Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 34 (2007) 1777–1799
computers & operations research
www.elsevier.com/locate/cor

A heuristic genetic algorithm for product portfolio planning

Jianxin (Roger) Jiao*, Yiyang Zhang, Yi Wang

School of Mechanical and Aerospace Engineering, Nanyang Technological University, Nanyang Avenue 50, Singapore, 639798, Singapore

Available online 22 August 2005

Abstract

Product portfolio planning has been recognized as a critical decision facing all companies across industries. It aims at the selection of a near-optimal mix of products and attribute levels to offer in the target market. It constitutes a combinatorial optimization problem that is deemed to be NP-hard in nature. Conventional enumeration-based optimization techniques become inhibitive given that the number of possible combinations may be enormous. Genetic algorithms have been proven to excel in solving combinatorial optimization problems. This paper develops a heuristic genetic algorithm for solving the product portfolio planning problem more effectively. A generic encoding scheme is introduced to synchronize product portfolio generation and selection coherently. The fitness function is established based on a shared surplus measure leveraging both the customer and engineering concerns. An unbalanced index is proposed to model the elitism of product portfolio solutions.

Keywords: Mass customization; Genetic algorithm; Product portfolio; Variety management; Customer decision making

1. Introduction

To compete in the marketplace, manufacturers have been seeking for expansion of their product lines and differentiation of their product offerings with the intuitively appealing belief that large product variety may stimulate sales and thus conduce to revenue [1]. Initially variety does improve sales as the offerings become more attractive; but as the variety keeps increasing, the law of diminishing returns suggests that

* Corresponding author. Tel.: +65 67904143; fax: +65 67911859.
E-mail address: jiao@pmail.ntu.edu.sg (J. Jiao).

0305-0548/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2005.05.033

the benefits do not keep pace [2]. The consequence of variety explosion manifests itself through several ramifications, including increasing costs due to an exponential growth of complexity, inhibiting benefits from economy of scale, exacerbating inventory imbalances and warehouse suffocation, and jeopardizing the efficiency of manufacturing processes and distribution systems, to name but a few [3]. Facing such a variety dilemma, the company must optimize its external variety with respect to the internal complexity resulting from product differentiation [4].

On the other hand, the practice of making wide variety of products available and letting customers vote on the shelf seems not only to be wasteful or unaffordable, but also tends to constrain customers' ultimate satisfaction, leading to so-called mass confusion [5]. Pine et al. [6] have reported the common problem of companies giving customers more choices than they actually want or need. For example, Toyota found that 20% of its product variety accounted for 80% of its sales and Nissan reportedly offered 87 different types of steering wheels. Therefore, rather than creating various products in accordance with all anticipating customer needs, it becomes an important campaign for the manufacturer to offer the "right" product variety to the target market.

Such decisions on near-optimal amount of product offerings adhere to the general wisdom as suggested in the Boston Consulting Group's notion of product portfolio strategy [7]. While representing the spectrum of a company's product offerings, the product portfolio must be carefully set up, planned and managed so as to match those customer needs in the target market [8]. The product portfolio strategy has far-reaching impact on the company's business success to achieve financial goals in maximizing return and R&D productivity, to maintain the competitive edge of the business by increasing sales and market share, to allocate scare resources properly and efficiently, to forge the link between project selection and business strategies, to better communicate priorities within the organization both vertically and horizontally, and so on [9].

Product portfolio planning (PPP) has been classified as a combinatorial optimization problem, in that each company strives for the optimality of its product offerings through various combinations of products and attribute levels [10]. In practice, the PPP problem has been approached from two different perspectives. The first one is to select a product portfolio from a finite set of candidate items [11]. The second one is to construct a product portfolio directly from part-worths utility data [12]. If the number of attributes and the number of attribute levels go large, or most combinations of attribute levels are to be used for defining feasible products, it deems to be computationally infeasible to enumerate the utilities of all candidate items [13].

Towards this end, this paper develops a heuristic genetic algorithm (HGA) for solving the PPP problem. The objective of this research is to develop a practical solution method that can find near-optimal solutions and assist marketing managers in product portfolio decision making. The proposed HGA constructs a product portfolio directly from part-worths utilities estimated based on conjoint analysis [14]. The proposed solution method does not refer to any specific type of products, and thus it can be applied to a variety of product or service development problems.

The remainder of the paper proceeds as follows. In the next section, the background leading to this research is reviewed and accordingly the key technical challenges are identified. Section 3 presents a formal description and formulation of the PPP problem. An optimization framework and the structural properties of the model are discussed in Section 4. The HGA design and solution procedures are introduced in Section 5. An application of the proposed HGA to notebook computer portfolio planning is reported in Section 6. Managerial implications and possible extensions of this research are discussed in Section 7 and the paper is concluded in Section 8.

2. PPP as a combinatorial optimization problem

PPP has its origins in the fields of optimal product design [15], product positioning [10] and product line design [12]. All these problems constitute a type of combinatorial optimization problems due to their purpose of achieving a near-optimal combination of discrete products and/or attribute levels [13]. In general, combinatorial optimization problems are characterized by a finite number of feasible solutions. Let $E = \{e_1, e_1, \ldots, e_n\}$ be a finite set, Ω a set of feasible solutions defined over E, and $f : \Omega \to R$ an objective function. A combinatorial optimization problem is to find a solution in Ω whose objective value is minimum or maximum [16].

By intuition, finding the near-optimal solution for a finite combinatorial optimization problem could be done by simple enumeration. In practice, however, this technique is often impossible because the number of feasible solutions may be enormous. A number of methods and algorithms have been developed to solve combinatorial optimization problems. Sait and Youssef [17] have divided them into two groups: exact algorithms and approximation algorithms. Due to the enumerative nature, exact algorithms are not easy to design with moderate computational effort as can be seen from the complexity theory [18]. A major trend in solving such hard problems is to utilize an effective heuristics search [19]. In recent studies, some meta-heuristics such as multi-start local search [20], simulated annealing [21], tabu search [22] and genetic algorithms [23] have been commonly adopted. Reeves [24] and Aarts and Lenstra [25] have reported a thorough survey of these approaches to commonly defined combinatorial problems.

A dynamic-programming heuristic method has been developed by Kohli and Sukumar [12]. Nair et al. [13] have developed a beam search heuristic method and performed a computational study of the beam search method and the dynamic programming heuristic method. Based on an extensive comparative computational study Alexouda [26] has found that the evolutionary algorithms are close to optimal and, in most cases, the evolutionary algorithms obtain a better solution than that found by the beam search method. However, genetic algorithms are in general incapable of fine-tuning for obtaining the global optimum [20]. As a result, various modifications have been done by incorporating local search techniques into the evolution process for particular problems [27,28].

3. Description of the PPP problem

Consider such a scenario that a large set of product attributes, $A \equiv \{a_k | k = 1, \ldots, K\}$, have been identified (a few methods are available, for example, [29]), given that the firm has the capabilities (both design and production) to produce all these attributes. Each attribute, $\forall a_k \in A$, possesses a few levels, may it be discrete or continuous, i.e., $A_k^* \equiv \{a_{kl}^* | l = 1, \ldots, L_k\}$.

A set of potential product profiles, $Z \equiv \{\vec{z}_j | j = 1, \ldots, J\}$, are generated by choosing one of the levels for certain attributes, subjective to satisfying certain configuration constraints [30]. That is, a product assumes certain attribute levels that correspond to a subset of A. Each product, $\forall \vec{z}_j \in Z$, is defined as a vector of specific attribute levels, i.e., $\vec{z}_j = [a_{kl_j}^*]_K$, where any $a_{kl_j}^* = \varnothing$ indicates that product $\vec{z}_j$ does not contain attribute a_k; and any $a_{kl_j}^* \neq \varnothing$ represents an element of the set of attribute levels that can be assumed by product $\vec{z}_j$, i.e., $\{a_{kl_j}^*\}_K \in \{A_1^* \times A_2^* \times \cdots \times A_K^*\}$.

A product portfolio, Λ, is a set consisting of a few selected product profiles, i.e., $\Lambda \equiv \{\vec{z}_j | j = 1, \ldots, j^\dagger\} \subseteq Z, \exists J^\dagger \in \{1, \ldots, J\}$, denotes the number of products contained in the product portfolio.

Every product is associated with certain engineering costs, denoted as $\{C_j\}_J$. The manufacturer must make decisions to select what products to offer as well as their respective prices, $\{p_j\}_J$.

There are multiple market segments, $S \equiv \{s_i | i = 1, \ldots, I\}$, each containing homogeneous customers, with a size, Q_i. Various customer preferences on diverse products are represented by respective utilities, $\{U_{ij}\}_{I \cdot J}$. Product demands or market shares, $\{P_{ij}\}_{I \cdot J}$, are described by the probabilities of customers' choosing products, denoted as customer or segment-product pairs, $\{(s_i, \vec{z}_j)\}_{I \cdot J} \in S \times Z$.

4. Mathematical model of product portfolio planning

This research addresses the PPP problem with the goal of maximizing an expected surplus from both the customer and engineering perspectives [31]. The price has been commonly treated as a separate attribute that can be chosen from a limited number of values for each product [13,32]. Adding price as one more attribute, the attribute set becomes $A \equiv \{a_k\}_{K+1}$, where a_{K+1} represents the price possessing a few levels, i.e., $A^*_{K+1} \equiv \{a^*_{(K+1)l} | l = 1, \ldots, L_{K+1}\}$. Then, a PPP problem can be formulated as a mixed integer program, as below,

$$\text{Maximize} \quad E[V] = \sum_{i=1}^{I} \sum_{j=1}^{J} \frac{U_{ij}}{C_j} P_{ij} Q_i y_j, \tag{1a}$$

$$\text{Subject to} \quad \sum_{l=1}^{L_k} x_{jkl} = 1, \quad \forall j \in \{1, \ldots, J\}, \ \ \forall k \in \{1, \ldots, K+1\}, \tag{1b}$$

$$\sum_{k=1}^{k+1} \sum_{l=1}^{L_k} |x_{jkl} - x_{j'kl}| > 0, \quad \forall j, j' \in \{1, \ldots, J\}, \ \ j \neq j', \tag{1c}$$

$$\sum_{j=1}^{J} y_j \leqslant J^{\dagger}, \quad \forall J^{\dagger} \in \{1 \ldots, J\}, \tag{1d}$$

$$x_{jkl}, y_j \in \{0, 1\}, \quad \forall j \in \{1, \ldots, J\}, \ \ \forall k \in \{1, \ldots, K+1\}, \ \ \forall l \in \{1, \ldots, L_k\}, \tag{1e}$$

where $E[\cdot]$ denotes the expected value of the shared surplus, V, which is defined as the utility per cost, modified by the probabilistic choice model, $\{P_{ij}\}_{I \cdot J}$, and the market size, $\{Q_i\}_I$; C_j indicates the cost of offering product $\vec{z}_j$; U_{ij} refers to the composite utility of segment s_i for product $\vec{z}_j$ constructed from part-worth utilities of individual attribute levels, $\{A^*_k\}_{K+1}$; and y_j is a binary variable such that $y_j = 1$ if the manufacturer decides to offer product $\vec{z}_j$ and $y_j = 0$ otherwise.

In the above mathematical program, there are two types of decision variables involved, i.e., x_{jkl} and y_j, representing two layers of decision-making in portfolio planning, respectively. The first layer is the selection of attribute and their levels for different products (i.e., product generation); the second one is to decide which products to offer (i.e., product selection). Both types of decisions depend on a simultaneous satisfaction of the target segments. The manufacturer's decisions about *what* (i.e., layer I

decision-making) and *which* (i.e., layer II decision-making) products to offer to the target segments are implied in various instances of $\{x_{jkl}|\forall j,k,l\}$ and $\{y_j|\forall j\}$, respectively. As a result, a near-optimal product portfolio, $\Lambda^{\dagger} \equiv \{\vec{z}_j^{\dagger}|j=1,\ldots,J^{\dagger}\}$ is yielded as a combination of selected products corresponding to $\{y_j|\forall j\}$, where each selected product , $\vec{z}_j^{\dagger}$, comprises a few selected attributes and the associated levels corresponding to $\{x_{jkl}|\forall j,k,l\}$.

Following the part-worth model, the utility of the ith segment for the jth product, U_{ij}, is assumed to be a linear function of the part-worth preferences of the attribute levels of product $\vec{z}_j$, i.e.,

$$U_{ij} = \sum_{k=1}^{K}\sum_{l=1}^{L_k}(w_{jk}u_{ikl}x_{jkl} + \pi_j) + \varepsilon_{ij}, \tag{2}$$

where u_{ikl} is the part-worth utility of segment s_i for the lth level of attribute a_k (i.e., a_{kl}^{*}) individually; w_{jk} is the utility weights among attributes, $\{a_k\}_K$, contained in product $\vec{z}_j$; π_j is a constant associated with the derivation of a composite utility from part-worth utilities with respect to product $\vec{z}_j$; ε_{ij} is an error term for each segment-product pair; and x_{jkl} is a binary variable such that $x_{jkl}=1$ if the lth level of attribute a_k is contained in product $\vec{z}_j$ and $x_{jkl}=0$ otherwise. The choice probability, P_{ij}, that a customer or a segment, $\exists s_i \in S$, chooses a product, $\exists\,\vec{z}_j \in Z$, with N competing products, is defined under the conditional multinomial logit choice rule [33].

Jiao and Tseng [34] have proposed to model the cost consequences of providing variety based on varying impacts on process capabilities. To circumvent the difficulties inherent in estimating the accurate cost figures, this research adopts a pragmatic costing approach based on standard time estimation developed by Jiao and Tseng [35]. Then, the expected cycle time can be used as a performance indicator of variations in process capabilities [34]. Hence, the cost function, C_j, corresponding to product $\vec{z}_j$, can be formulated based on the respective one-side specification limit process capability index.

To select the best product portfolio with nearly the same shared surplus, a selection rule is adopted to identify the most balanced product portfolio. According to Li and Azarm [36], a balanced product portfolio means that all products contribute evenly or nearly evenly to the shared surplus; otherwise an unbalanced product portfolio. In general, a balanced product portfolio is more preferable, as it tends to perform more stable when there exist unexpected changes in the market. An unbalanced product portfolio, to the contrary, may suffer significantly when market changes diminish the performance of one or two dominating products in the portfolio. To quantify the extent of a balanced distribution of products' individual contributions to the entire product portfolio, an unbalance index is defined as the following:

$$\psi = \sqrt{\sum_{j=1}^{J^{\dagger}}\left(\frac{E[V_j]}{E[V]} - \frac{1}{M}\right)^2}, \tag{3}$$

where M is the total number of products in a portfolio, $E[V_j]$ is the expected shared surplus of product $\vec{z}_j$, and $E[V]$ is the expected shared surplus of all products, $\{\vec{z}_j\}_{J^{\dagger}}$. In an absolutely balanced portfolio, the shared surplus of portfolio is evenly distributed among all products, i.e., $E[V_j]/E[V] \to 1/M$, thus $\psi \to 0$. Therefore, the lower the value of the unbalance index is, the more balanced is the

distribution of shared surplus (fitness) among the products, and thus the more desirable is the portfolio chromosome.

5. A heuristic GA for PPP

As the number of attributes and levels associated with a product increases, so does the number of combinations of products for portfolios. A product with nine attributes of three levels each may produce $3^9 = 19,683$ possible variants. A product portfolio consisting of maximal three such products may yield $(3^9)^3 + (3^9)^2 + (3^9)^1 = 7.62598 \times 10^{12}$ possible combinations. Complete enumeration to obtain optimal product selections in portfolio planning becomes numerically prohibitive [37]. The conjoint-based search for an optimal product portfolio always results in combinatorial optimization problems because typically discrete attributes are used in conjoint analysis [10]. Nearly all of these problems are known to be mathematically intractable or NP-hard, and thus mainly heuristic solution procedures have been proposed for the various problem types [13].

Comparing with traditional calculus-based or approximation optimization techniques, genetic algorithms (GA) have been proven to excel in solving combinatorial optimization problems [38]. The GA approach adopts a probabilistic search technique based on the principle of natural selection by survival of the fittest and merely uses objective function information, and thus is easily adjustable to different objectives with little algorithmic modification [23]. An important feature of GA is that it allows product profiles to be constructed directly from attribute level part-worths data [12]. This is particularly preferable to reference set enumeration if the number of attributes and their levels is large and most multi-attribute products represented by different attribute level combinations are economically and technologically feasible [13].

Hence, a GA approach is employed in this research to solve the mixed integer program in Eqs. (1a)–(1e). The focus is to develop an efficient algorithm that is capable of producing acceptable solutions for the combinatorial optimization problem involving a wide variety of configurations of attributes and their levels as well as product profiles in portfolio planning. In accordance with a generic variety structure inherent in product families [30], a heuristic GA is formulated as follows.

5.1. Generic encoding

The first step in the implementation of a heuristic GA involves the representation of a problem to be solved with a finite-length string called chromosome. A generic strategy for encoding the portfolio planning problem is illustrated in Fig. 1, with an example shown in Fig. 2. A product portfolio is represented by a chromosome consisting of a string. Each fragment of the chromosome (i.e., substring) represents a product contained in the portfolio. Each element of the string, called gene, indicates an attribute of the product. The value assumed by a gene, called allele, represents an index of the attribute level instantiated by an attribute. A portfolio (chromosome) consists of one to many products (fragments of chromosome), exhibiting a type of composition (AND) relationships. Likewise, each product (fragment of chromosome) comprises one to many attributes (genes). Nevertheless, each attribute (gene) can assume one and only one out of many possible attribute levels (alleles), suggesting an exclusive all (XOR) instantiation.

The format of an allele may be either a binary or integer number [23]. The binary format is the most general form widely used for modeling the binary-selection type of problems [39]. In our case,

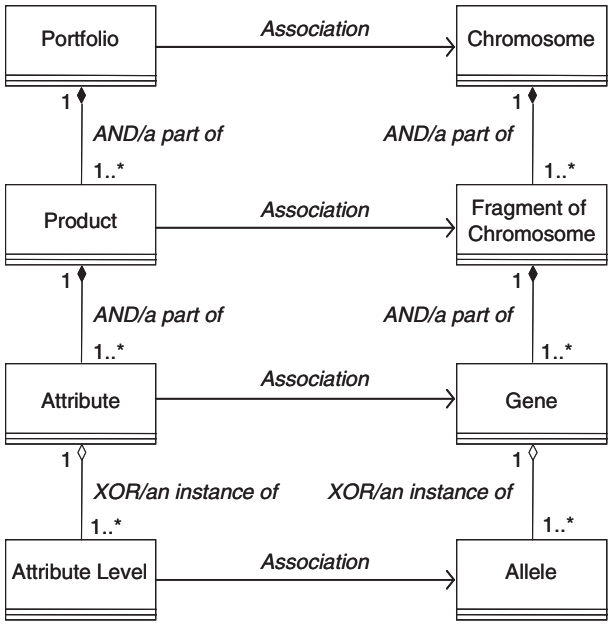

Fig. 1. Generic encoding for product portfolio.

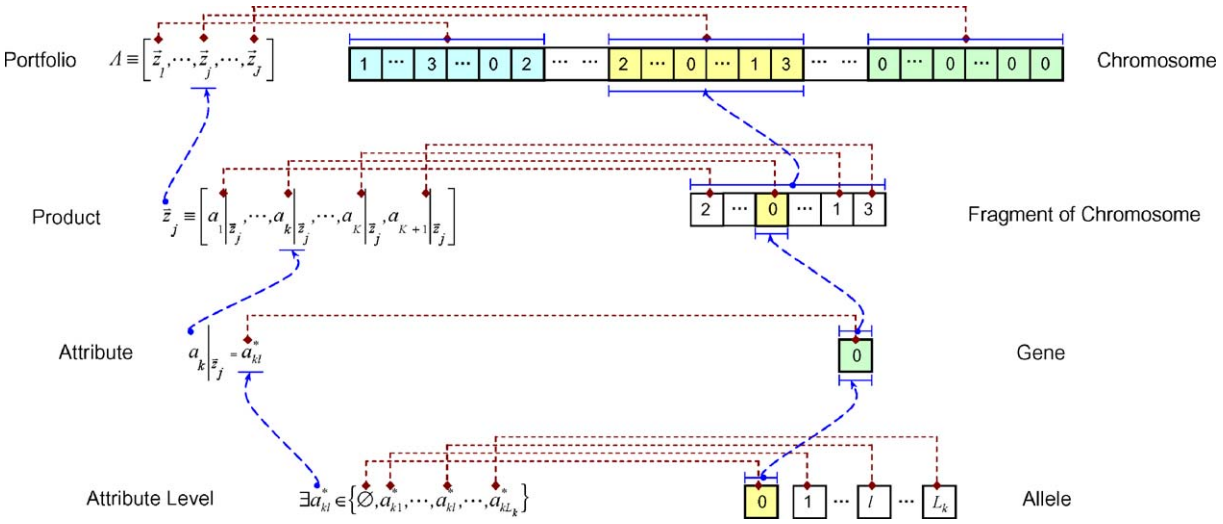

Fig. 2. An illustration of generic encoding.

each attribute (genes) may assume multiple levels (alleles), resulting in a multi-selection problem. Therefore, the integer format is adopted for representing multiple choices among attribute levels. Each gene assumes an integer number that corresponds to the index of the attribute level associated with a particular attribute.

Given $J^{\dagger} \leqslant J$ products to be selected for a product portfolio, $\Lambda = \{\vec{z}_j\}_{J^{\dagger}}$, and $K+1$ attributes in each product, $\vec{z}_j$, a generic string of the chromosome is defined to be composed of J substrings, with $J - J^{\dagger}$ empty substrings corresponding to those unselected products, and contains a total number of $J \cdot (K+1)$ genes, with each substring consisting of $K+1$ genes.

Further introduce an allele equal to 0 as the default value for every gene. This indicates that the corresponding attribute is not contained in a product. Then with L_k possible levels for an attribute, a_k, the corresponding gene may assume an allele from the set, $\{0, 1, \ldots, L_k\}$, meaning that a total number of $L_k + 1$ alleles are available for each gene. This corresponds to the fact that an attribute, a_k, may assume a de facto level, that is, $\exists a^*_{kl} \in \{\varnothing, a^*_{k1}, \ldots, a^*_{kl}, \ldots, a^*_{kL_k}\}$. If all genes throughout a substring assume $\{0\}_{K+1}$ alleles, then it means that the corresponding product is not selected in the portfolio. In this way, a chromosome enables a unified structure, through which various portfolios consisting of different numbers of products can be represented within a generic product portfolio, $\Lambda = \{\vec{z}_j\}_J$. Each individual portfolio can be instantiated from the same generic product portfolio by indirect identification of zero or non-zero alleles for all substrings [30].

For example, the chromosome shown in Fig. 2 suggests that product $\vec{z}_j$ is not selected for the portfolio (i.e., $y_J = 0$) as the corresponding substring is totally empty. As far as product $\vec{z}_1$ is concerned (i.e., $y_1 = 1$), the 1st allele assumes a value of 2 indicating that the 1st attribute of the product chooses the 2nd attribute level associated with this attribute (i.e., $x_{112} = 1$). The last $(K+1)$ allele of the 1st substring suggests that the price attribute takes on the 3rd price level for product $\vec{z}_1$ (i.e., $x_{1(K+1)3} = 1$). On the other hand, the kth allele assumes a value of 0, indicating that the kth attribute is not contained in product $\vec{z}_1$ (i.e., $x_{1kl} = 0, \forall l \in \{1, \ldots, L_k\}$).

Following the basic GA procedures [39], the PPP problem in Eqs. (1a)–(1e) is solved iteratively, as depicted below and also shown in Fig. 3.

5.2. Initialization

Initialization involves generating initial solutions to the problem. The initial solutions can be generated either randomly or using some heuristic methods [40]. Considering the feasibility of product configurations, an initial population of product portfolios of size M, $\{\Lambda_m\}_M$, is determined a priori and accordingly M chromosome strings are encoded, respectively. Each chromosome string is assigned a fitness value in lieu of its expected shared surplus obtained by calculating Eq. (1a).

The population size, M, directly affects the computational efficiency of GA. A larger population size gives the algorithm a higher chance of success by exploring a larger solution space; but leads to more calculations. While a standard value could be suggested by extensive experimentation [23], this research sets a population size of 100 chromosomes.

5.3. Handling of configuration constraints

In order to obtain feasible solutions, each chromosome must satisfy certain configuration constraints on product generation from combinations of attribute levels. They constitute two types of constraints: compatibility constraints and selection constraints. Compatibility constraints refer to the restrictions on choices of attribute levels (e.g., size compatible) and are generally described as IF THEN rules [30,41]. Selection constraints refer to those conjoint, exclusiveness, divergence and capacity conditions as postulated in Eqs. (1b)–(1e).

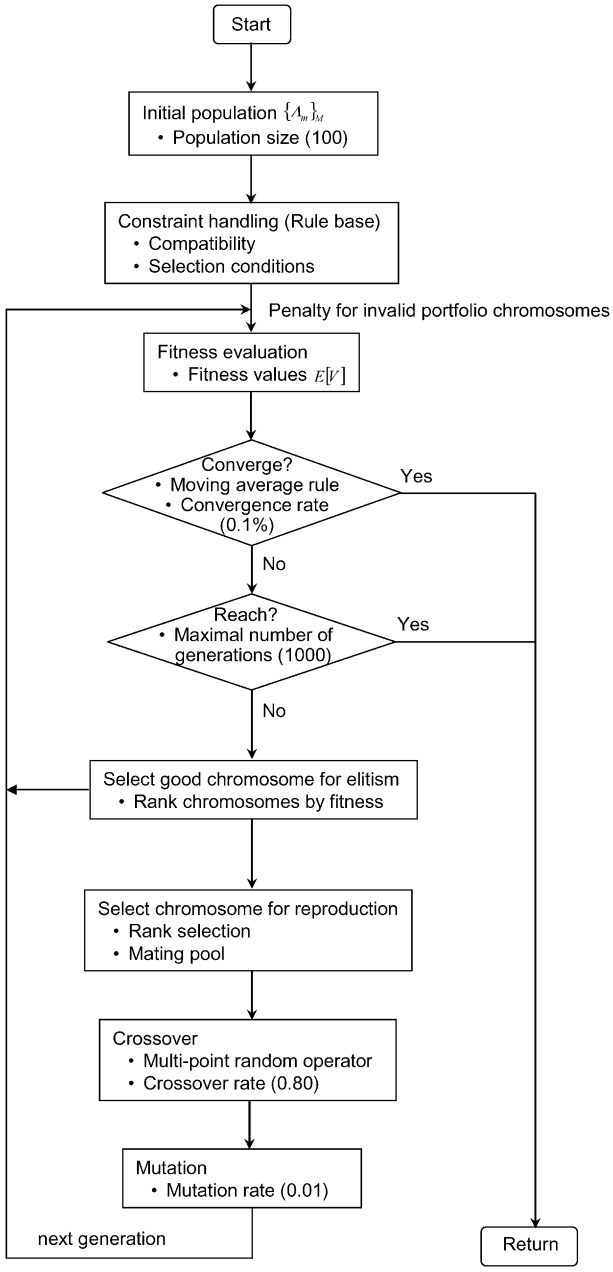

Fig. 3. Procedure of the heuristic genetic algorithm.

A number of methods of constraint handling have been reported in the literature, such as the repairing, variable restricting, and modifying generic operator methods [39]. This research adopts a penalizing strategy. Whenever a new chromosome is generated, a constraint check is conducted with respect to all types of constraints, and those invalid ones are penalized in the population.

Most existing GA implementations incorporate constraint handling into the GA process. This makes GA operations very complex and less efficient. For example, Steiner and Hruschka [38] have introduced extra exit conditions for crossover and mutation in order to deal with the divergence constraint. This research designs a separate constraint check module as a filter at the outset of the GA process. The constraint rules are generated based on the designers' experience and production capability. The generated rules are stored in a pool. Whenever a new chromosome is produced, it must be checked with the pool. If any genes of the new chromosome are found in the pool, the chromosome is penalized. As a result, only valid chromosomes are kept high fitness, whilst a standard GA process can be maintained without being intervened by concerning the validity of GA operations or the feasibility of each offspring.

5.4. Fitness function

A fitness function must be used to evaluate the fitness value of each individual chromosome within the population of each generation. Good chromosomes should probably expose to more opportunities to be selected as a parent, whereas poor ones may not be selected at all. Within the context of PPP, the fitness function used is the expected shared surplus as described in Eq. (1a).

5.5. Selection and reproduction

With the optimization of an expected shared surplus, the fitness values are continuously increasing until a near-optimal solution is found. Once the fitness function is defined and used for the first generation, the GA starts the parent selection and reproduction process. Parent selection is a process that allocates reproductive opportunities among chromosome population. The most popular selection method is the roulette wheel selection. The roulette wheel selection is one probabilistic selection method, that is, a reproduction probability is assigned to each chromosome based on its fitness value. Then the roulette wheel is filled using the respective cumulative probabilities of every chromosome. The areas of the sections on the wheel depend on the fitness values of the associated chromosomes, with fitter chromosomes occupying larger areas in this biased roulette wheel, thus increasing their chances of survival. The roulette wheel selection can be implemented by generating random numbers between 0 and 1 in accordance with the cumulative reproduction probabilities [40].

The advantage of probabilistic selection is that the better the chromosomes are, the more chances to be selected they have. Thus, those chromosomes with better fitness gain more opportunities to change their good components to reproduce better offspring. But a biased selection sometimes may lead to premature convergence although it enables the convergence of the search [23]. Imagine a roulette wheel selection where all the chromosomes in the population are placed, the size of the section in the roulette wheel is proportional to the value of the fitness function of every chromosome—the bigger the value is, the larger the section is. In this case, if one chromosome is dominant in the population, this dominant chromosome with bigger fitness value will be selected more times. For example, if the best chromosome fitness is 90% of the sum of all fitness then the other chromosomes will have very few chances to be selected. Thus, the diversity of the population is destroyed, so as the performance of the global searching capability of genetic algorithm. In this case, this research adopts rank selection to select the appropriate chromosomes for crossover and mutation operations. The rank selection is also a probabilistic selection method. Rank selection ranks the population first and then every chromosome receives fitness value determined by this ranking. The worst will have the fitness 1, the second worst 2, etc., and the best will have fitness

N (number of chromosomes in population). Rank selection decreases the difference between dominant chromosomes and non-dominant ones, thus all the chromosomes have a chance to be selected to keep the diversity of the population.

5.6. Crossover

After reproduction, each two of the parent strings in the mating pool are picked randomly and each pair of strings undergoes crossover with a probability. Crossover requires two individual chromosomes to exchange their genetic compositions. The offspring thus inherits some genes from parents via such operations. While a number of crossover operators are available for specific encoding schemes [40], this research adopts a multi-point random crossover operator. The idea behind multi-point is that parts of the chromosome that contribute to most of the performance of a particular individual may not necessarily be contained in adjacent substrings. Compared with single-point crossover operator, the disruptive nature of multi-point crossover appears to encourage the exploration of the search space, rather than favoring the convergence to highly fit individuals early in the search, thus making the search more robust.

For the PPP problem, the product portfolio comprises several different products which are composed of many attributes. The complexity of the problem results in a long string representing the chromosome. Adopting single-point crossover is inclined to keep most adjacent substrings intact thus resulting in the premature. In this regard, for each substring, we adopt single-point crossover operator to encourage its changing. Thus, the whole chromosome is implemented with a multi-point crossover operation.

Within a generic encoding chromosome, for every substring, one crossover point is randomly located and the integer string of an offspring is first copied from the first parent from the beginning till the crossover point; and then the rest is added by copying from the second parent from the crossover point to the end. The order of combination is reversed for the other offspring. In regard to the generic chromosome, for each substring, there are $(K-1)$ cutting points, and there are totally $J \cdot (K-1)$ cutting points.

The probability of crossover is characterized by a crossover rate, indicating the percentage of chromosomes in each generation that experience crossover. Crossover aims at producing new chromosomes that possess good elements of old chromosomes. Nonetheless it is also desirable to allow some chromosomes, in particular those good ones, to survive without change in the next generation (namely elitism). Therefore, this research adopts a crossover rate of 0.80. In practice, this value could be selected based on sensitivity analysis of trial examples using crossover rates that range, for example, 0.05–0.95.

5.7. Mutation

Mutation is applied to each offspring individually after crossover. It randomly picks a gene within each string with a small probability (referred to as mutation rate) and alters the corresponding attribute level at random. This process enables a small amount of random search, and thus ensures that the GA search does not quickly converge at a local optimum. But it should not occur very often, otherwise the GA becomes a pure random search method [23]. Empirical findings have suggested a mutation rate of 0.01 as a rule of thumb to obtain good solutions [39]. While reproduction reduces the diversity of chromosomes in a population, mutation maintains a certain degree of heterogeneity of solutions which is necessary to avoid premature convergence of the GA process [38].

5.8. Termination

The processes of crossover and reproduction are repeated until the population converges or reaches a pre-specified number of generations. The number of generations has direct consequence on the performance of the algorithm. A maximal number can be set ex ante at a large number. However, the algorithm may have found a solution before this number is ever reached. Then extra computations may have to be performed even after the solution has been found. Balakrishnan and Jacob [42] have shown a moving average rule that can provide a good indication of convergence to a solution. More specifically, the GA process terminates if the average fitness of the best three strings of the current generation has increased by less than a threshold (namely convergence rate) as compared with the average fitness of the best three strings over three immediate previous generations.

To leverage possible problems of termination by either convergence or maximal number of generations alone, this research adopts a two-step stopping rule to incorporate both. A moving average rule is used for the first stopping check. The convergence rate is set at 0.1%. In practice, this value could be determined based on sensitivity analysis of trial examples according to the particular problem context. Then a maximal number of generations is specified as the criterion for the second stopping check. In our case, a number of 1000 is used. Similarly, this value could be determined based on trial runs in line with specific problems under study. These two steps complement each other. If the search is very difficult to converge (for example, in the case of a very tight convergence rate), the second stopping criterion helps avoid running the GA process infinitely. If can converge at the near-optimal solution with a few generations, then there is no need to run as many generations as the maximal number.

Moreover, in each generation the highest fitness value achieved so far and its corresponding string keep updated and stored. This makes sure that the best product portfolio solution found, not only from the final generation but also over all generations, is returned at convergence. Upon termination, the GA returns the product portfolio with the highest fitness (expected shared surplus) as well as the contained products in terms of specific configurations of attribute levels. All intermediate results of each generation (e.g., product portfolio candidates and their fitness values) and some descriptive statistics (e.g., numbers of crossovers and mutations, average population fitness, population standard deviation and status-quo of product portfolio solution) are recorded in the output report. Thus decision makers can track the progress of the GA or examine other feasible product portfolio solutions that are of high fitness values.

6. Application to notebook computer portfolio planning

6.1. Background

The proposed framework has been applied to the notebook computer portfolio planning problem for a world-leading computer manufacturing company. For illustrative simplicity, a set of key attributes and available attribute levels are listed in Table 1. Among them, “price” is treated as one of the attributes to be assumed by a product. Every notebook computer is thus described as a viable configuration of available attribute levels.

With regard to the class-member relationships defined in Fig. 1, notebook computer portfolios comprise a four-layer AND/OR tree structure, as shown in Fig. 4. The first layer is portfolios, each of which consists of one or more products. Each product consists of a few attributes, thus constituting the second layer. The fourth layer represents the value options for each attribute located at the third layer, indicating the

Table 1
List of attributes and their feasible levels for notebook computers

Attribute		Attribute levels		
a_k	Description	a^*_{kl}	Code	Description
a_1	Processor	a^*_{11}	A1-1	Pentium 2.4 GHz
		a^*_{12}	A1-2	Pentium 2.6 GHz
		a^*_{13}	A1-3	Pentium 2.8 GHz
		a^*_{14}	A1-4	Centrino 1.4 GHz
		a^*_{15}	A1-5	Centrino 1.5 GHz
		a^*_{16}	A1-6	Centrino 1.6 GHz
		a^*_{17}	A1-7	Centrino 1.7 GHz
		a^*_{18}	A1-8	Centrino 1.8 GHz
		a^*_{19}	A1-9	Centrino 2.0 GHz
a_2	Display	a^*_{21}	A2-1	12.1" TFT XGA
		a^*_{22}	A2-2	14.1" TFT SXGA
		a^*_{23}	A2-3	15.4" TFT XGA/UXGA
a_3	Memory	a^*_{31}	A3-1	128 MB DDR SDRAM
		a^*_{32}	A3-2	256 MB DDR SDRAM
		a^*_{33}	A3-3	512 MB DDR SDRAM
		a^*_{34}	A3-4	1 GB DDR SDRAM
a_4	Hard Disk	a^*_{41}	A4-1	40 GB
		a^*_{42}	A4-2	60 GB
		a^*_{43}	A4-3	80 GB
		a^*_{44}	A4-4	120 GB
a_5	Disk Drive	a^*_{51}	A5-1	CD-ROM
		a^*_{52}	A5-2	CD-RW
		a^*_{53}	A5-3	DVD/CD-RW Combo
a_6	Weight	a^*_{61}	A6-1	Low (below 2.0 KG with battery)
		a^*_{62}	A6-2	Moderate (2.0–2.8 KG with battery)
		a^*_{63}	A6-3	High (2.8 KG above with battery)
a_7	Battery Life	a^*_{71}	A7-1	Regular (around 6 hours)
		a^*_{72}	A7-2	Long (7.5 h above)
a_8	Software	a^*_{81}	A8-1	Multimedia package
		a^*_{82}	A8-2	Office package
a_9	Price	a^*_{91}	A9-1	Less than $800
		a^*_{92}	A9-2	$800–$1.3K
		a^*_{93}	A9-3	$1.3K–$1.8K
		a^*_{94}	A9-4	$1.8K–$2.5K
		a^*_{95}	A9-5	$2.5K above

instantiation of an attribute by one out of many levels. This tree structure not only reflects a generic variety structure [30] underlying notebook computer product offerings, but also establishes the searching space for the GA solver.

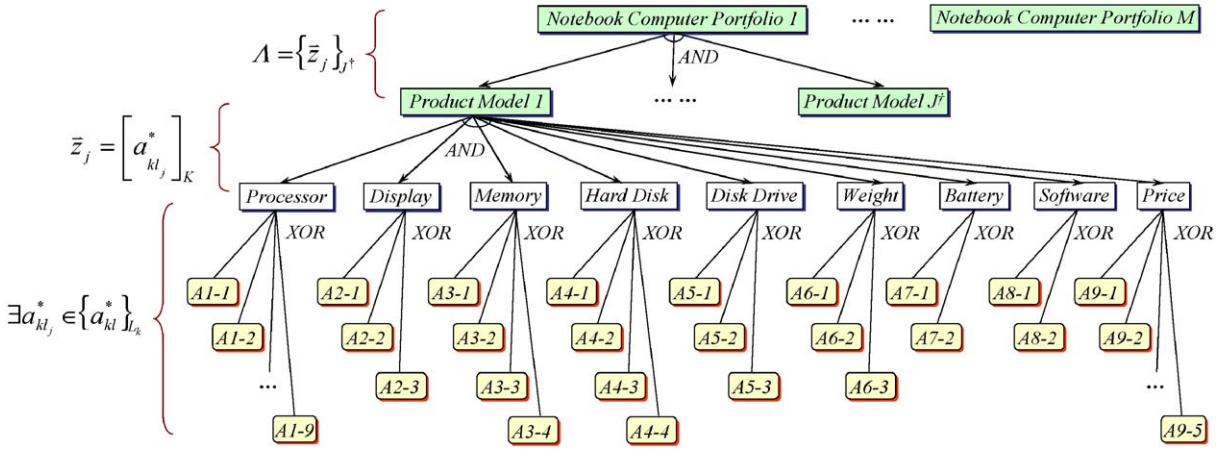

Fig. 4. Generic variety structure for notebook computer portfolios.

Based on a total number of 30 customers acting as the respondents, three segments are identified, i.e., s_1, s_2, and s_3. With conjoint analysis, the part-worth utilities for each respondent in one segment can be derived. Averaging the part-worth utility results of all respondents belonging to the same segment, a segment-level utility is obtained for each attribute level. Fig. 5 shows the part-worth utilities of three segments with respect to every attribute level. With assembly-to-order production, the company has identified and established standard routings as basic constructs of its process platform. Based on empirical studies, costing parameters and part-worth standard times for all attribute levels are determined.

6.2. HGA implementation

To determine a near-optimal notebook computer portfolio for the target three segments, the GA procedure is applied to search for a maximum of expected shared surplus among all attribute, product and portfolio alternatives. Assume that each portfolio may consist of a maximal number of $J^{\dagger} = 5$ products. Then a chromosome string comprises $9 \times 5 = 45$ genes. Each substring is as long as nine genes and represents a product that constitutes the portfolio.

Based on the economical analysis, some constraints are generated to ensure the profitability. The constraints are represented by "IF–THEN" rules, that is, "IF $x_1 = 9$, THEN $x_9 \neq 1$; IF $x_1 = 3$, THEN $x_9 \neq 2$; IF $x_3 = 4$, THEN $x_9 \neq 1$; IF $x_4 = 4$, THEN $x_9 \neq 1$". These constraints restrict the customers buying high performance notebook computer with too low price. Through constraint check, the invalid chromosomes are penalized by subtracting a large number and passed on for further evaluation. For every generation, a population size of $M = 100$ is maintained, meaning that only top 100 fit product portfolios are kept for reproduction.

In addition, it is not uncommon that in the notebook computer business most manufacturers directly order components and parts from their suppliers. It is thus reasonable to assume that the competitors of the company under our study offer the same product attributes and levels. As a result, the status-quo product alternatives in the current generation are used as the pool of competing products for the choice model in Eq. (3).

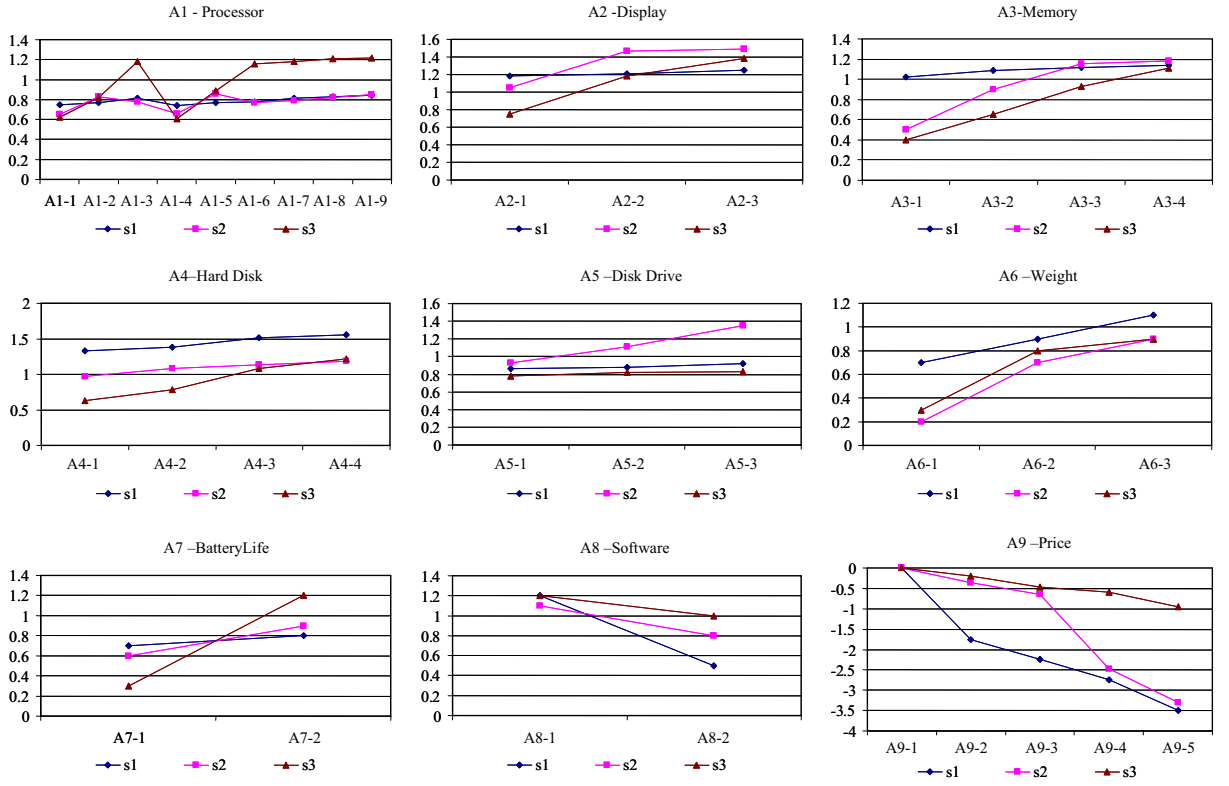

Fig. 5. Plots of segment-level part-worth utilities.

6.3. Results

The results of GA solution are presented in Fig. 6. As shown in Fig. 6, the fitness value keeps being improved generation by generation. Certain local optima (e.g., around 70 generations) are successfully overcome. The saturation period (240–280 generations) is quite short, indicating the GA search is efficient. This proves that the moving average rule is a reasonable convergence measure. It helps avoid such a possible problem that the GA procedure may run unnecessarily as long as 1000 generations. Upon termination at the 287th generation, the GA solver returns the near-optimal result, which achieves an expected shared surplus of 109, as shown in Table 2.

6.4. Performance evaluation

Fig. 7 compares the achievements, in terms of the normalized shared surplus, cost, and utility with choice probability, of 20 product portfolios in the 287th generation that returns the near-optimal solution. It is interesting to see that the peak of utility achievement (portfolio #6) does not contribute to producing the best fitness as its cost is estimated to be high. On the other hand, the minimum cost (portfolio #20) does not mean the best achievement of shared surplus as its utility performance is low. Also interesting to observe is that the worst fitness (portfolio #20) performs with neither the lowest utility achievement

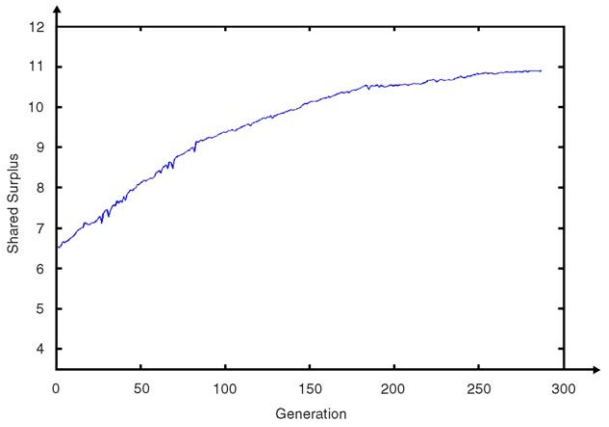

Fig. 6. Results of GA solution: shared surpluses among generations.

nor the highest cost figure. The best portfolio (#1) results from a higher level of utility and moderate cost performances.

Fig. 8 shows the performance of individual constituent products in terms of unbalance index for each portfolio in the 287th generation. It is noted that top five portfolios all contain a moderate number (2–3) of products, whereas those portfolios consisting of more products (e.g., portfolios #11, 16 and 19) seldom produce very good performances. This exactly illustrates the granularity tradeoff issue in PPP. In fact, too many products introduced in a portfolio may even bring about competition among themselves. On the other hand, none of the top 20 portfolios contains only one product. In practice, a single-product portfolio is not a desired case either, as it facilitates a limited coverage of diverse customer segments. Fig. 8 shows that three product portfolios are outstanding with respect to their shared surplus (portfolio # 1, 2, 3) with their normalized shared surplus of 1, 0.94, and 0.58, respectively. Among these portfolios, the portfolio 1 is the best with respect to its unbalance scores with unbalanced score of 0.2 and thus is selected as the final choice.

7. Sensitivity analysis

It is most important to keep the population diversity during the GA searching process. Low diversity may cause "inbreeding", thus weakening the exploratory capability. Many parameters can influence the population diversity. For example, an excessively high crossover rate will cause the solution to converge quickly before the optimum is found. On the other hand, a low crossover rate decreases the population diversity and results in long computation time. The mutation rate also influences the GA performance, as it determines the frequency of random search. Generally, a very low mutation rate is recommended to avoid that the GA process becomes a pure random search, which impairs the property of GA. The population size may be the most distinct factor influencing the population diversity. For a complex problem, large population size is preferred to ensure exploration in a large search space. In this section, we evaluate the performance of the HGA by means of sensitivity analysis. Based on varying parameter values, such as the population size, the crossover and mutation rates, HGA performance is examined for different problem sizes.

Table 2
Near-optimal solution of notebook computer portfolio

Product portfolio $\Lambda^{\dagger}$	Chromosome $\Lambda^{\dagger} = [6, 3, 3, 3, 1, 1, 1, 1; 9, 2, 1, 4, 2, 3, 2, 2, 2; 0, 0, 0, 0, 0, 0, 0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$			
Constituent products $\{\vec{z}_j^{\dagger}\}_{J^{\dagger}}$	Substring $\vec{z}_1^1 = [6, 3, 3, 3, 3, 1, 1, 1, 1]$		Substring $\vec{z}_2^1 = [9, 2, 1, 4, 2, 3, 2, 2, 2]$	
	$a_k^{\dagger}$	$a_{kl}^{\dagger}$	$a_k^{\dagger}$	$a_{kl}^{*\dagger}$
Attributes $\{a_k^{\dagger}\}_{(K+1)^{\dagger}}$	Processor	Centrino 1.6 GHz	Processor	Centrino 2.0 GHz
	Display	15.4" TFT XGA/UXGA	Display	14.1" TFT SXGA
	Memory	512 MB DDR SDRAM	Memory	128 MB DDR SDRAM
	Hard disk	80 GB	Hard Disk	1 GB DDR SDRAM
	Disk drive	DVD/CD-RW Combo	Disk Drive	CD-RW
Attributes levels $\{a_{kl}^{*\dagger}\}_{(K+1)^{\dagger}}$	Weight	Low (below 2.0 KG with battery)	Weight	High (2.8 KG above with battery)
	Battery life	Regular (around 6 h)	Battery Life	Long (7.5 h above)
	Software	Multimedia package	Software	Office package
	Price	Less than $800	Price	$800–$1.3K
Expected shared surplus $E[V^{\dagger}]$	109			
Unbalance index $\psi^{\dagger}$	0.2			

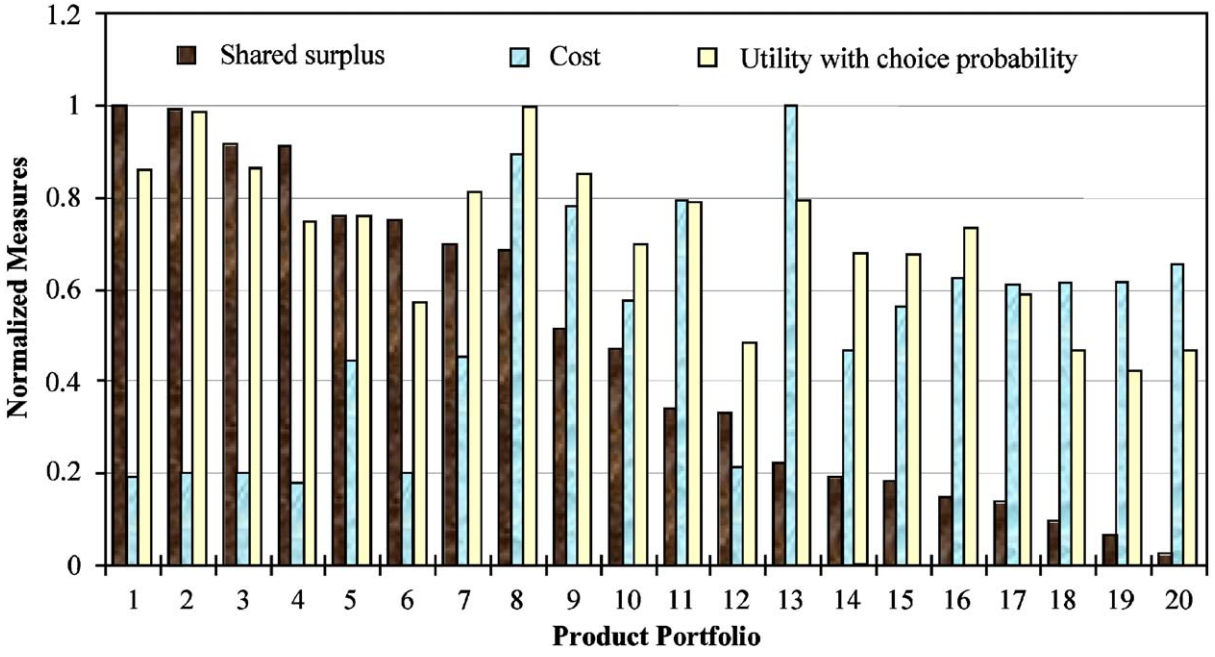

Fig. 7. Performance comparison of product portfolio population in the 287th generation.

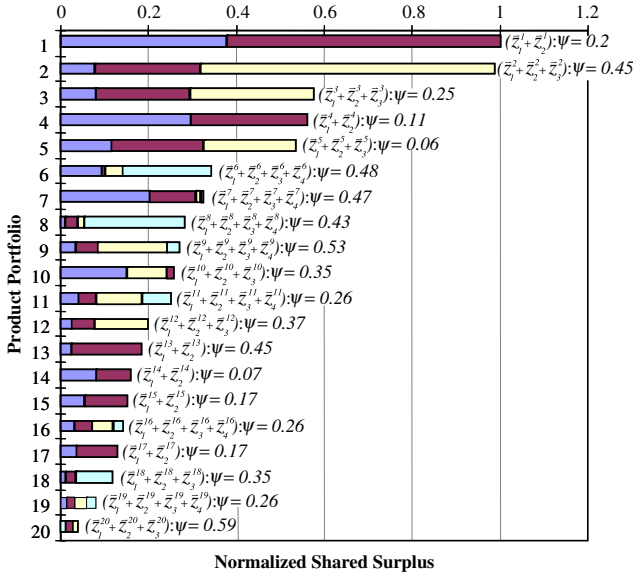

Fig. 8. Comparison of constituent products for each product portfolio produced in the 287th generation.

Table 3
Performance of different problem sizes and respective parameter values

Problem type	Number of attributes	Problem size	Parameter value and performance						Number of runs
			Population		Crossover		Mutation		
			Size	Ave_App (%)	P_c	Ave_App (%)	P_m	Ave_App (%)	
Simple	3	15	20	97.3	0.6	95.6	0.005	96.2	360
Moderate	6	30	50	96.7	0.7	96.3	0.01	95.8	360
Complex	9	45	100	94.2	0.8	97.1	0.01	94.1	360

7.1. Problem size

In accordance with different parameter values required for varying problem sizes, three cases are constructed to represent three different problem sizes for notebook computer portfolio specification. The first case represents a simple problem size, where three attributes are selected, including processor, memory, and weight that are of 9, 4, and 3 levels, respectively. The second case corresponds to a moderate problem size, consisting of six attributes, i.e., processor, memory, weight, hard disk, display, and battery life, which assume 9, 4, 3, 4, 3, and 2 attributes levels, respectively. The third case stands for a very complex problem size, in which all nine attributes and their possible levels are considered. Table 3 lists all three scenarios.

7.2. Experiment design

The proper parameter values for the population size, the crossover and mutation rates are recommended through sensitivity analysis. To setup the experiments, 4 values are considered for population size, namely 20, 50, 80, and 100. Likewise 3 values of crossover rate (0.6, 0.7, 0.8) and 3 values of mutation rate (0.005, 0.01, 0.03) are used. Therefore, sensitivity analysis experiment is constructed based on a $4 \times 3 \times 3$ full design. For more complex analysis, where more values are involved, other experiment design method, such as orthogonal design and factorial design, can be employed. The values of these parameters are selected based on the rule-of-thumb from most of GA applications—a crossover rate at least 0.6 and a very low mutation rate.

7.3. Parameter selection

The full design generates 36 scenarios. For each scenario, the GA runs for 10 times to collect the mean value of its performance. Thus, the parameter values for each problem size are recommended on the basis of 360 test runs. The average degree of approximation (Ave_App) associated with GA solutions is adopted as the performance indicator of each problem type. The best GA parameter values are recommended as shown in Table 3.

As illustrated in Table 3, a larger population size is required for a complex problem in order to keep the population diversity. A population of diverse products is necessary to guarantee thorough exploration in

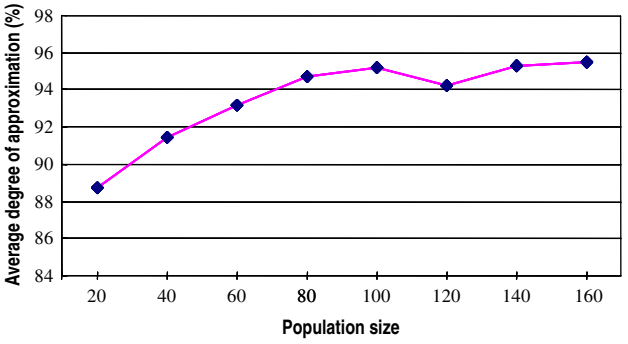

Fig. 9. Performance of different population sizes.

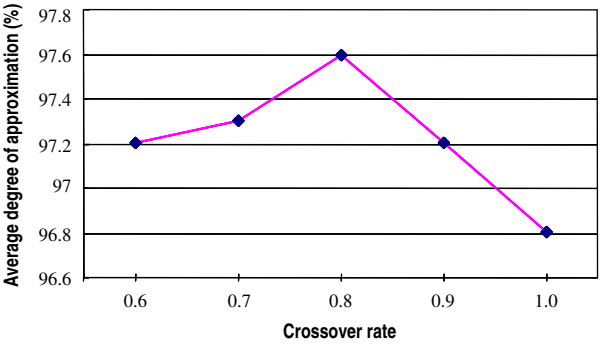

Fig. 10. Performance of different crossover rate values.

the search space so as to achieve a high degree of average approximation. The crossover rate (p_c) of 0.8 is recommended to encourage more chromosomes to exchange their promising parts and to generate the offspring with better performance. It also demonstrates the tendency that a higher crossover rate leads to better approximation. For complex problem, a higher mutation rate is recommended to avoid search's falling into local optimum. For the simple problem type, a lower mutation rate is recommended so that the search does not become a pure random search.

Fig. 9 shows the average degree of approximation for each population size based on an interval of 20 within the range [20,160] for the complex problem type. The crossover and mutation rates are set to 0.8 and 0.01, respectively. As illustrated in Fig. 9, too large a population size (160) may contribute to the improvement of performance to only a modest extent.

Fig. 10 shows the average degree of approximation for varying crossover rate based on an interval of 0.1 within the range [0.6, 1.0] for the complex problem type. The population size and mutation rate are set to 100 and 0.01, respectively. It suggests that too large a crossover rate may decrease the performance. This is consistent with previous findings from GA applications, that is, a large crossover rate may cause too many chromosomes to change, thus leading to premature.

8. Discussion

As witnessed in the case study, the strength of GA lies in the ability to carry out repeated runs without major changes of parameter values or defining different initial populations, thus improving the chance of finding an optimal or at least a near optimal solution. It is also possible to insert solutions obtained from other techniques into the initial population. Hence, rather than generating all the members of the initial population at random, the GA can use a prior knowledge about potential optima to arrange the initial population or improve on an existing solution that can perform as a kind of lower bound or benchmark for GA performance.

In the case study, we have tested the performance of the proposed heuristic GA on a small to medium sized problem. Although we do not know how well the heuristic GA would perform on large problems in an absolute sense, we believe, given the popularity of high performance computation, what important is not the computational efficiency of GA, but the quality of solutions returned by the GA procedure. For instance, even the most complex GA problem scenario among all testing runs in the case study and sensitivity analysis takes less than 30 s to solve using Matlab® and with a Pentium IV 2.4 MH PC. From the experience of using the product portfolio balance rule, it is obvious that the heuristic GA also provides high flexibility with regard to the final decision making of a product portfolio. For example, the decision maker may be provided with quite a number of solutions using similar high fitness values that are of his expectations. In this way, the decision maker can instrument additional fitness criteria as appropriate for selecting the best product portfolio.

If interactions between attributes are to be considered, the additive main-effect utility model could be easily extended to include interaction terms without introducing additional decision variables. The interaction terms merely affect a string's fitness evaluation by taking into account the associated part-worth utilities for preference evaluation. A possible means to deal with technologically infeasible configurations of attribute levels is to incorporate related interaction terms into the cost functions and to impose certain penalties on those high cost-carrying attribute levels. This may be implemented as a hybrid constraint handling strategy during the GA procedure.

9. Conclusion

A heuristic genetic algorithm is developed and applied to solve the combinatorial optimization problem involved in PPP. The study indicates that the GA works efficiently in searching for near-optimal product portfolio solutions. Although the model is used to solve a seller's problem of introducing a new product portfolio with the objective of maximal shared surplus, the proposed framework could easily be adjusted to handle such complex problems as maximizing share-of-choices and extending an existing product portfolio by allowing for already existing items to be owned by the seller. This is supported by the flexibility of the GA procedure that merely uses objective function information, and therefore is capable of accommodating different fitness criteria without any substantial modification of the algorithm.

Acknowledgements

This research is supported by Singapore NTU-Gintic Collaborative Research Project (U01-A-130B).The authors would like to express their sincere thanks to Professors Mitchell M. Tseng, Martin Helander and

Halimahtun M. Khalid, and colleagues of Global Manufacturing & Logistics Forum at Nanyang for their valuable advices.

References

[1] Ho TH, Tang CS. Product variety management: research advances. Boston/Dordrecht/London: Kluwer Academic Publishers; 1998.
[2] Child P, Diederichs R, Sanders FH, Wisniowski S. SMR forum: the management of complexity. Sloan Management Review 1991;33(1):73–80.
[3] Wortmann JC, Muntslag DR, Timmermans PJM. Customer driven manufacturing. London: Chapman & Hall; 1997.
[4] Tseng MM, Jiao J. Design for mass customization. CIRP Annals 1996;45(1):153–6.
[5] Huffman C, Kahn B. Variety for sale: mass customization or mass confusion? Journal of Retailing 1998;74(4)491–513.
[6] Pine BJ, Victor B, Boynton AC. Making mass customization work. Harvard Business Review 1993;71(5):108–21.
[7] Henderson BD. The product portfolio. Boston, MA: Boston Consulting Group; 1970.
[8] Warren AA. Optimal control of the product portfolio. Ph.D. thesis, The University of Texas at Austin; 1983. 335p.
[9] Cooper R, Edgett S, Kleinschmidt E. Portfolio management for new product development: results of an industry practices study. R & D Management 2001;31(4):361–81.
[10] Kaul A, Rao VR. Research for product positioning and design decisions: an integrative review. International Journal of Research in Marketing 1995;12(4):293–320.
[11] Dobson G, Kalish S. Positioning and pricing a product line. Marketing Science 1988;7(2):107–25.
[12] Kohli R, Sukumar R. Heuristics for product-line design using conjoint analysis. Management Science 1990;36(12): 1464–78.
[13] Nair SK, Thakur LS, Wen K. Near optimal solutions for product line design and selection: beam search heuristics. Management Science 1995;41(5):767–85.
[14] Green PE, Krieger AM. Models and heuristics for product line selection. Marketing Science 1985;4(1):1–19.
[15] Krishnan V, Ulrich K. Product development decisions: a review of the literature. Management Science 2001;47(1):1–21.
[16] Nemhauser GL, Wolsey LA. Integer and combinatorial optimization. New York: Wiley; 1988.
[17] Sait SM, Youssef H. Iterative computer algorithms with applications in engineering: solving combinatorial optimization problems. Los Alamitos, CA: IEEE Computer Society Press; 1999.
[18] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP-completeness. New York: Freeman Publisher; 1979.
[19] Kamrani AK, Gonzalez R. A genetic algorithm-based solution methodology for modular design. Journal of Intelligent Manufacturing 2003;14(6):599–616.
[20] Houck CR, Joines JA, Kay MG. Comparison of genetic algorithms, random restart, and two-opt switching for solving large location-allocation problems. Computers & Operations Research 1996;23(6):587–96.
[21] Kirkpatrick S, Gelatt Jr. CD, Vecchi MP. Optimization by simulated annealing. IBM Research Report, RC 9355; 1982.
[22] Glover F. A user's guide to tabu search. Annals of Operations Research 1993;41(3):3–28.
[23] Holland JH. Adaptation in natural and artificial systems. Cambridge, MA: MIT Press; 1992.
[24] Reeves CR. Modern heuristic techniques for combinatorial problems. Oxford: Blackwell Publisher; 1993.
[25] Aarts E, Lenstra JK. Local search in combinatorial optimization. Chichester: Wiley; 1997.
[26] Alexouda G. An evolutionary algorithm approach to the share of choices problem in the product line design. Computers & Operations Research 2004;31(13):2215–29.
[27] Renders J-M, Flasse S. Hybrid methods using genetic algorithms for global optimization. IEEE Transactions on Systems, Man and Cybernetics, Part B 1996;26(2):243–58.
[28] Chu PC, Beasley JE. A genetic algorithm for the generalized assignment problem. Computers & Operations Research 1997;24(1):17–23.
[29] Jiao J, Zhang Y. Product portfolio identification based on association rule mining. Computer-Aided Design 2005;37(2):149–72.
[30] Du X, Jiao J, Tseng MM. Architecture of product family: fundamentals and methodology. Concurrent Engineering: Research and Application 2001;9(4):309–25.

[31] Jiao J, Zhang Y, Wang, Y. Product portfolio planning with customer-engineering interaction. IIE Transactions 2005;37(9):801–14.
[32] Moore WL, Louviere JJ, Verma R. Using conjoint analysis to help design product platforms. Journal of Product Innovation Management 1999;16(1):27–39.
[33] Ben-Akiva M, Lerman S. Discrete choice analysis: theory and application to travel demand. Cambridge: The MIT Press; 1985.
[34] Jiao J, Tseng MM. Customizability analysis in design for mass customization. Computer-Aided Design 2004;36(8): 745–57.
[35] Jiao J, Tseng MM. A pragmatic approach to product costing based on standard time estimation. International Journal of Operations & Production Management 1999;19(7):738–55.
[36] Li H, Azarm S. An approach for product line design selection under uncertainty and competition. Transactions of the ASME, Journal of Mechanical Design 2002;124(3):385–92.
[37] Tarasewich P, Nair SK. Designer-moderated product design. IEEE Transactions on Engineering Management 2001;48(2):175–88.
[38] Steiner WJ, Hruschka H. A probabilistic one-step approach to the optimal product line design problem using conjoint and cost data, Review of Marketing Science Working Papers, 1(4): Working Paper 4. http://www.bepress.com/roms/vol1/iss4/paper4; 2002.
[39] Gen M, Cheng R. Genetic algorithm and engineering optimization. New York: Wiley; 2000.
[40] Obitko M. Introduction to genetic algorithms. http://labe.felk.cvut.cz/∼obitko/ga/; 2003.
[41] Jiao J, Zhang L, Pokharel S. Planning process platforms for variety synchronization from design to production. IEEE Transactions on Engineering Management, 2005;51 (Fortcoming).
[42] Balakrishnan PVS, Jacob VS. Genetic algorithms for product design. Management Science 1996;42(1):1105–17.