MANAGEMENT SCIENCE
DOI 10.1287/mnsc.1080.0864ec
pp. ec1–ec10

e-companion

ONLY AVAILABLE IN ELECTRONIC FORM

Electronic Companion—"Optimizing Product Line Designs: Efficient Methods and Comparisons" by Alexandre Belloni, Robert Freund, Matthew Selove, and Duncan Simester, *Management Science*, DOI 10.1287/mnsc.1080.0864.

I. Implementation Details of Algorithms

Genetic Algorithms

Our genetic algorithm implementation starts with a population of 500 randomly chosen product lines. The earnings level associated with each of these initial solutions is tested, and the 250 with the highest earnings levels are chosen to move on and create the next generation of solutions. Of these 250 survivors, 125 pairs are chosen at random (with replacement) to "mate" and produce a total of 250 offspring. The literature suggests a number of different genetic crossover methods to create these offspring. We implement the "single-point crossover," meaning that all attributes beyond a randomly chosen crossover point are swapped. For example, a child might inherit the first product and the first five features of the second product from the "father," and inherit the rest of the features of the second product and all other products from the "mother." We find that this crossover method performs slightly better than uniform crossover, in which each attribute of each offspring has a fifty percent chance of taking on the value of each parent. After the process of reproduction takes place, each of the 500 current solutions (250 original solutions plus 250 offspring) is copied to produce 500 new solutions. These new solutions then undergo a process of mutation, in which each feature of each solution is changed with a random probability (we used a mutation rate of 5%). Finally, of the 1,000 total solutions now under consideration, the 500 with the highest earnings level move on to form the next generation. This process continues until a defined stopping condition is met, which in our implementation occurs when the entire population is homogeneous, that is, all 500 product lines are the same. For the simulated datasets, we increase the population size to 2,000 product lines, which improves performance without an excessive increase in running time for these datasets. On the real dataset, such an increase in population size substantially increases computation time with relatively little improvement in performance, so we use the smaller population size.

Simulated Annealing

Our implementation of simulated annealing was done following the guidelines in Aarts, Korst, and van Laarhoven (1997). Like the coordinate ascent algorithm, simulated annealing tests random feature changes and accepts all changes that increase earnings. The key feature of simulated annealing is that it also accepts some changes that reduce earnings. If a feature change reduces earnings, then the probability of accepting the change is given by the following exponential function:

$$P(AcceptChange) = \exp\left(\frac{NewEarnings - OldEarnings}{C}\right)$$

where C is a control parameter that is analogous to temperature in the physical annealing process. Note that the greater the drop in earnings, the lower the probability that the feature change will be accepted. Also, as the control parameter C (the "temperature") drops, the probability of a given change being accepted drops as well.

The term "cooling schedule" refers to the choice of a set of values of the control parameter *C* and the number of feature changes to test for each value. Although Aarts, Korst, and van Laarhoven (1997) offer some general guidelines, setting the cooling schedule is partly a matter of trial-and-error and problem-specific experience. We found that it was best to start with a value large enough that almost all feature changes are accepted, calculate each subsequent value by multiplying the current value by 0.8 (for the simulated problems, we use 0.9), and end with a value small enough that very few negative changes are accepted. We divided the problem into 29 time stages, and tested 10,000 feature changes in each stage, yielding a total of 290,000 tested feature changes. As the method progressed through the cooling schedule, we kept track of the best solution ever reached.

For the original problem, the control parameter was set at each stage according to the following schedule: 1443, 1154, 923, 739, 591, 473, 378, 303, 242, 194, 155, 124, 99, 79, 63, 51, 41, 32, 26, 21, 17, 13, 11, 9, 7, 5, 4, 3, 0.1.

For the simulated problems we use: 21.0, 18.9, 17.0, 15.3, 13.8, 12.4, 11.2, 10.0, 9.0, 8.1, 7.3, 6.6, 5.9, 5.3, 4.8, 4.3, 3.9, 3.5, 3.2, 2.8, 2.6, 2.3, 2.1, 1.9, 1.7, 1.5, 1.4, 1.2, 1.1, 0.1.

II. Description of the Lagrangian relaxation/branch-and-bound method

Preliminaries

We present a detailed description of our Lagrangian relaxation/branch and-bound method for the product line design problem. This method is used to generate guaranteed optimal solutions for problems where complete enumeration is computationally intractable. Lagrangian relaxation is generically designed to compute upper bounds on the (unknown) optimal objective function value (see for example Fisher 1981), thus providing a conservative estimate of how good any feasible solution really is. It is important that these upper bounds are, by construction, relatively efficient to compute. Unfortunately, such efficiency comes at a price, since there is no guarantee that the upper bound will match the true optimal value. The difference between the optimal objective function value and the lowest upper bound obtained by Lagrangian relaxation is the "duality gap" in mathematical programming parlance.

The branch-and-bound method (see for example Bertsimas and Tsitsiklis 1997), can be used to solve a given problem to optimality. It uses a "divide and conquer" approach to explore the set of feasible solutions, which in our case is the set of all product lines. In order to avoid exhaustive enumeration of all feasible solutions, branch-and-bound uses upper and lower bounds on the optimal objective value to restrict the search to promising regions only. In order to enhance the computational efficiency of branch-and-bound, we use upper bounds derived from Lagrangian relaxation to restrict the search, and we also use heuristics that generate feasible product lines to provide lower bounds on the optimal solution. This approach (using Lagrangian relaxation to produce bounds at relevant nodes

of the branch-and-bound tree) was also used in Camm et al. (2006), although they solved a significantly different problem and hence used a different Lagrangian relaxation construction as well as a different branch-and-bound tree construction. We emphasize that branch-and-bound and Lagrangian relaxation methods have been extensively used in the literature on a variety of discrete optimization applications. See, for example, Lucena (1992). However, one needs to carefully exploit the particular structure of the problem at hand in order to obtain a successful implementation for large-scale problems.

Mathematical Programming Formulation and Lagrangian Relaxation Problem

We now describe our mathematical programming formulation of the product line design problem. Let J denote the set of all possible products and let I denote the set of consumers. For each consumer $i \in I$, her preference is represented by a mapping $\prec_i$, where $j_2 \prec_i j_1$ means that consumer i prefers j_1 over j_2. The conjoint model implies that all products can be linearly ordered by each consumer, or equivalently, that the preferences are transitive. Let p_j denote the profit obtained from selling one unit of product j for $j \in J$, and denote by k the size of the product line to be designed ($k = 5$ in our problem instance). We associate a binary variable x_{ij} with consumer i's decision to buy product j or not, and we associate another binary variable y_j with the firm's decision to produce product j or not. The resulting mathematical programming formulation of the product line design problem is as follows:

(MP): $\max_{x,y} \sum_{j\in J}\sum_{i\in I} p_j x_{ij}$

s.t. $\sum_{j\in J} y_j \le k$		(1.1) (capacity)
$\sum_{j\in J} x_{ij} \le 1$	for all $i \in I$	(1.2) (consumption)
$x_{ij} \le y_j$	for all $(i,j) \in I \times J$	(1.3) (availability)
$x_{ij} \le 1 - y_{j'}$	for all $(i,j) \in I \times J, j \prec_i j'$	(1.4) (preferences)
$x_{ij} \in \{0,1\}, y_j \in \{0,1\}$	for all $(i,j) \in I \times J$	(1.5) (binary)

There are four types of constraints in the model: the firm's capacity, consumers' consumption, product availability, and consumers' preferences. The capacity constraint (1.1) ensures that the size of the product line is at most k. The model ensures that each consumer will choose at most one product in (1.2). Moreover, consumers can choose product j only if it is available, which is reflected in (1.3). Lastly, each consumer must choose according to her preferences among the existing products, captured by the preference inequalities (1.4). This model is general enough to capture all the relevant features of our problem instance, including m competing products (simply introduce m

artificial products which are always available (the corresponding $y_j = 1$) and have zero profit, and use $k + m$ instead of k).

In order to define the Lagrangian relaxation, we need to construct a dual function based on (MP) which will generate upper bounds on the optimal value. The underlying idea is to remove some constraints from the model and instead introduce penalties into the objective function for violating those same constraints. Define the set

$$Q := \left\{ (x,y) \in \{0,1\}^{I \times J + J} : x_{ij} \le y_j, \sum_{j \in J} y_j \le k \right\} = \left\{ (x,y) : (1.1), (1.3), \text{ and } (1.5) \text{ are satisfied} \right\}$$

and nonnegative Lagrange multipliers λ and σ for the consumption constraints (1.2) and preference constraints (1.4), respectively. Now define the following dual function:

$$f(\sigma,\lambda) = \max_{(x,y) \in Q} \sum_{(i,j)} p_j x_{ij} - \sum_{i \in I} \sigma_i \left(\sum_{j \in J} x_{ij} - 1 \right) - \sum_{(i,j)} \sum_{j' \succ_i j} \lambda_{ijj'} \left(x_{ij} + y_{j'} - 1 \right). \qquad (2)$$

Lagrangian relaxation is built around the observation that for every $\sigma \ge 0$ and $\lambda \ge 0$, *f(σ, λ)* is an upper bound on the optimal value of (MP). To see this, consider any feasible solution (*x, y*) of (MP). By construction, (*x, y*) $\in$ *Q* and the additional terms are nonnegative since $\sigma \ge 0$ and $\lambda \ge 0$. In order to obtain the sharpest upper bound, we attempt to solve the associated Lagrange dual problem:

$$\begin{aligned} \min_{\sigma,\lambda} \quad & f(\sigma,\lambda) \\ \text{s.t.} \quad & \sigma \ge 0, \lambda \ge 0. \end{aligned} \qquad (3)$$

Since *f(σ, λ)* is actually a convex function, this minimization can be efficiently implemented, at least in a theoretical sense.

Computational Considerations of the Lagrangian Relaxation Problem

Two fundamental conditions must be met for the Lagrangian relaxation approach to be sensible in practice: first, the dual function (2) must be easy to evaluate for any given $(\sigma,\lambda) \ge 0$, and second, the dual problem (3) must not be large-scale, that is, the number of Lagrange multipliers should not be excessively large, say, no larger than 10^6. We now examine these two issues in detail. In order to evaluate the dual function *f(σ, λ)*, we need to exploit its particular structure. The key observation is that a consumer can buy many products in the relaxed problem (2). Thus, looking at the "Lagrangian profit" associated with the variable x_{ij} in (2) ($\ell_{ij} := p_j - \sigma_i - \sum_{j' \succ_i j} \lambda_{ijj'}$), it is optimal to set $x_{ij} = 0$ if $\ell_{ij} < 0$. On the other hand, if $\ell_{ij} \ge 0$, it is optimal to set $x_{ij} = y_j$, that is, consumer *i* buys all products with positive Lagrangian profits that are available. Therefore we can restate the dual function (2) as:

$$f(\sigma,\lambda) = \max_y \sum_{j\in J}\left(\sum_{i\in I}\max\{\ell_{ij},0\} - \sum_{j'\succ_i j}\lambda_{ijj'}\right)y_j + \sum_{j\in J}\sum_{j'\succ_i j}\lambda_{ijj'} + \sum_{i\in I}\sigma_i$$
$$\text{s.t.}\sum_{j\in J} y_j \le k,\ y_j \in \{0,1\}\ j\in J. \qquad (4)$$

For fixed σ and λ, the evaluation of (4) is simply to choose the k products associated with the k largest coefficients $c_j = \left(\sum_{i\in I}\max\{\ell_{ij},0\} - \sum_{j'\succ_i j}\lambda_{ijj'}\right)$.

As mentioned earlier, the value obtained in (3) usually does not equal the optimal value of (1). In order to obtain even better bounds, we construct "valid inequalities" for the model (1) and dualize them as we did with the consumption and preferences constraints. (We refer the reader to Nemhauser and Wolsey (1988) for methods for constructing and using valid inequalities.) Although such valid inequalities are redundant for (1), they are not redundant for the associated dual problem (3). We have developed three different families of valid inequalities specifically for our algorithmic setting:

Enlarged preference inequalities: for each $j\in J$, let $H(j) \subseteq \{(i,m): j\succ_i m\}$ be a set of consumer-product pairs, and let N be the number of different consumers in $H(j)$. The following inequality is valid for (MP):

$$\sum_{(i,m)\in H(j)} x_{im} + Ny_j \le N. \qquad (5)$$

Circular preference inequalities: given three consumer-product pairs $\{(i_k,j_k)\}_{k=1}^3$ such that $j_2\succ_{i_1} j_1 \succ_{i_3} j_3 \succ_{i_2} j_2$ holds, the following inequality is valid for (MP):

$$x_{i_1 j_1} + x_{i_2 j_2} + x_{i_3 j_3} \le 1. \qquad (6)$$

Exclusive preference inequalities: for a subset of consumers $Z\subset I$, let $\{T_i\}_{i\in Z}\subset J\setminus\{j\}$, be a family of disjoint subsets of products such that

- for $a,b\in Z$, $k\succ_a m$ and $m\succ_b k$ for all $(m,k)\in T_a\times T_b$, and
- for $a\in Z, j\succ_a m$ for all $m\in T_a$.

Then the following inequality is valid for (MP):

$$y_j + \sum_{i\in Z}\sum_{k\in T_i} x_{ik} \le 1. \qquad (7)$$

Notice that the total number of dualized constraints is much bigger than 10^6 (in fact, it is exponential in the number of consumers and products), contradicting one of the fundamental conditions mentioned earlier. To overcome this, we use a dynamic version of the traditional Lagrangian relaxation approach proposed by Lucena (1992) which considers only a small (dynamic) subset of the inequalities at each iteration. This dynamic strategy is particularly suitable to our problem context since we expect that only a very small subset of the constraints will be active at any optimal solution. The relaxed

solutions and associated Lagrange multipliers are used to add and remove inequalities to the set of "active" dualized constraints in a manner that will be described shortly.

In our implementation, all of the consumption constraints (1.2) are always actively dualized, whereas the preference constraints (1.4), enlarged preference (5), circular (6), and exclusive preference (7) constraints are dualized in a dynamic fashion. For the sake of exposition, we describe our general method using only the preference constraints and their Lagrange multipliers λ, instead of all of the inequalities. Denote by D_t the set of dualized inequalities at iteration *t* and let $z(\lambda) = (x(\lambda), y(\lambda))$ denote a solution that achieves (2) for the given $\lambda \geq 0$. Note that a subgradient $s(\lambda)$ of the function $f(\lambda)$ at λ is easily computed once $z(\lambda)$ is known (for details see Bonnans et al. 2003). Our algorithmic scheme for dynamically changing the set of dualized inequalities is as follows:

Step 1 (Initialization). Let $D_0 := \varnothing$, $\lambda^0 := 0$, and $t := 0$.

Step 2 (Evaluate the Function). Compute $f(\lambda^t)$, $z^t = z(\lambda^t)$, and $s^t = s(\lambda^t)$.

Step 3 (New Inequalities). Generate the set A_t of new inequalities based on z^t.

Step 4 (Update Multipliers). Update multipliers λ^t to λ^{t+1} based on s^t.

Step 5 (Update Inequalities). Let $R_t \subseteq \left\{ i \in D_t : \lambda_i^{t+1} = 0 \right\}$ and define $D_{t+1} = D_t \cup A_t \setminus R_t$.

Step 6 (Stopping Criterion). Verify a stopping criterion.

Step 7 (Loop). Set $t \leftarrow t+1$ and go to Step 2.

The convergence of this scheme to an optimal solution depends on the particular rules used to update the Lagrange multipliers, the rules to update the set of active inequalities D_t, and on the rules for the stopping criterion. Under certain regularity assumptions, convergence has been established when bundle methods are used to update the Lagrange multipliers and a specific rule is followed to manage the inequalities (for details see Belloni and Segastizábal 2004). Our implementation uses a variant of the subgradient optimization method (for example, see Fisher 1981) which was also used in Lucena (1992 and 1993), and Belloni and Lucena (2004).

One important step that bears more detailed discussion is how to generate new inequalities based on $(x(\lambda), y(\lambda))$. This is known as the *separation problem* in combinatorial optimization. In our case, due to the structure of the problem, the vector $y(\lambda)$ will have at most *k* nonzero elements, which implies that at most *kI* components of $x(\lambda)$ will be nonzero. This reduces the search for violated inequalities to a relatively small subset only, since any violated inequality must have nonzero variables. For example, for each consumer the search for a violated enlarged preference inequality with $N = 1$ consists of only *k* evaluations of her preference for each of the products assigned to the product line (up to *k* products). We developed elementary heuristics to search for violated circular preference and exclusive preference constraints. Again, the integrality of the relaxed solutions $(x(\lambda), y(\lambda))$ considerably simplifies the search.

The Branch-and-Bound Method.

We start by briefly reviewing branch-and-bound, and refer the more curious reader to Bertsimas and Tsitsiklis (1997) for a more complete description. Branch-and-bound starts with a feasible set and generates associated upper and lower bounds. It chooses to "branch" the current problem into two subproblems where a particular variable is fixed at 0 and 1 in each of the two subproblems, respectively. The method then proceeds recursively: for each subproblem, upper and lower bounds are computed, another variable is chosen to be fixed, and more branches are created. Note that the upper bound will decrease as more variables are fixed. Hopefully, the lower bounds will exceed the upper bounds for many branches, thereby eliminating the need to search those branches any further.

In order to apply the branch-and-bound method to solve (MP), we need to define the so-called branching rule as well as the method to generate upper bounds. The Lagrangian relaxation method is the natural choice to generate upper bounds, since fixed variables can be easily incorporated into the subproblem. In our implementation no new equalities were inserted or deleted at any node, i.e., we kept the same set of inequalities used at the root node. On the other hand, we did update the Lagrange multipliers on certain nodes of the tree to take advantage of the impact of fixed variables. Furthermore, if exactly $k-1$ variables were fixed at 1 in some subproblem, we then solved the subproblem exactly.

With respect to the branching rule, the "*x*" variables were never fixed and branched on our implementation, since fixing and then branching on "*y*" variables had a much larger impact on the upper bounds. The component *j* chosen for branching was determined by computing the largest coefficient c_j.

Table A1. Competing Bags and Optimal Solution
Shaded box indicates feature is included.

Feature	**Competitor**			**Timbuk2**				
Price ($)	70	85	100	80	95	95	100	100
Large Size								
Red Color								
School Logo								
Handle								
PDA Holder								
Cell Holder								
Mesh Pocket								
Velcro Flap								
Reinforcing Boot								

Table A2. Average Performance by Problem Size*

Attributes, levels, products	3,5,4		5,3,4		7,2,4		3,8,3		5,5,3		7,3,3		**Average**
Possible product lines	$9.7*10^6$		$1.4*10^8$		$1.1*10^7$		$2.2*10^7$		$5.1*10^9$		$1.7*10^9$		
Number of customers	50	100	50	100	50	100	50	100	50	100	50	100	
Lagrangian Relaxation with Brand & Bound	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	**100.0%**
Coordinate Ascent (One-Opt)	96.2%	98.4%	91.9%	96.9%	93.7%	95.0%	96.5%	97.1%	91.7%	92.9%	89.5%	92.9%	**94.4%**
Coordinate Ascent (Two-Opt)	98.6%	99.1%	97.6%	97.1%	97.2%	96.6%	97.8%	96.0%	92.1%	93.7%	91.4%	95.2%	**96.0%**
Coordinate Ascent (Three-Opt)	99.0%	98.3%	95.5%	97.3%	95.9%	95.6%	95.6%	96.5%	96.1%	92.5%	89.9%	96.0%	**95.7%**
Genetic Algorithm	100.0%	100.0%	99.8%	99.9%	100.0%	100.0%	100.0%	100.0%	99.9%	99.7%	99.9%	99.7%	**99.9%**
Simulated Annealing	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	**100.0%**
Divide & Conquer	97.7%	99.3%	98.9%	98.4%	99.0%	98.4%	99.5%	99.7%	98.4%	98.6%	99.0%	98.0%	**98.7%**
Greedy Heuristic	98.1%	99.3%	96.9%	95.6%	96.2%	96.9%	99.0%	99.3%	97.9%	96.4%	97.3%	97.1%	**97.5%**
Product-Swapping	99.0%	99.4%	98.2%	98.4%	98.2%	98.7%	99.0%	99.3%	98.2%	97.6%	97.6%	98.2%	**98.5%**
DP Heuristic	97.8%	98.9%	96.5%	96.5%	95.5%	96.6%	97.1%	98.3%	96.0%	95.5%	91.7%	95.2%	**96.3%**
Beam Search	99.3%	99.8%	99.1%	99.4%	98.8%	99.2%	99.7%	99.8%	98.8%	98.5%	98.6%	98.6%	**99.1%**
Nested Partitions	95.9%	98.0%	94.5%	94.4%	95.7%	95.4%	92.2%	96.4%	88.5%	91.8%	92.5%	91.8%	**93.9%**

*Each data point shows the average performance relative to the optimal solution, over ten datasets.

References

Aarts, E., J. Korst, and P. van Laarhoven. "Simulated Annealing." in *Local Search in Combinatorial Optimization*, E. Aarts and J.K. Lenstra (Eds.), John Wiley & Sons, Chichester, UK, 1997, 91-120.

Belloni, A., and C. Sagastizábal, Dynamic Bundle Methods: Application to Combinatorial Optimization, (submitted to *Mathematical Programming*) MIT Operations Research Center Working Paper OR-370-04, 2004.

Belloni, A., and A. Lucena, Lagrangian Heuristic for the Linear Ordering Problem, *Metaheuristics: Computer Decision-Making,* Kluwer Academic Publishers, 2004.

Bertsimas, D., and J. Tsitsiklis, *Introduction to Linear Optimization*, Athena Scientific, 1997.

Bonnans, J. F., Gilbert, J. Ch., Lemaréchal, C., and Sagastizábal, C., *Numerical Optimization: Theoretical and Practical Aspects*, Springer-Verlag, Berlin, 2003.

Camm, J.D., Cochran, J.J., Curry, D.J., and Kannan, S., "Conjoint Optimization: An Exact Branch-and-Bound Algorithm for the Share-of-Choice Problem," *Management Science*, 52, 3 (2006), 435-447.

Fisher, M., The Lagrangian relaxation method for solving integer programming problems, *Management Science*, 27:1–18, 1981.

Lucena, A., Steiner problem in graphs: Lagrangean relaxation and cutting-planes, *COAL Bulletin,* 21:2–8, 1992.

Lucena, A., Tight bounds for the Steiner problem in graphs, *Proceedings of NETFLOW93*, pp. 147–154, 1993.

Nemhauser, G., and L. Wolsey, *Integer and Combinatorial Optimization*, Wiley & Sons, 1988.