

Optimizing Product Line Designs: Efficient Methods and Comparisons

Alexandre Belloni

The Fuqua School of Business, Duke University, Durham, North Carolina 27708, abn5@duke.edu

Robert Freund, Matthew Selove, Duncan Simester

MIT Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Massachusetts 02142
{rfreund@mit.edu, mselove@mit.edu, simester@mit.edu}

We take advantage of recent advances in optimization methods and computer hardware to identify globally optimal solutions of product line design problems that are too large for complete enumeration. We then use this guarantee of global optimality to benchmark the performance of more practical heuristic methods. We use two sources of data: (1) a conjoint study previously conducted for a real product line design problem, and (2) simulated problems of various sizes. For both data sources, several of the heuristic methods consistently find optimal or near-optimal solutions, including simulated annealing, divide-and-conquer, product-swapping, and genetic algorithms.

Key words: product line design; conjoint; optimization

History: Accepted by Jagmohan S. Raju, marketing; received February 22, 2005. This paper was with the authors 1 year and 1 month for 3 revisions. Published online in *Articles in Advance* July 1, 2008.

1. Introduction

Firms designing a new product line can use conjoint analysis to estimate the utility that customers associate with each level of the product's attributes. These part-worth estimates make it possible to predict the market share and profitability of any particular product line design. Firms then face the problem of finding the design that maximizes predicted earnings or market share (for example). Because this optimization problem is NP-hard (Kohli and Krishnamurti 1989), previous research has developed heuristic methods that attempt to find optimal or near-optimal solutions in a reasonable amount of time. This research has typically evaluated these methods in one of two ways: (1) comparing the *relative* performance of different methods (without knowing how close they come to the global optimum), or (2) evaluating *absolute* performance compared to the global optimum. The second approach requires that the global optimum is known, and so this approach has been limited to problems for which it is computationally practical to enumerate every feasible solution.

In this paper, we take advantage of recent advances in optimization methods and computer hardware to find guaranteed optimal solutions to problems that are far too large for complete enumeration. The largest problem we study has approximately 5×10^{15} feasible solutions. With the computer used in this study we can evaluate about 30,000 product lines per second, and so it would take over 5,000 years

to solve the problem through complete enumeration. In contrast, using discrete optimization methods that combine Lagrangian relaxation with branch-and-bound (hereafter referred to simply as "Lagrangian relaxation") we find a guaranteed optimal solution to this problem after one week of computation time.

The Lagrangian relaxation method itself is not a practical algorithm—most managers would consider it too complicated and too computationally intensive for practical use. However, this method enables us to compute guaranteed optimal solutions, which are then used to benchmark the solutions generated by other methods. Although none of the methods that we test (except for Lagrangian relaxation) guarantee global optimality, several of them consistently reach optimal solutions nonetheless. Even when we introduce simulated measurement error based on the standard errors of our part-worth estimates, the most successful methods still generate solutions within 5% of the true optimal earnings. Thus, for the problems we study, relatively simple methods are capable of computing near-optimal solutions with little computational burden.

This is not the first paper to use Lagrangian relaxation to solve product design problems. In a recent paper, Camm et al. (2006) develop a computationally efficient method that uses Lagrangian relaxation with branch-and-bound to solve the *market share* maximization problem for the design of a single product. In

contrast, we focus on the *profit* maximization problem for the design of k products, where $k = 3, 4$, or 5. As we will discuss in §2.4, because the problems are different, the Lagrangian relaxation approach in the Camm et al. (2006) paper is substantially different from the one in this paper.

2. Description of Methods

We compare nine product line optimization methods, which can be grouped into three broad categories:

1. Methods that operate in attribute space.
2. Methods that operate in product space.
3. Methods that evaluate partially formed products.

In this section, we provide a brief description of these methods and the Lagrangian relaxation method that we use to provide an optimality guarantee.¹

2.1. Methods That Operate in Attribute Space

Methods in this category begin by choosing a random solution (or set of solutions) and measuring the earnings level associated with this initial solution. The methods seek to improve the current solution by changing one or more product attributes and then testing the impact that this change has on earnings.

2.1.1. Coordinate Ascent. The coordinate ascent method was first applied to the product optimization problem by Green et al. (1989). This method begins by choosing a random product line and evaluating the profitability of this solution. The method then cycles through each product feature in a randomly chosen order, testing every possible level of each feature, and accepting feature changes that improve earnings while rejecting those that do not. These iterations continue until they no longer yield an improvement in earnings. The simple “one-opt” version of this algorithm only tests a single feature change at a time. We also implement “two-opt” and “three-opt” versions of the algorithm, which simultaneously test two and three feature changes at a time. The coordinate ascent method is guaranteed to find a locally optimal solution, where the “neighborhood” of locality is defined to include all solutions that differ from the current solution by a single feature (or by two features if using two-opt or by three features if using three-opt). However, the method is not guaranteed to find a global optimum because other solutions that differ by even more features may lead to higher earnings.

¹ This section does not provide an exhaustive list of product design methods. In addition to the methods we test, Dobson and Kalish (1988) develop a heuristic to design a near-optimal product line assuming that prices are continuous, McBride and Zufryden (1988) develop a linear programming method for finding a globally optimal solution to the product line design problems (but they find that their method is computationally prohibitive for profit maximization problems of the size we investigate), and Balakrishnan et al. (2004) develop hybrid methods that combine aspects of the genetic algorithm and beam search methods.

2.1.2. Genetic Algorithm. The biological process of natural selection provided the original inspiration for genetic algorithms. Genetic algorithms have been applied to a wide variety of problems in the operations research literature and were first applied to the optimal product design problem by Balakrishnan and Jacob (1996). Alexouda and Paparrizos (2001), Steiner and Hruschka (2003), and Balakrishnan et al. (2004) have also used genetic algorithms on product line design problems. Genetic algorithms start with a population of random solutions. The “fittest” members of this initial population survive and move on to produce the next generation of solutions. New solutions enter the population through a process of reproduction (in which pairs of product lines “mate” to produce offspring that inherit attributes from each parent) and mutation (in which product lines undergo random changes to individual product features). This process continues until a given stopping condition is reached. Additional implementation details are provided in the online appendix (provided in the e-companion).² Balakrishnan and Jacob (1996) suggest that genetic algorithms might be expected to perform well because they search for solutions from a number of different points in the solution space, increasing the odds of finding good solutions.

2.1.3. Simulated Annealing. Simulated annealing is a popular “algorithm of last resort” for difficult discrete optimization problems (see Aarts et al. 1997). The name of the method is derived from the physical process of annealing, in which a liquid is slowly cooled in a heat bath to form a solid in a low-energy state. As far as we know, this method has not previously been applied to the optimal product line design problem. Simulated annealing starts with a randomly chosen solution and proceeds to test random feature changes to the current solution. However, unlike coordinate ascent, the simulated annealing algorithm sometimes accepts feature changes that *reduce* earnings. The probability of accepting such a negative change depends on the magnitude of the drop in earnings and also decreases over time as the algorithm progresses through a preset “cooling schedule.” See the online appendix for additional implementation details. Because simulated annealing sometimes accepts feature changes that reduce earnings, it has the ability to escape from a locally optimal solution in the hope of finding a better solution. For this reason, the method is expected to outperform one-opt coordinate ascent.

² An electronic companion to this paper is available as part of the online version that can be found at <http://mansci.journal.informs.org/>.

2.2. Methods That Operate in Product Space

Methods in this category search for an optimal solution by changing entire products (rather than just individual attributes). These methods can only be used if it is computationally practical to enumerate across the range of possible products, which represents the number of possible combinations of attributes in a single product.

2.2.1. Greedy Heuristic. This method was first applied to the optimal product line design problem by Green and Krieger (1985) and has also been used (with some modifications) by Dobson and Kalish (1993) and Steiner and Hruschka (2003). The greedy heuristic begins by creating a product line that includes only one product, selected as the single product that maximizes earnings. It then proceeds to add one product at a time to the product line, always choosing the product that maximizes earnings given the set of products that have already been selected. The method stops when the desired number of products has been reached. This method is not guaranteed to find a local optimum because, for example, the first product added may not even be locally optimal given the subsequent products that are added.

2.2.2. Divide-and-Conquer Heuristic. Green and Krieger (1993) suggest applying a “divide-and-conquer” heuristic to the optimal product line design problem. This method divides the product line into groups of attributes and completely enumerates all possible combinations for one group while holding the other groups fixed. In our implementation, we treat each product as its own “group” of attributes. Thus, we start with a random product line and then optimize the choice of the first product, holding all other products constant, and then we move on to the second product, and so on. This process continues until it is impossible to improve earnings by changing any single product. This heuristic is guaranteed to find a locally optimal solution, with the local neighborhood defined to include all solutions that differ from the current solution by a single product. Because this method uses a broader definition of the local neighborhood than one-opt coordinate ascent, it can continue to find better solutions after reaching a point that would be considered a local optimum by the one-opt method.³

³ We also considered an alternative implementation of the divide-and-conquer heuristic, which grouped the product features into three groups. The first group comprised the price feature alone, whereas the other two groups included an equal number of nonprice features. Because this alternative implementation took significantly longer to run and produced less profitable solutions, we did not further explore this implementation.

2.2.3. Product-Swapping Heuristic. This is our name for a method that is similar to what Green and Krieger (1985, p. 8) call the “interchange heuristic.” The product-swapping heuristic begins by choosing a random product line and evaluating the earnings level produced by this solution. It then tests each candidate product that is not part of the current solution to see if there is a product in the current solution whose replacement by the candidate product will increase earnings. If such a swap does improve earnings, then the candidate product is added, and the current product is removed from the current solution. This process continues until it is impossible to improve earnings by swapping in any single product. Like the divide-and-conquer heuristic, the product-swapping heuristic is guaranteed to find a local optimum, with the local neighborhood defined to include all solutions that differ from the current solution by a single product.

2.3. Methods That Evaluate Partially Formed Products

Methods in this third category initially consider only a subset of product features, evaluating “partially formed” products. These methods are designed specifically for problems in which the number of possible product types is so large that their enumeration is computationally prohibitive. For problems of this scale, it is not practical to implement methods that operate in product space, and methods that operate in attribute space would require significantly larger computation times.

2.3.1. Dynamic Programming Heuristic. Kohli and Krishnamurti (1987) developed a dynamic programming (DP) heuristic to solve the optimal product design problem for a single product, which was subsequently extended to handle multiple products by Kohli and Sukumar (1990). This heuristic works by building the product line one attribute at a time. For example, consider a problem in which the first product attribute has seven possible levels and the second attribute has two possible levels. The first stage of the heuristic evaluates each of the seven possible levels of the first attribute in terms of their impact both on consumer utilities and on marginal product profitability. The second stage considers all fourteen possible combinations of attributes one and two. Whenever the number of profiles in which a new attribute level appears exceeds the number of products in the product line, the method eliminates certain combinations from consideration, deciding which attribute combinations to maintain with an algorithm similar to the greedy heuristic. This process continues until the final attribute is reached and the number of attribute combinations that remain equals the number of desired products in the product line. As suggested by Kohli

and Sukumar (1990), we implement the heuristic for a number of different random attribute orderings. For each reported iteration of the DP heuristic, we have retained the best solution from ten runs of the heuristic with random attribute orderings.

2.3.2. Beam Search Heuristic. Beam search methods were originally developed for artificial intelligence search problems in speech and image recognition. Nair et al. (1995) were the first to apply these methods to the optimal product line design problem. The beam search heuristic is similar to the dynamic programming heuristic (described above), but there are two key differences. First, instead of proceeding one attribute at a time, beam search works by simultaneously combining different sets of attributes. Second, instead of simultaneously creating an entire product line, beam search adds one product to the product line at a time, in a manner similar to the greedy heuristic.⁴ As with the dynamic programming heuristic, we randomize the attribute ordering before implementing the beam search. The beam search simultaneously creates fourteen product lines, of which the best is ultimately retained.

2.3.3. Nested Partitions Heuristic. Shi et al. (2001) apply a “nested partitions” heuristic to the product line design problem. This method divides the solution space into various feasible regions, estimates which region appears most promising, and then subdivides the most promising region into smaller regions for further exploration. See Shi and Olafsson (2000) and Shi et al. (2001) for implementation details. The nested partitions method can use other product line design heuristics to help calculate the promising index of a region. We use the one-opt coordinate ascent method, taking the maximum of three iterations (increased to twenty iterations for the simulated problems) to calculate each region’s promising index.⁵ The nested partitions method can also use different rules for deciding how far to backtrack when the surrounding region is more promising than the regions under consideration. We backtrack one level when

this happens. We stop the method when a complete product line has been produced.

2.4. Guaranteed Optimality Methodologies: Enumeration, Branch-and-Bound, and Lagrangian Relaxation

In cases where the number of distinct feasible product lines is not too large, previous literature (for example, Kohli and Sukumar 1990) has used complete enumeration to identify the globally optimal solution and guarantee its optimality. Our method, which combines Lagrangian relaxation with branch-and-bound, allows us to provide an optimality guarantee for problems that are too large for complete enumeration. We provide a brief intuitive description of the method in this section. The online appendix provides a detailed technical description.

Branch-and-bound (see Bertsimas and Tsitsiklis 1997) is a discrete optimization method that attempts to avoid enumerating portions of the feasible set by showing that they cannot possibly contain an optimal solution. For example, imagine we have used a heuristic, such as those previously described in this section, to find a feasible solution that produces earnings of $\$Y$. If we can show that any feasible solution that includes a particular product generates earnings of less than $\$Y$, then we know that this particular product is not part of an optimal solution, and we do not need to explore solutions that include this product.

To find an upper bound on the earnings that can be generated by a given set of solutions, we use Lagrangian relaxation (see Bertsimas and Tsitsiklis 1997, Bertsekas 1999). This method relaxes some of the constraints in a problem to create a new problem that is easier to solve. For example, one of the constraints we relax says that each consumer purchases at most one product. In the relaxed problem, consumers can purchase any number of the available products. For any solution in which a consumer purchases more than one product, the Lagrangian relaxation method subtracts a penalty from the earnings of that solution. Likewise, the method adds a reward to the solution’s earnings when a consumer purchases less than one product. The solution to the relaxed maximization problem then provides an upper bound on the optimal earnings in the original problem. The key to the success of this strategy is finding tight upper bounds to rule out portions of the feasible set as quickly as possible. The method searches for the tightest possible upper bounds by varying the penalties that are applied to the objective function when a solution violates the relaxed constraints.

Lagrangian relaxation with branch-and-bound has been used to solve many different discrete optimization problems, but most complicated problems require

⁴The techniques also use slightly different criteria for deciding which attribute combinations to retain at each stage. Kohli and Sukumar (1990) describe their criterion on page 1470 of their paper. Nair et al. (1995) describe their criterion on page 774 of their paper. We have found that the beam search finds better solutions when using the criterion proposed by Kohli and Sukumar (1990), which we implement for both methods.

⁵Shi et al. (2001) use a genetic algorithm, but for our problems, repeatedly using a genetic algorithm to calculate the promising index would substantially increase the running time of the method; and because the nested partitions method designs one product at a time, it would still be unlikely to produce better designs than the (very fast) greedy algorithm. Using genetic algorithms in the nested partitions method is more appropriate when it is not possible to enumerate all of the products.

extensive customization and fine-tuning of these methods. Camm et al. (2006) develop an approach that uses these methods to solve the problem of designing a single product when the objective is to maximize market share. The problem that we consider poses significant new challenges for two reasons: (i) we consider multiple products, and (ii) we solve for the profit maximization problem (which includes the market-share problem as a special case). Therefore, our approach uses a different initial problem formulation, a different Lagrangian relaxation construction, and a different branch-and-bound tree construction. We also take advantage of recently developed techniques for applying Lagrangian relaxation to problems with a very large number of constraints. See Lucena (1992, 1993), Belloni and Lucena (2004), and Belloni and Sagastizábal (2008) for a discussion of these techniques.

3. Results Using Real Conjoint Data

The first problem that we use to compare the different methods is an actual product line design problem faced by Timbuk2, a manufacturer of messenger bags. As part of their product development process, the company worked with a group of academic researchers to conduct a conjoint study that focused on price and nine binary product features. Additional details are reported in Toubia et al. (2003).

We assume that the company wants to design a product line with five products and would like to maximize its predicted earnings. For each respondent we estimated individual part-worths for each feature using ordinary least squares (OLS). We use a “first choice” demand model that assumes each customer purchases the product that provides the greatest utility. We consider seven different price levels (\$70, \$75, \$80, \$85, \$90, \$95, \$100). Because the conjoint data only include two price levels (\$70 and \$100) we interpolate to derive part-worths for the intermediate levels. Timbuk2 provided estimates of the cost of each feature.

Table 1 shows the average part-worth and incremental marginal cost for each feature. On average,

Table 1 Summary Statistics

Feature	Average part-worth	Incremental marginal cost (\$)
\$5 Price increase	-7.6	-5.00
Large size	17.9	3.50
Red color (not black)	-36.0	0.00
School logo	9.0	2.00
Handle	37.7	3.50
Gadget holder	5.2	3.00
Cell phone holder	5.5	3.00
Mesh pocket	9.7	2.00
Velcro flap	18.2	3.50
Reinforcing boot	24.4	4.50

respondents dislike price increases, prefer large bags to small ones, and dislike the color red (meaning they prefer black). Of seven other optional features, the most popular are a handle (in addition to a shoulder strap), a velcro flap for covering a laptop computer, and a reinforcing boot that protects the contents of the bag.

To derive status quo utilities for each consumer, we include a selection of three competing products. These competing bags are arbitrarily designed to include a product with all nine of the optional features priced at \$100, a product with five of the features priced at \$85, and a product with no optional features priced at \$70. We assume that if the competing products offer the same utility as one of the five Timbuk2 products then the customer purchases the competing product. The features included in the competing products and in the optimal Timbuk2 product line are listed in Table A1 in the online appendix. We also assume that the competitor cannot respond. We discuss this last issue further in our review of the limitations and opportunities for future research (see §5).

The combination of nine features plus price yields a total of 3,584 unique products ($2^9 \times 7$). There are over 4.9×10^{15} different combinations of five-bag product lines that could be chosen from this set. Note that, although the number of feasible *product lines* is very large, the range of *individual products* is relatively small. This has some important implications for our evaluation of the optimization methods. First, as explained in the previous section, several of the methods we test can only be implemented on problems in which one can quickly enumerate all possible products, which is the case with our problem. In addition, we take advantage of the small number of possible products to speed up the performance of the methods by precomputing (and storing) the matrix of customer utilities for each product ($3,584 \times 324$ elements). We also precompute the profit margins that the firm earns from each product (3,584 elements). Precomputing these values significantly reduces the computations that the methods must perform for subsequent product line evaluations. The computational and memory requirements of these precomputing tasks are trivial, but for other instances of the product line design problem these tasks might be costly or prohibitive because the number of possible product types increases exponentially with the number of attributes in each product.

Table 2 presents results for ten trials of each optimization method. For each method, the table reports the average earnings, the best earnings reached over ten trials, computational intensity, and the difficulty of implementing the different methods. The CPU time was measured while running the methods in Matlab on an IBM Thinkpad laptop with a 1.7-GHz Pentium

Table 2 Initial Comparison of Methods on the Real Conjoint Data Set

	Average earnings	Average as % of optimal	Best solution reached over 10 trials	Best as % of optimal	CPU time per trial	Subjective difficulty
Lagrangian relaxation	12,226	100.0	NA*	NA*	1 week	Very high
Coordinate ascent (one-opt)	11,217	91.7	12,021	98.3	0.2 sec.	Low
Coordinate ascent (two-opt)	11,977	98.0	12,056	98.6	5.4 sec.	Low
Coordinate ascent (three-opt)	11,999	98.1	12,056	98.6	302.8 sec.	Low
Genetic algorithm	12,105	99.0	12,226	100.0	16.5 sec.	Medium
Simulated annealing	12,226	100.0	12,226	100.0	128.7 sec.	Medium
Divide-and-conquer	12,176	99.6	12,226	100.0	12.5 sec.	Low
Greedy heuristic	12,035	98.4	NA*	NA*	3.5 sec.	Low
Product-swapping	12,218	99.9	12,219	99.9	14.1 sec.	Low
DP heuristic	11,539	94.4	11,904	97.4	5.5 sec.	High
Beam search	11,486	93.9	12,053	98.6	1.9 sec.	High
Nested partitions	11,818	96.7	12,035	98.4	8.4 sec.	High

*These methods were only run once because they always produce the same solution.

processor and 512 MB of RAM. To describe the difficulty of implementing the methods, we report our subjective assessment of relative difficulty. In general, the methods we label as having a “medium” or “high” level of difficulty require some problem-specific fine-tuning of parameter values, whereas those we label as “low” do not.

The earnings level of \$12,226 determined by the Lagrangian relaxation method is the globally optimal earnings of the problem. Among the more practical methods, the genetic algorithm, simulated annealing, divide-and-conquer, and product-swapping perform best, reaching solutions that are, on average, within 1% of the optimum, and the greedy heuristic produces earnings that are 1.6% less than the optimal outcome. The three methods that evaluate partially formed products (beam search, the DP heuristic, and nested partitions) find average solutions that range from 3% to 6% below the optimum. One-opt coordinate ascent has the lowest average earnings, 8.3% less than the optimum. Moving from one-opt to two-opt leads to a large jump in accuracy (from 91.7% to 98.0%), while increasing CPU time from 0.2 seconds to 5.4 seconds. On the other hand, moving from two-opt to three-opt leads to a negligible gain in accuracy (from 98.0% to 98.1%) but a large increase in computation time, from 5.4 seconds to 302.8 seconds. Although some of the differences in average earnings might appear small, it is important to remember that these numbers do not account for fixed costs. If profit margins net of fixed costs are low, then small differences in gross earnings could imply substantial differences in net earnings. For example, if fixed costs are \$10,000, then net earnings when using the coordinate ascent are on average 45% lower than the optimal earnings.

There have been previous comparisons of selected pairs of these methods. The findings reported in Table 2 are consistent with these comparisons. For example, Balakrishnan and Jacob (1996) present results

comparing genetic algorithms and the DP heuristic. Their findings also favor the genetic algorithm. Similarly, Alexouda and Paparrizos (2001) find that genetic algorithms outperform beam search, whereas Steiner and Hruschka (2003) report that genetic algorithms outperform the greedy heuristic.

We also investigate two random strategies for choosing products. When we randomly choose five products from the set of 3,584 possible products, the average earnings level reached in 100 random solutions is only \$5,541 (45.3% of the optimum). In the second random strategy we first choose three products to undercut the competitors and then randomly select the other two products. To undercut the competitors we either match the features and undercut the price or match the price and add an additional feature.⁶ One hundred repetitions of this strategy yields average earnings of \$9,659 (79.0% of the optimum).

Finally, to demonstrate the sensitivity of earnings to changes in feature levels, we measure the average earnings produced by solutions that differ from the global optimum by only a single feature in a single product. There are $5 \times 9 = 45$ possible nonprice deviations and $5 \times 6 = 30$ possible price deviations. Nonprice deviations reduce earnings by an average of \$648 (5% of the optimum), with a range from \$33 (~0%) to \$1,815 (15%). Price deviations reduce earnings by an average of \$1,269 (10%), with a range from \$18 (~0%) to \$4,250 (35%).

3.1. Robustness Testing

Although none of the methods were explicitly designed to account for measurement error, measurement error is unavoidable in practice. To test the

⁶ One of the competing bags had no optional features and was priced at the minimum price (\$70). We cannot undercut this price. Instead, we charged the same price and added a “handle” (which had the highest average part-worth). This strategy does not lead to 100% market share because some people dislike the optional “handle” feature.

Table 3 Performance Under Measurement Error

	Average earnings	Percentage of optimal (%)	Features in common* (%)	Products in common (%)
Coordinate ascent (one-opt)	10,879	89	77	15
Coordinate ascent (two-opt)	11,449	94	80	25
Genetic algorithm	11,615	95	82	30
Simulated annealing	11,786	96	86	40
Divide-and-conquer	11,763	96	86	40
Greedy heuristic	11,659	95	85	40
Product-swapping	11,767	96	86	41
DP heuristic	11,376	93	84	39
Beam search	10,824	89	80	26
Nested partitions	11,385	93	80	28

*Percentage of features for which the solutions produced under measurement error have the same values as the true optimal solution, maximized across the 5! ways of matching the products in the two product lines.

robustness of the methods in the presence of measurement error, we repeated our analysis after perturbing the original part-worth estimates. These perturbations were accomplished by adding (simulated) error to the part-worths: $u'_{i,j} = u_{i,j} + \varepsilon_{i,j}$, where $u_{i,j}$ is the original part-worth for respondent i on product feature j , $\varepsilon_{i,j}$ is a zero-mean, independent (but not identical across customers or attribute levels) normal error term, and $u'_{i,j}$ is the perturbed part-worth. The standard deviation of the perturbations was obtained by using the standard errors for the respective $u_{i,j}$ terms from the OLS estimations. These standard errors averaged approximately 55% of the part-worth absolute values.

We repeated the perturbation process 100 times, obtaining 100 sets of perturbed part-worths for each respondent (in addition to the original part-worths). We treat the perturbation terms as measurement error. Under this interpretation, there is only one set of “true” part-worths: the original part-worths that we analyzed in Table 2. However, we assume that the researcher can only observe the part-worths that are subject to measurement error.

The findings are reported in Table 3. The correlation between performance on the original data set (Table 2) and this robustness check (Table 3) is very high ($\rho = 0.93$). Most methods now perform 2%–5% worse than they do when we assume the data set is 100% accurate. This degradation reflects the loss of information due to measurement error. On average the solutions produced under measurement error have between 77% (for coordinate ascent) and 86% (for product-swapping, divide-and-conquer, and simulated annealing) of feature levels in common with the true optimum. We also report the percentage of complete products in common. Several methods have 40% (two of five) products in common with the optimum. This is because they usually select the two optimal products that match the nonprice features and undercut the price of competing products.

Table 4 Comparison of Methods on Simulated Data

	Average performance (%)	Finds optimal solution (%)	Finds solution >95% of optimal (%)	Average CPU time (sec.)
Lagrangian relaxation	100.0	100.0	100.0	659.4
Coordinate ascent (one-opt)	94.4	8.3	51.7	<0.1
Coordinate ascent (two-opt)	96.0	15.8	65.8	0.6
Coordinate ascent (three-opt)	95.7	13.3	67.5	22.2
Genetic algorithm	99.9	81.7	100.0	11.8
Simulated annealing	100.0	100.0	100.0	131.8
Divide-and-conquer	98.7	45.8	97.5	0.7
Greedy heuristic	97.5	23.3	82.5	0.2
Product-swapping	98.5	39.2	95.8	0.8
DP heuristic	96.3	10.0	70.8	0.9
Beam search	99.1	46.7	99.2	0.4
Nested partitions	93.9	4.2	44.2	2.2

Notes. Average performance is shown as a percentage of the optimal solution. The next two columns show the percentage of trials for which each method finds the optimal solution and a solution greater than 95% of the optimum, respectively.

4. Results Using Simulated Data

To help evaluate the generalizability of the results in §3, we also test the performance of the methods using simulated data. Because our goal is to measure the performance of heuristics relative to the true optimum (as determined by Lagrangian relaxation), we limit our simulations to problem sizes for which the Lagrangian method runs in a reasonable amount of time. Although this allows us to study substantially larger problems than could be solved using complete enumeration, we cannot evaluate some of the extremely large problems tested in previous papers (Shi et al. 2001, Balakrishnan et al. 2004).

Following Kohli and Sukumar (1990), we assume individual part-worths and seller profits for each attribute level obey iid uniform $[0, 1]$ distributions. For each simulation, we randomly select three status quo products. We assume that these products are offered by competing firms and that each consumer chooses the product that offers the highest utility. In our simulated problems, the focal firm designs a product line consisting of 3 or 4 products, each composed of 3, 5, or 7 attributes that can take on 2, 3, 5, or 8 different levels. We select 12 different problem sizes and generate 10 problem instances for each size, for a total of 120 simulated data sets.⁷

Table 4 provides a summary of the results averaged across the different problem sizes (detailed results for each problem are provided in Table A2 in the online appendix). There is a moderately strong correlation ($\rho = 0.61$) between the average performance

⁷ We choose problem sizes for which Lagrangian relaxation finds an optimal solution relatively quickly.

of the methods on the simulated data and the results from the real data set (Table 2). Whereas simulated annealing and the genetic algorithm perform at least as well on the simulated data as on real data set, the divide-and-conquer method (98.7% now versus 99.6% previously) and the product-swapping heuristic (98.5% now versus 99.9% previously) now produce solutions that are slightly further from the optimum. We conjecture that this is because the optimal solution in the previous section included two products that were identical to the competing products but priced slightly lower. Methods searching in product space could easily identify these two products. In the simulated problems, all attributes are “fit” attributes, so that different customers prefer different levels of these attributes. This means that a strategy of undercutting competitors does not apply.

Although simulated annealing finds the optimal solution for all 120 data sets, it also has a running time that is one or two orders of magnitude larger than other methods. The strong performance of this method might result simply from more extensive search, rather than from greater efficiency in the way search is performed. To test this, we ran 500 iterations of the one-opt coordinate ascent method for each data set. This takes an average of about 10 seconds per data set (still an order of magnitude faster than simulated annealing). For 118 of the 120 data sets, the best one-opt solution is the true optimum. Thus, from the perspective of a computation time versus performance trade-off, it could be argued that simulated annealing is less efficient than one-opt coordinate ascent.

5. Conclusions

We have compared a broad range of optimal product line design methods on both real and simulated data. The comparisons take advantage of recent advances that make it possible to compute guaranteed optimal solutions to problems that are too large for complete enumeration. Four of the heuristic methods—simulated annealing, product-swapping, divide-and-conquer, and genetic algorithms—perform particularly well, consistently reaching near-optimal solutions even in the presence of measurement error.

Our results suggest that the product line design problem may share similar characteristics to other problems that are easier to solve (to near optimality) in practice than indicated by computational theory. For example, the traveling salesman problem (see Gutin and Punnen 2002) and the knapsack problem (see Kellerer et al. 2004) are both theoretically hard combinatorial problems. Nevertheless, relatively simple heuristics find near-optimal solutions to large versions of these problems.

The research presented in this paper is subject to several limitations. The first is that our measure of accuracy presumes that the part-worths accurately describe actual customer behavior. Although we demonstrate that the results are robust to unbiased errors in the part-worth estimates, they may not survive systematic biases. If these biases can be predicted, it may be possible to develop a model that does describe actual behavior. For example, Kivetz et al. (2004) find that modifying conjoint analysis to incorporate a compromise effect significantly improves choice predictions.

Another concern is that none of the optimization methodologies consider the competitive response to new product introductions. The optimal solutions all predict large market shares for the focal firm, which could reasonably be expected to provoke an aggressive competitive reaction (particularly in the price attribute). Developing an optimization methodology that adequately accounts for competitive reaction and is tractable for large-scale design problems remains an important challenge for future research.

Finally, although we have been able to compute a guaranteed optimal solution to a product line design problem that is far too large to be solved with complete enumeration, in practice, firms sometimes face even larger problems. For now, we do not know how well the simple methods we study in this paper would perform on these larger problems. In time, as both the algorithms and computational power improve, we anticipate that the frontier of problems that can be solved to guaranteed optimality will continue to expand.

6. Electronic Companion

An electronic companion to this paper is available as part of the online version that can be found at <http://mansci.journal.informs.org/>.

Acknowledgments

The authors thank Karen Freeman for research assistance on this project; Ely Dahan, John Hauser, and Olivier Toubia for the data used in the study; and Andreas Schulz for insightful comments and for suggesting connections to the stable set problem in discrete optimization. This research has been partially supported through the Singapore-MIT Alliance.

References

- Aarts, E., J. Korst, P. van Laarhoven. 1997. Simulated annealing. E. Aarts, J. K. Lenstra, eds. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, UK, 91–120.
- Alexouda, G., K. Paparrizos. 2001. A genetic algorithm approach to the product line design problem using the seller's return criterion: An extensive comparative computational study. *Eur. J. Oper. Res.* **134**(1) 165–178.
- Balakrishnan, P. V., V. S. Jacob. 1996. Genetic algorithms for product design. *Management Sci.* **42**(8) 1105–1117.

- Balakrishnan, P. V., R. Gupta, V. Jacob. 2004. Development of hybrid genetic algorithms for product line designs. *IEEE Trans. Systems, Man, Cybernetics* **34**(1) 468–483.
- Belloni, A., A. Lucena. 2004. Lagrangian heuristics for the linear ordering problem. M. G. C. Resende, J. P. de Sousa, eds. *Metaheuristics: Computer Decision-Making*. Kluwer Academic Publishers, Norwell, MA, 37–64.
- Belloni, A., C. Sagastizábal. 2008. Dynamic bundle methods. *Math. Programming*. Forthcoming.
- Bertsekas, D. P. 1999. *Nonlinear Programming*. Athena Scientific, Belmont, MA.
- Bertsimas, D., J. Tsitsiklis. 1997. *Introduction to Linear Optimization*. Athena Scientific, Belmont, MA.
- Camm, J. D., J. J. Cochran, D. J. Curry, S. Kannan. 2006. Conjoint optimization: An exact branch-and-bound algorithm for the share-of-choice problem. *Management Sci.* **52**(3) 435–447.
- Dobson, G., S. Kalish. 1988. Positioning and pricing a product line. *Marketing Sci.* **7**(2) 107–125.
- Dobson, G., S. Kalish. 1993. Heuristics for pricing and positioning a product line using conjoint and cost data. *Management Sci.* **39**(2) 160–175.
- Green, P. E., A. M. Krieger. 1985. Models and heuristics for product line selection. *Marketing Sci.* **4**(1) 1–19.
- Green, P. E., A. M. Krieger. 1993. Conjoint analysis with product-positioning applications. J. Eliashberg, G. L. Lilien, eds. *Handbooks in Operations Research & Management Science*, Vol. 5. Elsevier Science B.V., Amsterdam, 467–515.
- Green, P. E., A. M. Krieger, R. N. Zelnio. 1989. A componential segmentation model with optimal design features. *Decision Sci.* **20**(2) 221–238.
- Gutin, G., A. Punnen. 2002. *The Traveling Salesman Problem and Its Variation*. Kluwer Academic Publishers, Dordrecht, The Netherlands.
- Kellerer, H., U. Pferschy, D. Pisinger. 2004. *Knapsack Problems*. Springer, Berlin.
- Kivetz, R., O. Netzer, V. Srinivasan. 2004. Alternative models for capturing the compromise effect. *J. Marketing Res.* **41**(3) 237–257.
- Kohli, R., R. Krishnamurti. 1987. A heuristic approach to product design. *Management Sci.* **33**(12) 1523–1533.
- Kohli, R., R. Krishnamurti. 1989. Optimal product design using conjoint analysis: Computational complexity and algorithms. *Eur. J. Oper. Res.* **40** 186–195.
- Kohli, R., R. Sukumar. 1990. Heuristics for product-line design using conjoint analysis. *Management Sci.* **36**(12) 1464–1478.
- Lucena, A. 1992. Steiner problem in graphs: Lagrangian relaxation and cutting-planes. *COAL Bull.* **21** 2–8.
- Lucena, A. 1993. Tight bounds for the Steiner problem in graphs. *Proc. NETFLOW93*, 147–154.
- McBride, R. D., F. S. Zufryden. 1988. An integer programming approach to the optimal product line selection problem. *Marketing Sci.* **7**(2) 126–140.
- Nair, S. K., L. S. Thakur, K. Wen. 1995. Near optimal solutions for product line design and selection: Beam search heuristics. *Management Sci.* **41**(5) 767–785.
- Shi, L., S. Olafsson. 2000. Nested partitions method for global optimization. *Oper. Res.* **48**(3) 390–407.
- Shi, L., S. Olafsson, Q. Chen. 2001. An optimization framework for product design. *Management Sci.* **47**(12) 1681–1692.
- Steiner, W., H. Hruschka. 2003. Genetic algorithms for product design: How well do they really work? *Internat. J. Market Res.* **45**(2) 229–240.
- Toubia, O., D. I. Simester, J. R. Hauser, E. Dahan. 2003. Fast polyhedral adaptive conjoint estimation. *Marketing Sci.* **22**(3) 273–303.