procedure $|B_r| - 1$, or, as the following lemma shows, at most $|A|$ times to find the parametric solutions in $\mathcal{O}(|A|^2)$ time.

LEMMA 5: The value function for the parametric shortest path problem on a series-parallel graph has at most $|A| + 1$ breakpoints.

PROOF: From Lemmas 3 and 4 it follows

$$|B_r| \leq \sum_{j \in \mathcal{C}(r)} |B_j| - 1.$$

Recursively replacing the expression on the right, we find

$$|B_r| \leq \sum_{j \in \mathcal{L}(\mathcal{T})} |B_j| - \sum_{j \in \mathcal{T} \setminus \mathcal{L}(\mathcal{T})} 1 = 2|A| - (|A| - 1) = |A| + 1. \qquad \square$$

Before we conclude this section we make a few additional observations. In general the set of *all* parametric shortest paths (i.e., including ties) on a series-parallel graph may be exponentially sized [10]. Our arguments have shown that a series-parallel graph contains at most $|A|$ nondegenerate parametric shortest paths (on general graphs this set may have as many as $|V|^{\log|V|}$ paths). If we are interested in *all* parametric solutions, then it is possible to modify the dynamic programming algorithm, by keeping track of all ties, and implicitly store all parametric shortest paths with $\mathcal{O}(|A|^2)$ space. Of course, the enumeration of the actual paths may take exponential time (since the set may be exponentially sized).

Eisner and Severance [5] discuss a technique to find all the nondegenerate parametric solutions (their technique cannot be applied when we are interested in all parametric solutions). Their algorithm runs in $\mathcal{O}(BT)$ time, where B is the number of breakpoints in the parametric value function, and T is the running time of algorithm for the nonparametric version of the problem. As a consequence of Lemma 5, it follows that their approach also takes $\mathcal{O}(|A|^2)$ for the parametric optimization problem (using the $\mathcal{O}(|A|)$ algorithm for the nonparametric case). We note, however, that the dynamic programming approach described in this paper *explicitly shows* that the number of breakpoints is bounded by $|A| + 1$.

5. DISCUSSION

This paper has presented a framework for product design that permits the consideration of multiple objectives at the design stage. The model is general enough to accommodate many different application settings, and results in network formulations that, in the bicriteria case, can be efficiently solved via dynamic programming. As part of the solution procedure in this paper, we also presented a polynomial time algorithm for solving the parametric shortest path problem on series parallel graphs.

We now discuss some consequences and extensions of our work. In industrial settings cost sensitivity analysis is quite important. For example, product managers often want to know the impact of the cost of a component, as this could be used in negotiating contracts with suppliers. Cost sensitivity analysis involves varying the objective function coefficient of a single variable in the design problem and observing the change in solution as a function of that parameter. This is easily modeled as a parametric optimization problem, and thus cost sensitivity analysis is easily performed under this framework.

While we have considered the bicriteria case in this paper, the AND/OR tree framework can be used to model multicriteria problems as well. For example, suppose the product manager has d criteria ($g_1(\cdot)$, $g_2(\cdot)$, $\ldots$, $g_d(\cdot)$) under consideration. Then, the problem of finding Pareto optimal solutions to the multicriteria problem may be modeled (with the usual proviso on linear utility functions) as a parametric optimization problem with the objective $\lambda_1 g_1(\cdot) + \lambda_2 g_2(\cdot) + \cdots + \lambda_d g_d(\cdot)$, with the additional constraint $\sum_{i=1}^{i=d} \lambda_i = 1$. Using the approach of partially fixing variables outlined in this paper, one obtains a parametric shortest path problem with multiple parameters (which we refer to as a multiparametric shortest path problem), a significantly more complicated problem. By using some more advanced ideas from the field of computational geometry, specifically, Davenport-Schinzel sequences [22], it is also possible to solve this multiparametric shortest path problem in polynomial time.

Another extension of our research deals with the economic lot sizing problem, one of the most fundamental problems in supply chain management, which can be modeled as a network design problem on a series parallel graph [28]. The dynamic programming procedure described in this paper can be adapted to derive a polynomial time algorithm to solve parametric network design problems on series-parallel graphs [20].

REFERENCES

[1] M. Ball, J. Baras, S. Bashyam, R. Karne, and V. Trichur, On the selection of parts and processes during design of printed circuit board assemblies, INRIA/IEEE Symp Emerging Technol Factory Automation 3 (1995), 241–248.

[2] W.W. Bein, P. Brucker, and A. Tamir, Minimum cost flow algorithms for series-parallel networks, Discrete Appl Math 10 (1985), 117–124.

[3] J. Climaco and E. Martins, A bicriterion shortest path algorithm, Eur J Oper Res 11 (1982), 399–404.

[4] T. Cormen, C. Leiserson, and R. Rivest, Introduction to algorithms, MIT Press, Cambridge, MA, 1990.

[5] M. Eisner and D. Severance, Mathematical techniques for efficient record segmentation in large shared databases, J Assoc Comput Machinery 23 (1976), 619–635.

[6] S. Even, Graph algorithms, Computer Science Press, Rockville, MD, 1979.

[7] S. Finger and J. Dixon, A review of research in mechanical engineering design. Part I—descriptive, prescriptive, and computer-based models of design processes, Res Eng Des 1 (1989), 51–67.

[8] S. Finger and J. Dixon, A review of research in mechanical engineering design. Part II—representations, analysis and design for the life-cycle, Res Eng Des 1 (1989), 121–137.

[9] D. Gusfield, Sensitivity analysis for combinatorial optimization, Technical Report UCB/ERL M80/22, Electronics Research Laboratory, University of California, Berkeley, 1980.

[10] P. Hansen, "Bicriterion path problems," Multiple criteria decision making: Theory and application, G. Fandel and T. Gal (Editors), Springer-Verlag, New York, 1980, pp. 109–127.

[11] R. Hasin and A. Tamir, Efficient algorithms for optimization on series-parallel graphs, SIAM J Algebraic Discrete Methods 7 (1986), 379–389.

[12] M. Henig, The shortest path problem with two objectives, Eur J Oper Res 25 (1985), 281–291.

[13] E. Horowitz and S. Sahni, Fundamentals of data structures, Computer Science Press, Rockville, MD, 1983.

[14] V. Krishnan and K. Ulrich, Product development decisions: A review of the literature, Technical Report, Department of Operations and Information Management, The Wharton School, Philadelphia, PA, 1998.

[15] J. Mote, I. Murthy, and D. Olson, A parametric approach to solving bicriterion shortest path problems, Eur J Oper Res 53 (1991), 81–92.

[16] D. Nau, M. Ball, J. Baras, A. Chowdhury, E. Lin, J. Meyer, R. Rajamani, J. Splain, and V. Trichur, Generating and evaluating designs and plans for microwave modules, AI Eng Des Manuf 14 (2000), 289–304.

[17] N. Nilsson, Problem solving methods in artificial intelligence, McGraw-Hill, New York, 1971.

[18] P. Papalambros, Optimal-design of mechanical-engineering systems, J Mech Des 117 (1995), 55–62.

[19] F.P. Preparata and M.I. Shamos, Computational geometry: An introduction, Springer-Verlag, New York, 1985.

[20] S. Raghavan, Parametric network design on series-parallel graphs, draft, 2001.

[21] M. Schaffers, A polynomial time algorithm for the single source network flow design problem on series-parallel graphs, Technical Report 9062, Center for Operations Research and Econometrics, Louvain-la-Neuve, Belgium, 1990.

[22] M. Sharir and P. Agarwal, Davenport-Schinzel sequences and their geometric applications, Cambridge University Press, Cambridge, 1995.

[23] V. Srinivasan and G. Thompson, An operator theory of parametric programming for the transportation problem—I, Nav Res Logistics Quart 19 (1972), 205–225.

[24] V. Srinivasan and G. Thompson, An operator theory of parametric programming for the transportation problem—II, Nav Res Logistics Quart 19 (1972), 227–252.

[25] V. Trichur, Integer programming models for product design, Ph.D. thesis, The Robert H. Smith School of Business, University of Maryland, College Park, 1999.

[26] V. Trichur and M.O. Ball, A multi-objective integer programming framework for product design, Technical Report 98-60, Institute for Systems Research, University of Maryland, College Park, 1998.

[27] J. Valdes, R.E. Tarjan, and E.L. Lawler, The recognition of series-parallel graphs, SIAM J Comput 11 (1982), 289–313.

[28] J.A. Ward, Minimum-aggregate-concave-cost multicommodity flows in strong series-parallel networks, Math Oper Res 24 (1999), 106–129.