Chapter VIII Real World Application Examples

In the previous chapters we have introduced the methods of OO for single-objective unconstrained optimization (Chapter II), multi-objective optimization (Chapter IV), constrained optimization (Chapter V), and for simple and good enough strategies (Chapter VI). The purpose of this chapter is to demonstrate how these methods can be applied to real world problems. We consider four problems: three real world applications and a benchmark problem in team decision theory. In Section 1, we consider a scheduling problem for apparel manufacturing problem (Lee 1997, Bouhia 2004). We demonstrate how the OO method introduced in Chapter II helps to save the computing time by 2000 folds comparing with the brute force method in this problem. In Section 2, we consider a turbine blade manufacturing process optimization problem (Yang 1998 and Yang et al. 1997). The objective function in this problem is not stochastic simulation but deterministic complex calculation. In Section II.6 we have mentioned that OO can also be applied in this type of problem. We justify this through this application. Furthermore, we show how to obtain better estimate of the size of the selected set through appropriate interpolation when the noise level is different from the three values considered in the UAP table in Section II.5. In Section 3, we consider a remanufacturing system performance optimization problem (Song et al. 2005a, 2005b). There are constraints in this problem. We first demonstrate how to use COO introduced in Chapter V to deal with this problem directly. Then to better describe the requirements in engineering practice, we reformulate the problem to a two-objective optimization problem, and use VOO introduced in Chapter IV to solve the problem. We also demonstrate how we can incorporate the problem information to obtain less conservative estimate of the size of the selected set when the noise level is different from the values considered in the VOO-UAP table in Section IV.2. In Section 4, we consider the Witsenhausen problem, which is a famous problem in team decision theory and has not been solved for nearly forty years since the problem was first proposed by Witsenhausen in 1968 (Witsenhausen 1968). This is a strategy optimization problem, which has an extremely large search space. We demonstrate how OO helps to discover properties of the good strategies, thus successively narrows down the search space, and substantially improves the strategy that has been obtained before the application of

OO to this problem. Based on the properties thus discovered, Lee et al. obtained the best-so-far strategy for this problem (Lee et al. 2001). We also demonstrate how to use the OBDD introduced in Chapter VI to quantify the complexity of the milestone solutions to this famous problem. Combining OO with OBDD, we demonstrate how to search for a good and simple strategy in this problem.

As in all cases, we present only the salient features of the problem, the methodology used, together with enough results to show the improvement obtained and/or savings achieved. Readers can consult the original references for minute details.

1 Scheduling problem for apparel manufacturing

In this section, we apply the OO method in Chapter II to solve a scheduling problem for apparel manufacturing which are subject to the whims of fashion. The manufacturing system is characterized by the co-existence of the two production lines, i.e., one with long lead time and low cost, the other a flexible one with short lead time and high cost. The goal is to decide: (1) the fraction of the total production capacity to be allocated to each individual line, and (2) the production schedules so as to maximize the overall profit and yet avoid stock shortage. The problem is difficult and it is prohibitive to search for the best solution in view of the tremendous computing budgets involved. Using ordinal optimization introduced in Chapter II, we have obtained very encouraging results – not only have we achieved a high proportion of "good enough" designs but also tight profit margins compared with a pre-calculated upper bound. There is also a saving of at least 2000 folds of the computation time if brute-force simulations were otherwise conducted (Lee 1997). The rest of this section is organized as follows. In Section 1.1, we introduce the background of this problem. A detailed problem formulation is presented in Section 1.2, together with a discussion on the challenges to solve the problem. Section 1.3 introduces the application of ordinal optimization in this problem, including how to randomly sample designs and how to construct a crude model, which is computationally fast, evaluating the performance of the designs roughly, and useful to compare the designs. The application procedure of OO in this scheduling problem is also summarized. Section 1.4 gives several experimental results to show how ordinal optimization helps to save the computing budgets by 2000 folds, and the performance of the good enough design found by OO is close to the upper bound of the performance of the optimal design. We make a brief conclusion in Section 1.5.

1.1 Motivation

In the past thirty years, technological advancements, international competitions and new market dynamics have had major impacts on the North American apparel manufacturing industry. The conventional analysis of the apparel industry predicts that the apparel industry will collapse rapidly and migrate to nations with low labor costs. Although apparel industries still exist in the United States nowadays, intense competition encourages management to develop new production and supply methodology in order to remain competitive (see (Harvard 1995)). One key issue involved is the allocation of scarce production resources over competing demands. Before we introduce the detailed scheduling problem from the manufacturer's viewpoint in the next subsection, let us first have a big picture of the entire apparel manufacturing system. A typical apparel manufacturing system is shown in Fig. 8.1. The retailers receive customer demands, which is usually random and sometimes have seasonal variations. For example, the demands on swimsuits are high in spring but low in fall, while the demands on ties are high on Father's day and Christmas but low in the rest of the year. It is an important strategic requirement to satisfy customer demands. Failing to do so can result not only in lost profits due to reduced sales, but also the lost of future market share. In order to deal with the randomness in the customers' demand, the retailers usually maintain a small inventory of the apparel. In the apparel market nowadays, customers demand the variety of products. Thus the retailers have responded to their customers' wishes by maintaining a small inventory of many different styles of apparel and demanding rapid (usually weekly) replenishment of store inventory from the apparel manufacturers.

From the manufacturer's viewpoint, to fulfill the random replenishment orders from the retailers, an inventory of finished goods is established.

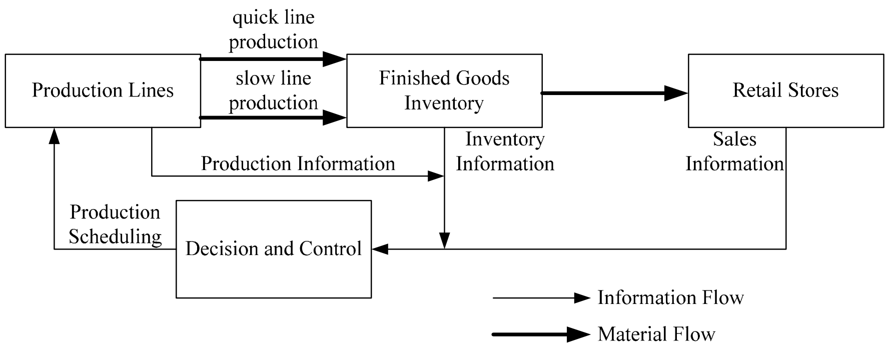

Fig. 8.1. Material and information flow chart of apparel manufacturing systems

This inventory is expensive to be maintained and should be no higher than necessary to meet demands. There are generally two ways to build up a required inventory level, i.e., using quick or regular production lines. In a regular production line, work flows from worker to worker in bundles with work buffers between each work station. The Work-In-Process (WIP) in each buffer is so large that it takes 20 to 25 days for a garment to pass through all operations, even though only 10 to 20 minutes of direct labor content is actually required to assemble the garment. Therefore, this kind of line has a long lead time, which is defined as the time from the receipt of the order from the retailers to the time the products are delivered to the finished goods inventory, i.e., the order is fulfilled. In a quick line, a small group of workers are cross trained to perform several sewing operations. The group of workers performs all of the sewing assembly operations on the apparel item. Workers move from one work station to another thereby minimizing the WIP in the production line. The cycle time in the quick line is less than the regular line; however, since a cross-trained worker is more expensive than a regular worker, and workers are generally less productive, on average, at several operations than they are at a single operation, the cost of the quick line is higher. The manufacturer is responsible for making decisions on how to manage future production on different production lines in order to maximize the overall manufacturing profit. We discuss more details of the manufacturer's scheduling problem in the next subsection.

1.2 Problem formulation

The manufacturer's decision and control should determine (1) the fraction of the total production capacity, γ, to be allocated to each production line; and (2) the scheduling strategy, α, that decides when to work on which demand on each production line. These decisions and controls are made weekly, because the replenishment requirement from the retailers is weekly-made, and this allows the manufacturer to collect new information (past production schedules, inventory and demand information) before making decisions and having control of the future. The goal is to maximize the overall manufacturing profit, which is the total revenue minus the material cost, the production cost (i.e., the cut, make, and trim cost and the shipping cost) and the holding cost for both the items in the finished goods inventory and in the WIP. The overall profit is affected by the demand, the production facilities, and the inventory dynamics. In the parlance of control theory, this is a full blown stochastic feedback optimal control problem. We will discuss these issues separately in the rest of this subsection. Before that, we need to introduce the concept of Stock Keeping Unit (SKU), which is used to describe different types of items in the demand and the production. A SKU is a particular style, fabric and size of an apparel item. A typical jeans manufacturer may make 10,000 to 30,000 distinct SKUs of jeans in a year. In a given season of the year, the number of SKUs manufactured may still be as high as 10,000. In our problem, suppose there are M SKUs in all.

1.2.1 Demand models

We assume that demand is weekly-made and there is no back-ordering, i.e., the retailers will be given whatever is left in the warehouse when the demand level is greater than the inventory level for each SKU. Assume the demand of SKU i at time t, $d_i(t)$, is $\max(\zeta(t),0)$, where $\zeta(t)$ is a Gaussian random variable with mean $\mu_i(t)$ and standard deviation $\sigma_i(t)$. If we neglect the truncated effect, average demand of SKU i at time t is equal to $\mu_i(t)$. When we consider seasonal effects, $\mu_i(t)$ will be a periodic function. Coefficient of variation of SKU i, $Cv_i(t)$, is defined as standard deviation divided by mean, i.e.,

$$Cv_i(t) = \frac{\sigma_i(t)}{\mu_i(t)}. \tag{8.1}$$

We assume that the coefficient of variation, $Cv_i(t)$, is a constant, and we will use Cv_i from now on.

There are usually three types of demands in apparel manufacturing system.

- Flat Demand

There is no consideration on seasonal effects. The average demand is constant throughout the year, i.e., $\mu_i(t)$ = constant. Fig. 8.2 shows the relationship between the average demand and real demand under different Cv's.

- Seasonal – Sine Demand

There is a consideration on seasonal effects and the changes of the average demand are smooth, i.e., $\mu_i(t) = A_i + B_i \sin(2\pi t/T)$ where T is the period of seasonal effects, and A_i, B_i are the amplitudes of the function, $A_i > B_i$. Fig. 8.3 shows the relation between the mean and the real demand under different Cv's.

- Seasonal – Impulse demand

Sometimes, there are peak demands caused by promotions or special holidays or both. A sudden jump in sales is often observed at the beginning of a peak sales period. The peak sales are often planned to be roughly equally spaced along a year and last for a short time (several weeks) compared with the regular selling period. The seasonal demand can be modeled by a two-level demand function, which is called impulse demand. Fig. 8.4 shows the relationship between the mean and the real demands under different *Cv*'s.

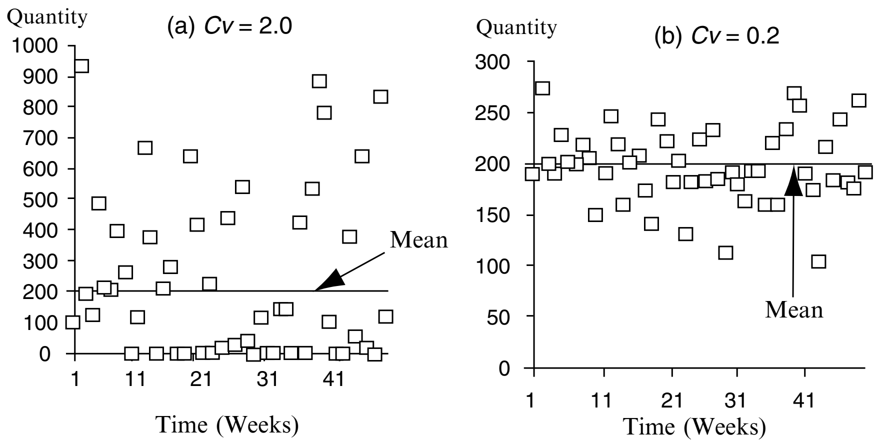

Fig. 8.2. Actual demand vs. mean for flat demand case when (a) *Cv* = 2.0 and (b) *Cv* = 0.2

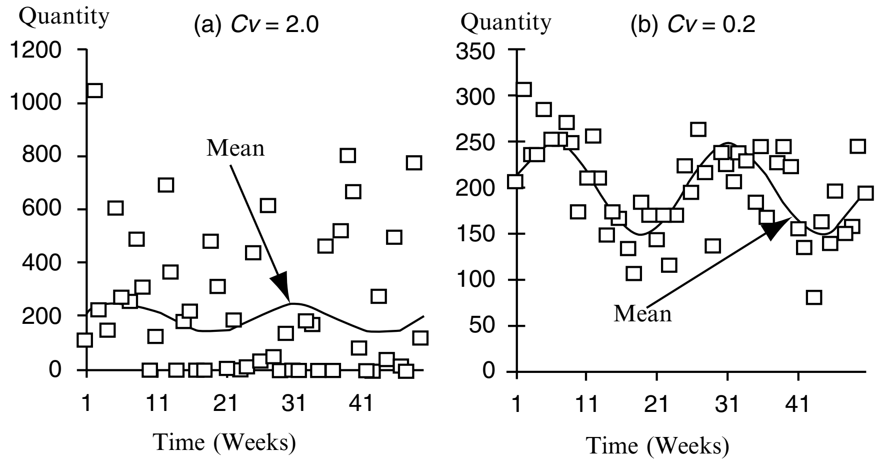

Fig. 8.3. Actual demand vs. mean for seasonal – sine demand case when (a) *Cv* = 2.0 and (b) *Cv* = 0.2

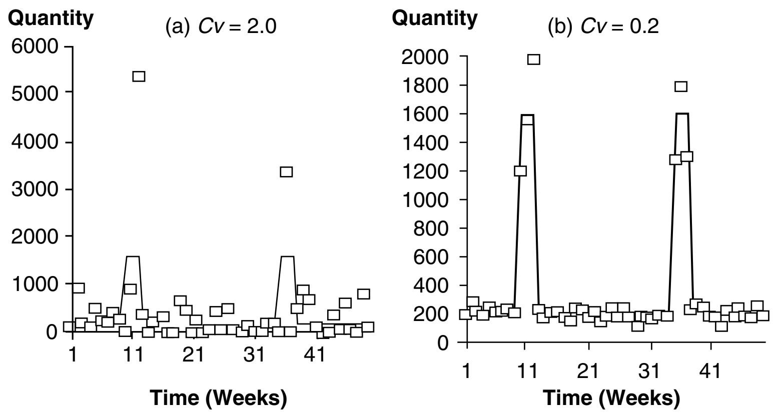

Fig. 8.4. Actual demand vs. mean for seasonal – impulse demand case when (a) $Cv = 2.0$ and (b) $Cv = 0.2$

1.2.2 Production facilities

As aforementioned, we consider two different kinds of production lines, a quick line and a regular (slow) line; the lead times of which are denoted respectively as L_q and L_s. Both L_q and L_s are assumed to be known and constant. By definition, $L_q < L_s$. The total production capacity is generally limited by the availability of resources such as equipments. In the apparel industry, production capacity is generally determined by available labor. In this problem, we assume the total capacity CP is equal to the weekly average demand over all SKUs. Regularly, there are 5 working days a week, and we allow one day overtime, therefore,

$$\text{Maximum capacity} = CP_{\max} = 1.2 \times CP.$$

As for the minimum capacity, it is clear that it should be at least greater than zero, but in most situations, it cannot vary greatly week to week. A reasonable assumption is we have to work at least 4 days a week. Therefore,

$$\text{Minimum capacity} = CP_{\min} = 0.8 \times CP.$$

The ratio of the quick line capacity to the total capacity, γ, is a constant to be determined. Therefore,

$$\text{Maximum capacity of the quick line} = u_{1\max} = \gamma \times CP_{\max},$$
$$\text{Minimum capacity of the quick line} = u_{1\min} = \gamma \times CP_{\min},$$

Maximum capacity of the regular line = $u_{2\max} = (1-\gamma)\times CP_{\max}$,
Minimum capacity of the regular line = $u_{2\min} = (1-\gamma)\times CP_{\min}$.

The production schedules of each week should be chosen within these limits. Let $u_{i1}(t)$ be the amount of SKU i to be scheduled on the quick line at time t and $u_{i2}(t)$ be the amount of SKU i to be scheduled on the regular line at time t. Therefore,

$$u_{j\max} \geq \sum_{i=1}^{M} u_{ij}(t) \geq u_{j\min} \text{ for } j = 1,2 .$$

Please note that the capacity constraint is on the amount of SKUs scheduled on both lines each week, not on the work-in-process. The reason is that, in an apparel manufacturing system, the time to directly produce the items is comparatively smaller than the waiting time, which is also the main cause of the lead time. We assume if the capacity constraint is satisfied when scheduling the SKUs, then it is always possible to manage the workers and the machines to finish the scheduled SKUs within the lead time of that production line, although the more SKUs are scheduled, the higher the WIP will be, and this will increase the holding cost for the WIP.

1.2.3 Inventory dynamic

There is a weekly replenishment from the finished goods inventory to the retail stores. Let $I_i(t)$ be the total inventory of SKU i at time t, and $W_i(t)$ be the total work-in-process (WIP) inventory of SKU i at time t. Then $I_i(1)$ is the initial inventory of SKU i, and $I_i(t+1)$ should be equal to the left inventory level in the last week after satisfying the demand, i.e., $\max(I_i(t)-d_i(t),0)$, plus the SKUs that are produced during the last week by both production lines, i.e., $\sum_{j=1}^{2} u_{ij}\left(t-L_j+1\right)$. So we have

$$I_i(t+1) = \max\left(I_i(t) - d_i(t), 0\right) + \sum_{j=1}^{2} u_{ij}\left(t - L_j + 1\right), \forall i = 1,\ldots,M .$$

The WIP of week t will be the sum of all the SKUs that are still in production, i.e.,

$$W_i(t) = \sum_{j=1}^{2} \sum_{k=t-L_j+1}^{t} u_{ij}(k) .$$

1.2.4 Summary

For each product, let C_{m} be the material cost, C_{L_j} the production cost if line j is used, and P_{S} the sales price. Let also C_{I} be the holding cost for the finished goods inventory and WIP per week and per product. For a given γ and α, $J_{\mathrm{total}}(\alpha,\gamma)$ denotes the total manufacturing profit gained from week t=1 to week t=Π, which is calculated as follows.

$$\begin{aligned}
J_{\mathrm{total}}(\alpha,\gamma) = &\sum_{i=1}^{M}\sum_{t=1}^{\Pi} P_S \min\left(I_i(t), d_i(t)\right) && \text{sales price} \times \text{sales} \\
&-\sum_{i=1}^{M}\sum_{t=1}^{\Pi}\sum_{j=1}^{2} C_{\mathrm{m}} u_{ij}(t) && \text{material cost} \times \text{production} \\
&-\sum_{i=1}^{M}\sum_{t=1}^{\Pi}\sum_{j=1}^{2} C_{L_j} u_{ij}(t) && \text{production cost} \times \text{production} \\
&-\sum_{i=1}^{M}\sum_{t=1}^{\Pi} C_I I_i(t) && \text{inventory cost} \times \text{inventory} \\
&-\sum_{i=1}^{M}\sum_{t=1}^{\Pi} C_I W_i(t) && \text{inventory cost} \times \text{WIP.}
\end{aligned}$$

The average weekly manufacturing profit, $J(\alpha,\gamma)$ is given by $J_{\mathrm{total}}(\alpha,\gamma)$ divided by Π, i.e.,

$$J(\alpha,\gamma) = \frac{1}{\Pi} J_{\mathrm{total}}(\alpha,\gamma) \tag{8.2}$$

Since the demand is random, the objective function of the scheduling problem is then to find γ and α in order to maximize the expected total manufacturing profit, i.e.,

$$\max_{\alpha\in\Phi,\gamma\in[0,1]} E\{J(\alpha,\gamma)\},$$

where Φ is the collection of all possible scheduling policies α.

There exist four nearly insurmountable challenges in this problem.

- First, in the apparel manufacturing system, different sizes, colors, or fashions of shirts are considered as different ***stock-keeping units***

(SKUs). There may be over ten thousands different SKUs in the system. The demand of each SKU varies weekly and exhibits seasonal trends.

- Second, since the exact demand is unknown in advance, in order to estimate precisely the expected profit of each strategy, one needs to perform numerous time-consuming and expensive Monte-Carlo simulations.
- Third, the number of applicable strategies is equal to the size of the possible production schedules raised to the power of the size of the information space. It is clear that this can be very large even for a moderately-sized problem.
- Fourth, since the neighborhood structure in the strategy space is not known and the performance value function cannot be explicitly represented in terms of strategy, the calculus and gradient decent algorithm cannot be applied.

Because of these difficulties, if we want to get the optimal solution to this problem, brute-force simulation, or large state-space dynamic programming is unavoidable. In practice where there are many different SKUs, it is computationally infeasible to find the global optimum. In the next subsection, we will apply OO to solve this problem and provide results which are not only good but also quantifiable.

1.3 Application of ordinal optimization

Since the number of SKUs is very large, the number of applicable strategies is an astronomically large number and Monte Carlo simulation is needed to evaluate the performance value of each strategy. Therefore, searching for the best solution is prohibitive in view of the tremendous computing budgets involved. As mentioned in Chapter II, if we do not insist on getting the optimal design, i.e., we soften our goal by having a high probability of getting any good enough design, the problem will become more approachable. When the goal is softened, we can tolerate imprecise performance estimates because we can have high confidence in obtaining a "good enough" design from a selected set. In this way the difficulties of the original problem can be overcome.

To apply ordinal optimization in this scheduling problem, two important questions must be answered: How can we randomly sample designs from the design space? What is the crude model that is computationally fast and can supply rough performance estimate? We discuss these two questions in turn.

1.3.1 Random sampling of designs

Each design is defined as (α, γ), where α is production schedule and γ is the ratio of the quick line capacity to the total capacity. Although samples of γ can be easily generated by a uniform random number generator, the random sampling of the production schedule α is not so straightforward. By definition, the production schedule should satisfy all the capacity constraints each week and make the level of the finished goods inventory track the demand in an appropriate way, so that most of the demands can be fulfilled and the inventory holding cost is reasonable. If we do not utilize the above information, but uniformly randomly sample production schedules that satisfy the capacity constraints, there is no reason to believe these schedules can track the demand and reduce the inventory holding cost appropriately. We should incorporate the above information in the random sampling of production schedules.

Suppose there is no uncertainty in the demand process $d(t)$, i.e., $d(t)=E[d(t)]$, and we can take $E[d(t)]$ to be a deterministic process, then we can arrange the production schedules to track $E[d(t)]$ as best as we can (see Fig. 8.5 (a)). This can be solved, in principle, by using well-known control theory tools such as dynamic programming, or other ad hoc heuristic methods, if the size is too large. However, as shown in Fig. 8.5(b), if the

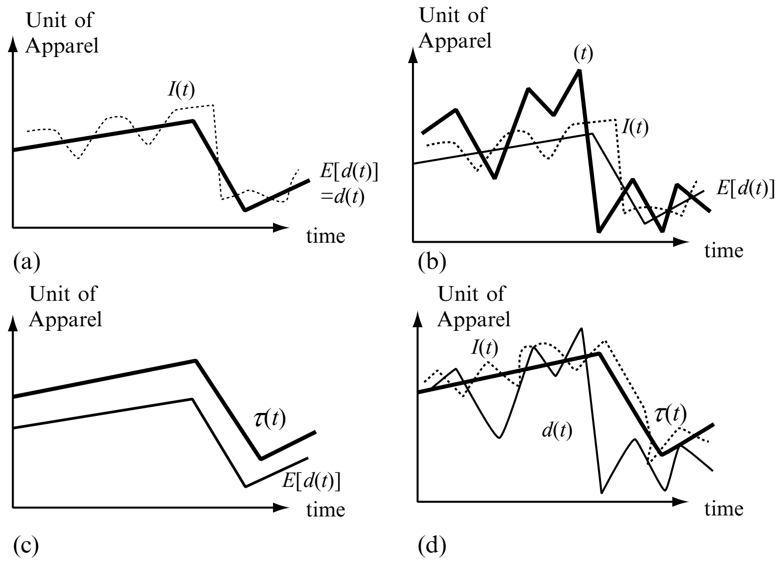

Fig. 8.5. A graphical illustration of how to select the scheduling strategy

demand process $d(t)$ is a random process, it is clear that tracking $E[d(t)]$ alone will not be satisfactory (in Fig. 8.5(b), we can see that the inventory $I(t)$ is too low to guarantee sales). Thus, we introduce another process to play the role of a deterministic process from which we can plan our scheduling strategy. This new process is called the target level, denoted as $\tau(t)$, which is used to replace what we have to, but cannot, track, i.e., $d(t)$. This is shown in Fig. 8.5(c). Notice that $\tau(t)$ is not a random process. Now we can solve a control problem to determine $u(t)$ to follow $\tau(t)$ as best as we can. $u(t)$ will be the production schedules. Therefore, we first find the target level of each SKU, then generate the production schedules that can track the target level. This is shown in Fig. 8.5(d). The remaining problem is to find a method to generate appropriate target levels and also a production schedule that will track the target level.

We must have higher inventory level when we have higher average demand, and also, when we have higher uncertainties, we must increase the inventory level in order to guarantee no shortage. So, a reasonable and simple way to generate the target level for each SKU is to let the target level of SKU i be proportional to the mean value and the standard deviation of the demand in the future, i.e., $\tau_i(t)=(a_1+a_2C_{vi})\mu_i(t+a_3)$, where $\mu_i(t)$ is the mean of the demand of SKU i at time t while Cvi is the coefficient of variation of SKU i, and a_1, a_2, and a_3 are constants constituting the design parameters and are randomly generated and used for all SKUs.

When the target level is given, we should arrange the production schedules so that the inventory level will be equal to the target inventory level by the time the SKUs exit the production lines. When the production capacity is not enough, the capacity will be allocated "fairly" among all the SKUs so that after the allocation the ratio of the inventory level to the target level is the same for each SKU. This algorithm was first proposed by (Bo et al. 1994) and modified by L. H. Lee (Lee 1997) to the cases when there are multiple SKUs, multiple production lines, and limited production capacity. Although this target tracking strategy does not guarantee the optimum, from our experimental results in Section 1.4, we will observe that it is not far from the optimum.

1.3.2 Crude model

The difficulty to accurately evaluate the performance of a design (α,γ) by simulation is due to the large number of SKUs, the long simulation in each replication, and the large number of replications in total. To obtain a crude model, which is computationally fast and only need to supply a rough performance estimate, we can do the following three relaxations. (1) As aforementioned, there might be 10,000 or even 30,000 different SKUs.

Instead of tracking the dynamics for each SKU during simulation, which will cost a lot of memory space and time, we aggregate the SKUs by the coefficient of variation. The mean of the demand of the aggregated SKU is equal to the sum of the means of the SKUs with similar *Cv*, and the *Cv* of the aggregated SKU will be equal to that *Cv*. In fact, if the demands of the SKUs are all positively correlated, it would be clear that this aggregation is appropriate. The reason is that, if $X_1, X_2, \ldots X_N$ are random variables with the same *Cv* and the correlation between X_i and X_j is equal to 1, $Y = \sum_{i=1}^{N} X_i$ will be a random variable with the same *Cv* and the mean demand is $\sum_{i=1}^{N} E\left[X_i\right]$. For other cases (say the demands of all the SKUs are independent, experiments have been done to justify this aggregation method (Lee 1997)). After the aggregation, there are usually no more than 100 SKUs and sometimes no more than 10 SKUs, which saves a lot of simulations. (2) Instead of simulating the system for several hundreds or thousands weeks, we can use a short simulation of only 100 weeks. (3) We can use a small number of replications (even only one replication).

In this way, we obtain a crude model. Although the performance estimate might be very different from the true performance values, the observed good enough designs set will nevertheless contain a lot of truly good enough designs.

Let us summarize the application procedure of OO in this scheduling problem (Box 8.1).

Box 8.1. Application procedure of OO in the scheduling problem of apparel manufacturing system

Step 1: Randomly generate *N* target levels as described in section 1.3.1.

Step 2: For each target level, randomly generate the capacity allocations between the two production lines. Then use the target tracking strategy to determine the production schedules of all the SKUs. The capacity allocation together with the production schedule is a design (α, γ).

Step 3: Aggregate the SKUs by the coefficient of variation. Then use the crude model to roughly estimate the performance of the designs.

Step 4: Estimate the observation noise level and the problem type.

Step 5: The user defines the size of good enough set *g*, and the required alignment level *k*.

Step 6: Use the UAP table in Section II.5 to calculate the size of the selected set *S*.

Step 7: The OO theory ensures that there are at least *k* truly good enough designs in the observed top-*s* designs with high probability.

1.4 Experimental results

In this subsection, we will present the experimental results in two experiments. First, we consider the case with 100 SKUs and show that the methods based on OO, as introduced in Section 1.3, can save the computing budgets by 2000 folds. Second, we modify the objective function to consider the requirement on satisfaction rates. The satisfaction rate is defined as the fraction of the time that the demand is satisfied by the finished goods inventory level. The experimental results demonstrate how the OO based method can be used as a platform to study the impacts of different factors on the total profit.

1.4.1 Experiment 1: 100 SKUs

There are 100 different SKUs. For each SKU, the demand at time t is a truncated Gaussian random variable with mean equals to $\mu(t)$ and coefficient of variation equals to Cv, i.e., $d(t)=\max(\zeta(\mu(t), Cv\mu(t)),0)$. We use seasonal-sine demand introduced in Section 1.2 to model the average demand $\mu(t)$. The ratio of the average demand from the peak season to the low season ranges from 3 to 7. The Cv of the SKUs ranges from 0.1 to 1.0, and the SKUs with higher Cv have lower demand than the SKUs with lower Cv. The ratio of the demand of the SKU with the highest Cv to the demand of the SKU with the lowest Cv is 5. The period of a season is 25 weeks, i.e., about half a year. We have 2 production lines, the lead time of the quick line is 1 week, while that of the regular line is 4 weeks. The weekly total production schedules should be maintained within 100±20% of the total production capacity. The "good enough" set G is defined as the top 5% of the solution space. We use the linear method introduced in Section 1.3 to generate the target inventory level, i.e.,

$$\text{target } \tau_i(t) = (a_1 + a_2 Cv_i)\,\mu_i(t + a_3).$$

In order to get the *true* performance value of a design, it will be necessary to run the detailed simulation. In this experiment, we assume that a detailed simulation utilizes the entire 100 SKUs with a simulation time = 500 weeks and the number of replications = 40.[1] When we estimate the *observed* performance value of the design, we run an aggregated 10 SKUs simulation

[1] In fact, when we run the simulations, it takes about a week to run on a Sun SPARC 20 station. The estimated performance values are still imprecise, but the errors are very small (the standard deviation of the error is about 0.05% of the performance value).

with time = 100 weeks and number of replication = 1.[2] Notice that the time needed to estimate the *observed* performance value is roughly ***1/2000*** of the time to estimate the *true* performance value of the design. We have reduced the computation time from 1 week to several minutes.

Table 8.1. The cost structure of the shirt manufacturer

Cost term	Value
Inventory holding cost per unit per week C_I (both finished good and WIP)	\$0.08
Quick line production cost per unit C_q	\$4.4
Regular line production cost per unit C_s	\$4
Material cost per unit C_m	\$10
Sale price per unit P_S	\$20

The cost structure is shown in Table 8.1. The results of the simulations are shown in Table 8.2.

Table 8.2. The alignment level and profit that we obtained when OO was used for the 100 SKUs case (periodic-sine demand)

s	k	J
1	1	356,834
5	4	358,999
10	7	358,999
20	11	358,999
50	26	359,504
100	38	359,504

In Table 8.2,

- s = number of designs selected by using the observed performance value.
- k = the average number of overlaps of the selected s designs with true top-50 designs, i.e., alignment level $|G \cap S|$. (These top-50 designs are obtained by running all 1000 designs for detailed simulation.[3])
- J = the best performance value (profit) in the selected s designs.

[2] From the simulation results, errors are about 3 to 4% of the performance value.

[3] Notice that this is a tremendous computational burden and precisely what our approach is trying to circumvent. However to lend credibility to our approach, this is the only way to prove its validity. Once established, we need not repeat this validation process in practical applications.

To get an idea of the absolute difference between the results obtained by OO and the true optimum, an upper bound of the profit can be obtained (Lee 1997). The idea is to consider a long enough simulation so that the system achieves the steady state. The average weekly production will be roughly equal to the average weekly sale; the ratio of the average weekly production of the quick line and the regular line should be close to the ratio of the capacity allocated to the quick line and the regular line (we can always fully utilize the capacity); and by Little's Law, the WIP should be equal to the average weekly production multiplied by the lead time. We will not deduce the upper bound in details. Please refer to (Lee 1997) for specific details. Based on these observations, the upper bound of the optimum profit is \$369,551.

From the results in Table 8.2, we can make the following observations.

- In order to get the *true* performance value[4] of all the designs, the simulations were run about one week, 24 hours a day, on a Sun SPARC 20 machine, but to get the *observed* performance values, we only needed a run of several minutes.
- The selected set *S* contains a high proportion of good enough designs. When we increase the size of selected set *S*, the number of alignments between the good enough set and the selected set *S* also increases.
- The performance value (manufacturing profit) of the best design in the selected set is indeed very close to the pre-calculated upper bound (3% from the upper bound), which means that this approach not only guarantees to find good designs but also the design is close to the optimum in this problem.

1.4.2 Experiment 2: 100 SKUs with consideration on satisfaction rate

For some companies, it is an important strategic requirement to satisfy customer demands. Failing to do so can result in not only the lost profits due to lost sales, but also the loss of future market share. This motivates a concept called the ***satisfaction rate***, which is simply the fraction of the time that the demand is satisfied by the inventory level. A satisfaction rate of 1 means that customer demands will always be fulfilled from inventory, or in other words, the inventory level is higher than the demand level every week. Satisfaction rate is defined as,

[4] The true performance values are obtained by running detailed simulation.

$$\text{satisfaction rate} = \frac{1}{\Pi}\sum_{t=1}^{\Pi} \iota\left(I(t)-d(t)\right)$$

Where

$$\iota(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}.$$

Therefore, in order to maintain a high level of satisfaction rate, it is unavoidable to keep a high inventory level, which will induce a cost. However, the relation between enforcing the satisfaction rate and the cost incurred is not obvious. In this section, by using the OO based method in Section 1.3, we can quickly find this relation, which will serve as a good indicator for the production and sales managers to know how to set their satisfaction rate level.

Assume that the satisfaction rate constraint is that the average satisfaction rate, *SR*, of all SKUs have to be above certain level, β, i.e.,

$$SR = \frac{1}{M\Pi}\sum_{i=1}^{M}\sum_{t=1}^{\Pi} E\left[\iota\left(I_i(t)-d_i(t)\right)\right] \geq \beta .$$

Therefore the scheduling problem becomes

$$\max_{\alpha\in\Phi,\gamma} E\left\{J(\alpha,\gamma)\right\} - \text{Penalty}(SR;\beta)$$

subject to the constraints on the production capacity and the inventory and WIP dynamics. After adding the satisfaction rate constraints, the problem becomes a constrained optimization problem. We can either use the constrained OO that was introduced in Chapter V to solve this problem directly, or use a penalty function to convert the problem back to unconstrained optimization. Since we will demonstrate the application of COO in a remanufacturing system performance optimization problem in Section 3, we focus on the second way in this section. The penalty function is a quadratic function which is defined as follows,

$$\text{Penalty}(x;\beta) = \begin{cases} c(\beta - x)^2 & \text{if } x < \beta \\ 0 & \text{otherwise} \end{cases} \tag{8.3}$$

where the coefficient c is a penalty function. If c is very large, we will have a hard constraint, i.e., the selected design has to satisfy the constraint. The good enough set is defined as the top-n% designs. The parameter setting is almost the same as in experiment 1, except that the sales price, P_S, is \$16, which is much lower. For the lower profit margin, we will keep a lower inventory level, and therefore the design that gives the optimum profit level will have a low satisfaction rate. With an interest in this problem, we will see the costs incurred when we enforce the high satisfaction rate constraint.

The results of the simulations are shown in Table 8.3.

Table 8.3. The results of the simulation when we have satisfaction rate constraints, where s is the number of designs selected by using the observed performance value (Note: Pre-determined upper bound for profit = \$106,492)

s	J with no satisfaction rate constraint	J with $\beta = 0.97$	J with $\beta = 0.98$	J with $\beta = 0.99$
1	\$96,030	\$92,686	\$93,819	\$88,283
5	\$96,413	\$95,210	\$93,819	\$92,147
10	\$96,413	\$95,210	\$94,022	\$92,147
20	\$96,413	\$95,210	\$94,022	\$92,147
50	\$96,413	\$95,210	\$94,022	\$92,147

From the results in Table 8.3, we observed that if we have to enforce the satisfaction rate higher than 0.97, there will be a profit lost of \$800[5]. This table, which is obtained within an hour, will be useful for a manager to know the cost associated with the satisfaction rate constraint. Actually, by using the OO-based method as a simulation-based optimization platform for the scheduling problems in apparel manufacturing systems, it is now possible to study many aspects of the system in a more quantitative way, such as the performance of new supply chain contracts between the manufacturers and the retailers (Bouhia 2004; Volpe 2005).

1.5 Conclusion

In this section, we apply the OO methods introduced in Chapter II to a scheduling problem in the apparel manufacturing system. We show how to incorporate the problem information in the initial random sampling of the designs and the construction of the crude model. The results are very promising. The OO-based method is very fast and only needs several

[5] When this constraint increases to 0.99, the cost incurred will be roughly \$4,000.

minutes to screen out the good enough designs. We only use 1/2000 of the computation time that brute-force simulation would have taken in Experiment 1. The performance of the design found by OO is not only within the top-5% of the design space, but also within 3% from an upper bound of the optimum. This method supplies a simulation-based optimization platform to quantitatively analyze the performance of the apparel manufacturing system, which supplies many possibilities for further improving the performance of the apparel manufacturing system. Note that we only consider the linear model to generate the target level in this section. There are also other models to approximate the periodic property of the mean value of the demand better than the linear model. Interested readers may refer to (Lee 1997) for more details.

2 The turbine blade manufacturing process optimization problem

In this section, we consider a turbine blade manufacturing process optimization problem (Yang 1998). The integrated blade and rotor is manufactured via extrusion, which is similar to the manufacturing of plastic parts, but with much tougher high strength metal used and higher quality requirement on the product. As an optimization problem, such a manufacturing process is distinguished by the large number of parameter settings of all the operations and the difficulty to accurately evaluate the quality of the final production. On the one hand, the parameters, such as the initial size of the billet, the ram velocity of the plunger, and the ambient temperature of the work piece being processed, usually take continuous values. There are a huge and in principle infinite number of possible parameter settings combinations. On the other hand, for security and combat considerations, the aircraft usually has high quality requirements on the turbine blade. This quality depends on the physical property of the turbine blade, such as the effective strain field, the effective strain rate field, and the maximum load-stroke. These physical properties are determined by the deformation process of the work piece during the manufacturing, which can only be accurately described by the finite element method (FEM). It usually takes hours if not days to use FEM to simulate (calculate) the entire deformation process, and accurately evaluate the quality of the turbine blade thus produced. Giving the extremely large search space, with the lack of structure information (such as the gradient information) of the search space, it is computationally infeasible to find the optimal parameter settings using brute force. In this section, we show how OO can help to solve this problem. The FEM is a deterministic but complex calculation. Based on our previous discussion in Section II.6, OO can also be applied in this type of problem. One purpose of this section is to justify this by using a real-life example. By applying ordinal optimization, we are able to find a good enough parameter setting based on a computationally fast but crude model, and save the computing budgets by 95%, comparing with brute force. We formulate the problem in Section 2.1, show the application of OO in Section 2.2, and briefly conclude in Section 2.3.

2.1 Problem formulation

Peripheral blades and central rotor compose the primary parts of an airplane turbine engine. The quality and reliability of the turbine blade is important for the functionality of the aircraft engine. To meet the symmetry requirement of operation under high rotations, the blades must be balanced around the rotor, which is a difficult and costly production stage if the blade and the rotor are produced separately first and then fused together. To solve this problem, the integrally-bladed rotor (IBR) is invented, which, as the name shows, is a component that integrates the blade and the rotor manufacturing (Fig. 8.6). The high engine performance demands of customers require that these components be made from traditionally "difficult-to-process" materials such as high-temperature titanium alloys, intermetallics, and Nickel-based superalloys. Often these materials are stronger than conventional tool materials and require special tooling and high temperatures for processing. Coupled with this tooling constraint are the high-strength and high-reliability requirements, which call for good to excellent control of the final metallurgical structure and pedigree of the materials. These requirements lead to cautious designs of processing operations, often with redundant operations to ensure acceptable final metallurgical characteristics. Therefore, the manufacturing of such components consists of a significant part of the life-cycle cost of a turbine engine. With growing demands on aeronautical technology, there is a strong interest in the optimization of the manufacturing process of these components.

The manufacturing process of an IBR is shown in Fig. 8.7. First, the raw material is cast into the billet with the required radius and height. Then, hot isostatic pressing (HIP) is used to reduce the porosity of metals, which improves the mechanical properties and increases workability. After that, by hammering on the end, the billet is made shorter and thicker. This operation is called "upset". Due to the high quality requirement on the IBR, there are two forge stages. In the first stage, by heat treatment, the cylindrical billet melts down, flows into the blocker die, and is rammed into the expected shape. This is called the blocker forge. In the second, the work piece is further forged to a shape near the net shape of an IBR, thus called the near-net-shape (NNS) forge. After all these, the work piece is machined to IBR. Among these operations, the blocker forge is the most complex one, which involves various thermo-mechanical processes, so we focus specifically on the optimization of the blocker forging stage in this case study. Roughly speaking, there are several stages when filling the blocker die in the blocker forge. A more detailed discussion will be presented in Section 2.2 and illustrated by Fig. 8.9.

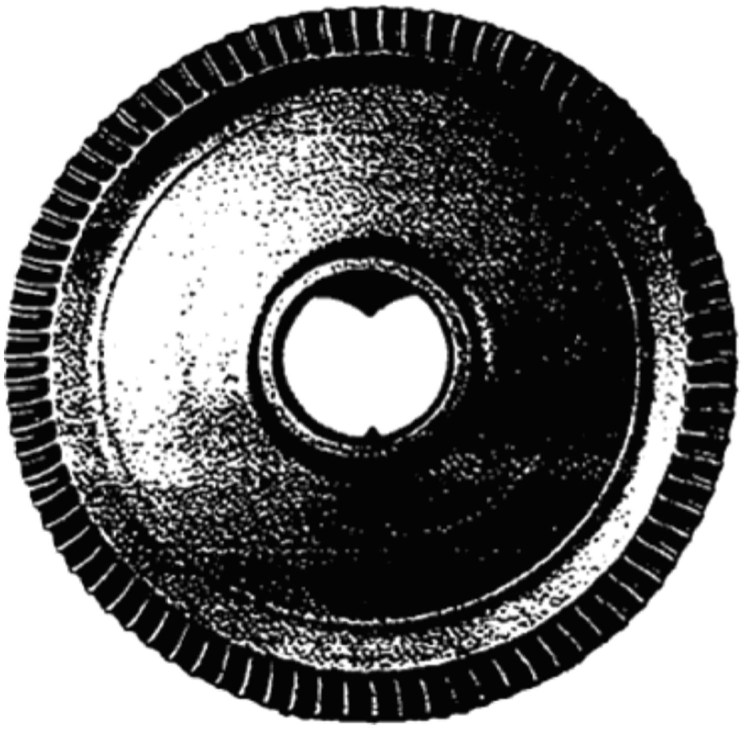

Fig. 8.6. An integrally-bladed rotor

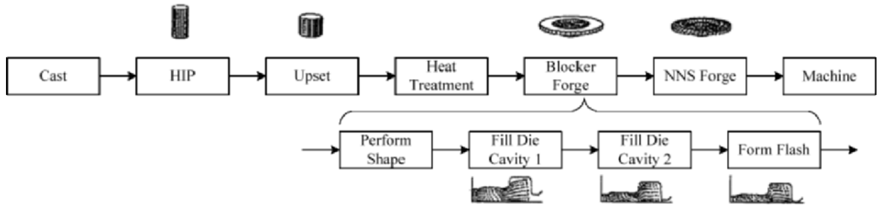

Fig. 8.7. The manufacturing process of an IBR

The three characteristics of the IBR problem that make it difficult to be solved by traditional optimization approaches can be described as follows.

1. **High complexity** in the structure of the problem. As Fig. 8.7 illustrates, many of the processes involved are nonlinear in nature and are interdependent. As a result, accurate evaluation of a single design (i.e., a specification of the parameter settings of the manufacturing process) via finite element calculations generally takes hours if not days or months to calculate, putting aside of the issue of a limited computational budget. It is not hard to see how the traditional searching approaches that rely on the availability of accurate design performance evaluations will become impractical under limited computation budgets and time constraints.
2. **Unpredictability of Inherent Imprecision.** In the nontrivial case when a simpler model that is different from the true model (the FEM model) is used for search, the model imprecision must be accounted for during the selection process. In essence, the IBR problem becomes a noisy search problem that most traditional optimization algorithms do not address.
3. **Very Large Search Space.** The design space of the IBR problem explodes exponentially with the large number of parameters involved. In the blocker forge stage considered in this case study, there are seven parameters (all defined over continuous interval ranges) and a choice of the die shape (from finite number of candidates) that can be controlled. (These parameters will be introduced later in this subsection.) Even if all parameters are discretized into 10 discrete values, the overall design space is roughly 10^7. With such a big space, the traditional build-and-test method is out of the question. For the same reasons, a computerized brute-force selection process will require a large number of evaluations, or numerical simulations, that are clearly in conflict with a limited time budget.

Giving the above difficulties, we have to find better ways to screen out some good parameter settings first, before we adopt the detailed FEM calculation. This is where OO helps. First, we introduce the mathematical problem formulation.

In the blocker forge, we can control the following parameters: the initial radius, height, and temperature of the billet, the temperature and shape of the die, the ram velocity, the ambient temperature, and the friction coefficient. These variables usually take real values within the parameter ranges, except for the die shape, which usually has a small number of candidates. Each design θ in this problem is defined as a specification of all the above seven parameter settings (the initial radius, height, and temperature of the billet, the temperature of the die, the ram velocity, the ambient temperature, and the friction coefficient) and a choice of the die shape. Then the design space Θ contains all the possible parameter settings. Given a design θ, through the detailed FEM simulation of the deformation process of the work piece during the operations, we can obtain the following physical quantities that determine the physical property of the IBR: the effective strain field, the effective strain rate field, the effective temperature field, the maximum stress field on the die surface, and the maximum load-stroke. The value of the cost function (definition follows) that evaluates θ, and the parameter setting of the operations, can be calculated.

The cost function used in this study consists of eleven terms, including five accounting costs, four quality loss penalties, and two inspection overheads. In the accounting cost category, the five cost factors considered are: material cost, initial reduction setup cost, initial reduction press cost, forge press cost, and die wear cost. The material cost refers to the market value of the initial billet. When the aspect ratio of the initial billet is too high, an initial reduction with a bottle-cap die set is necessary to avoid a buckling forge situation. The initial reduction setup cost and the initial reduction press cost refer to the setup cost and the operation cost per press run in this particular situation. The forge press cost is the cost of utilizing the press to complete the forge process. The die wear cost is the cost of the forge die set divided by the average number of production runs in the life of that TZM (Molybdenum Alloy) die set. The general equations[6] of the respective terms are:

Table 8.4. Accounting costs (Yang 1998)

Cost term	General form
Material cost	C_{ma}×total billet volume
Initial reduction setup cost	C_{rs}×function$_{\mathrm{rs}}$(aspect ratio)
Initial reduction press cost	C_{rp}×function$_{\mathrm{rp}}$(aspect ratio)
Forge press cost	C_{fp}
Die wear cost	C_{dw}×function$_{\mathrm{dw}}$(billet temperature, maximum die pressure, processing time length)

In the penalty category, the four terms reflect the constraints on the material properties and the limits of the die capacity. These four terms are: force penalty, heat treatment penalty, heat remedy cost, and strain induced porosity (SIP) damage penalty. Force penalty reflects the maximum force

[6]Since our purpose here is to give a general picture of the complexity of the problem, we choose not to display the detailed mathematical formula. Actual detail can be found in (Yang 1998).

constraint of the press through a quadratic penalty function. The heat treatment penalty reflects the material strain constraint through an approximation of the fraction globalized in Ti64 material from the strain information of the work piece. The heat remedy cost specifies the cost of the heat treatment to remedy the strain imperfections in the work piece. The SIP damage penalty reflects the strain rate constraint and the temperature constraint on the final product through the estimation of the equilibrium volume fraction of the alpha phase for Ti64. The general equation, again in the spirit of footnote #1, for the four constraints are:

Table 8.5. Penalty terms (Yang 1998)

Penalty term	General form
Force penalty	$P_{\text{fp}}\times\text{function}_{\text{fp}}$(maximum die force)
Heat treatment penalty	$P_{\text{ht}}\times\text{function}_{\text{ht}}$(billet strain, billet temperature)
Heat remedy cost	$P_{\text{hr}}\times\text{function}_{\text{hr}}$(billet strain, billet temperature)
SIP damage penalty	$P_{\text{SIP}}\times\text{function}_{\text{SIP}}$(billet strain rate, billet temperature)

Finally, in the inspection category, the two terms are: forge setup inspection cost and ultrasonic inspection cost. The forge setup inspection is a fixed cost term to insure safety during the forge process. The ultrasonic inspection cost is the mandatory ultrasonic non-destructive evaluation prior to the acceptance of the final product. The general equation for two inspection overheads can be described as:

Table 8.6. Inspection overheads (Yang 1998)

Inspection overhead	General form
Forge setup inspection	O_{si}
Ultrasonic inspection	O_{ui}

The cost function can be summarized as follows:

$$\begin{aligned} J(\theta) = {} & C_{\text{ma}}\times\text{total billet volume }(\theta) \\ & + C_{\text{rs}}\times\text{function}_{\text{rs}}(\text{aspect ratio}(\theta)) \\ & + C_{\text{rp}}\times\text{function}_{\text{rp}}(\text{aspect ratio}(\theta)) \\ & + C_{\text{fp}} \\ & + C_{\text{dw}}\times\text{function}_{\text{dw}}(\text{billet temperature }(\theta), \\ & \qquad \text{maximum die pressure }(\theta), \\ & \qquad \text{processing time length }(\theta)) \\ & + P_{\text{fp}}\times\text{function}_{\text{fp}}(\text{maximum die force}(\theta)) \\ & + P_{\text{ht}}\times\text{function}_{\text{ht}}(\text{billet strain}(\theta)) \end{aligned}$$

$$\begin{aligned}
&+ P_{\text{hr}}\times\text{function}_{\text{hr}}(\text{billet strain}(\theta))\\
&+ P_{\text{SIP}}\times\text{function}_{\text{SIP}}(\text{billet strain rate}(\theta),\\
&\qquad \text{billet temperature}(\theta))\\
&+ O_{\text{si}}\\
&+ O_{\text{ui}}.
\end{aligned}$$

2.2 Application of OO

The basic idea of OO is to use a crude model, which is computationally easy, to screen out quickly some good enough designs. From the last subsection, we can see that the value of the cost function depends on the parameter settings of the operations and the physical properties of the IBR thus produced. To accurately evaluate the physical properties of the IBR, the FEM model that describes the thermo-mechanical processes have to be used. In order to apply OO in this problem, it is crucial to find a crude model, which approximates the thermo-mechanical processes in a fast way and can give rough estimate of the physical properties of the IBR thus produced. Fortunately, the Ohio University Forge Simulation Model (the OU model) (Gunasekera et al. 1996) offers us such a choice. Compared with the FEM model, which contains all the details in the forging process, the OU model introduces the following simplifications. First, Gunasekera et al. showed that all changes in continuum properties such as strain, strain rate, and temperature can be described as functions of geometry or changes in the geometry with respect to time (Gunasekera et al. 1996). Based on this observation, instead of tracking down the changes in all the physical quantities at the same time like the FEM model does, the OU model only tracks down the change of the geometry of the work piece. Second, the general die shape (quarter cross-section view) is shown in Fig. 8.8. Instead of tracking the entire field of the continuum properties such as strain, strain rate, temperature, pressure, and grain size, the OU model divides the work piece into four parts: web, flange 1, flange 2, and flash. It calculates only the estimated average of these characteristic values in the regions with the assumption that these thermo-mechanical properties are uniform inside each region. Third, the evolution of the work piece during the forge process is simplified. When a billet is heated, input to the die, and rammed, it does not fill in every part of the die immediately. This process takes some time, and consists of five sub-stages as shown in Fig. 8.9. We can see that the shape of the work piece is not regular during these sub-stages. Instead of describing this deformation in details like the FEM model does, the OU model simplifies the five sub-stages as shown in Fig. 8.10. As we can see, the form of the work piece is more regular than in Fig. 8.9. Based on this simple approximation of the work piece geometry evolution, the OU model calculates the changes in the height and diameter, and calculates the strain and strain rate values. The calculation of other physical quantities, such as temperature, microstructure, die pressure, and the total die force, are calculated based on semi-empirical models together with some other approximations. In this way, the simulation is much simplified, and much faster than the FEM. A comparison study shows that it takes the FEM about 4 hours to evaluate one design, but only 0.1 seconds for the above crude model since it is made up of analytical formula (Yang 1998). This is a tremendous saving in computing time.

Note that the above crude model is a deterministic but simple calculation. Because the true model (i.e., the FEM) is too complex, the deterministic errors between the two models are complex and hard to predict. Based on our discussion in Section II.6, we can regard these errors as random noises, and treat the problem as if the true model is a stochastic simulation. We use a case study to justify these statements.

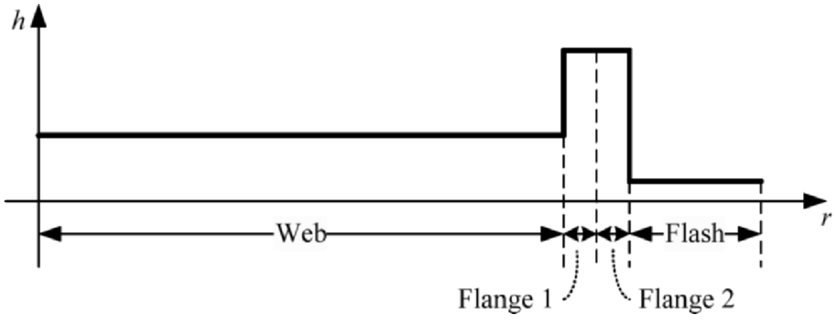

Fig. 8.8. General die shape (quarter cross-section view)

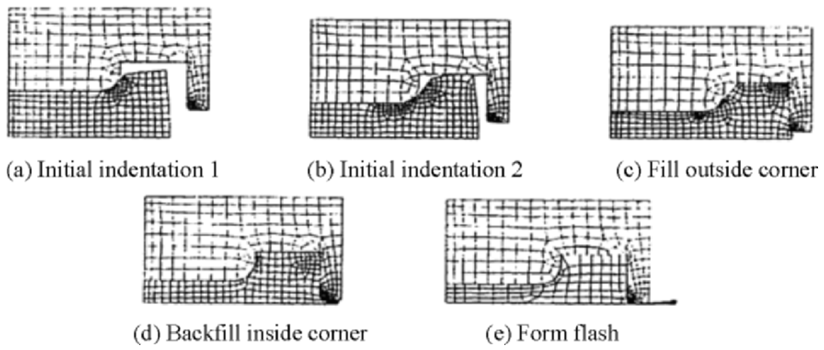
(a) Initial indentation 1
(b) Initial indentation 2
(c) Fill outside corner
(d) Backfill inside corner
(e) Form flash

Fig. 8.9. The geometry evolution of the work piece described by the FEM model (different time snaps of the work piece during the forge process) (Yang 1998)

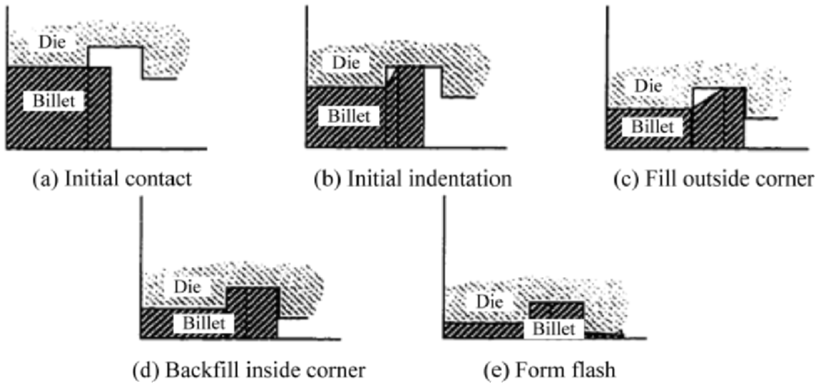

Fig. 8.10. The geometry evolution of the work piece described by the OU model (different time snaps of the work piece during the forge process)

In the following case study, we take the parameter ranges as shown in Table 8.7. The die shape can be defined by 6 variables, as shown in Fig. 8.11. There are four candidates for the die shape (Table 8.8). First we study the difference between the crude model and the true model (i.e., the FEM). We uniformly randomly sample 80 designs from the entire design space. We use the number 80 because it is too time-consuming to use the detailed model to accurately evaluate the performance for a large number of designs. Actually it takes about 14 days of continuous computing to finish the performance evaluation of these 80 designs using FEM. Comparing with so long a time, it is amazing how fast the crude model is. Only 8 seconds! We plot the observed cost vs. true cost in Fig. 8.12.

Table 8.7. Parameter ranges

Parameter	Parameter range ([min, max] unit)
Initial billet radius	[3, (flange radius – 0.5)] inch
Initial billet height	[(web height + 0.5), (3×initial billet radius)] inch
Die temperature	[1562, 1832] °F
Ram velocity	[0.1, 0.6] inch/sec
Initial billet temperature	[die temperature – 25, die temperature + 25] °F
Ambient temperature	[die temperature – 25, die temperature + 25] °F
Friction coefficient	[0.2, 0.8]

Table 8.8. Die shape candidates

Die shape index	r_1	r_2	r_3	h_1	h_2	h_3
0	8	9	10	1.0	2.4	0.15
1	8	9	11	0.5	2.4	0.20
2	8	9	10	1.0	2.4	0.20
3	8	9	11	1.0	2.4	0.05

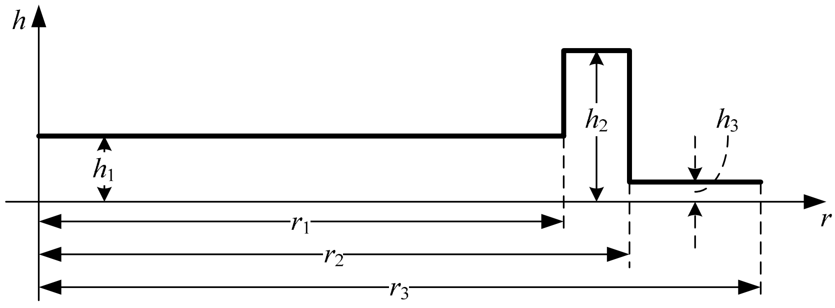

Fig. 8.11. Die shape variables

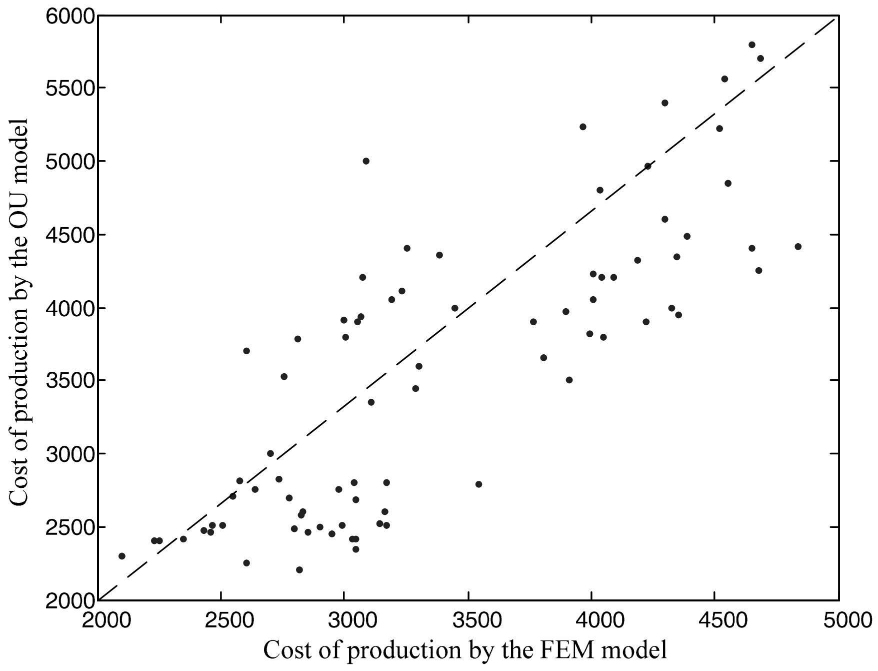

Fig. 8.12. Observed cost (by the OU model) vs. true cost (by the FEM model)

In Fig. 8.12, each dot on the graph represents a design with its x value being the true cost calculated by FEM, and its y value being the cost predicted by the crude model. The dotted line is the 45° line, on which all the points shall fall if the crude model conforms exactly to the FEM model. Data analysis shows that the prediction given by the crude model has an average %error (defined as (|crude model predicted value|-|FEM value|)/ |FEM value|) of 14% and a standard deviation of %error at 12%. The maximum %error observed is 62% and the minimum is 0%. This means the crude model does not give accurate performance evaluations. Actually in the 80-design instance shown in Fig. 8.12, the observed best design is the truly 20-th best. If we focus only on the best design, we can hardly succeed. Now, we apply OO to find some good enough designs with high probability.

Due to the extremely large computation needed to accurately evaluate 1000 designs (an estimate shows that it will take about 160 days to finish all the calculation), we only have the true performance of 80 designs. In the following, we will regard these 80 designs as the representative set Θ_N. Astute readers might notice that the UAP table in Section II.5 was obtained under the assumption of N=1000. They may ask whether it is reasonable to use that UAP table to estimate the size of the selected set when N=80. The numerical results, which will be shown later, justify this usage.

We show the observed OPC of these 80 designs in Fig. 8.13, which belongs to the neutral type. To see the difference between the OU model and the FEM model, we also show the corresponding true performances of these designs. Then we randomly select several of these 80 designs to estimate the normalized noise level, which is 0.1729. This belongs to the small noise level in the UAP table (Table 2.1 in Section II.5). For different values of the good enough set (g), the required alignment levels (k), the predicted values of s based on Eq. (2.42) are shown in Table 8.9, denoted as $\hat{s}_1$. Since the true noise level is smaller than 0.5, we use linear interpolation to obtain another group of predicted values of s, denoted as $\hat{s}_2$. This linear interpolation method will be explained in details later. For the instance of these 80 designs, we also present in Table 8.9 a size s^* whose value is decided such that there are at least k truly good enough (recall that we know the true performance of these 80 designs) designs in the observed top s^* ones. This quantity is shown here as a measure of the ideal size of the selected set to achieve the desired alignment level. We can see that $\hat{s}_1$ is always an upper bound of s^*, which shows the conservative nature of the UAP table. Now we show how to obtain less conservative estimate of s.

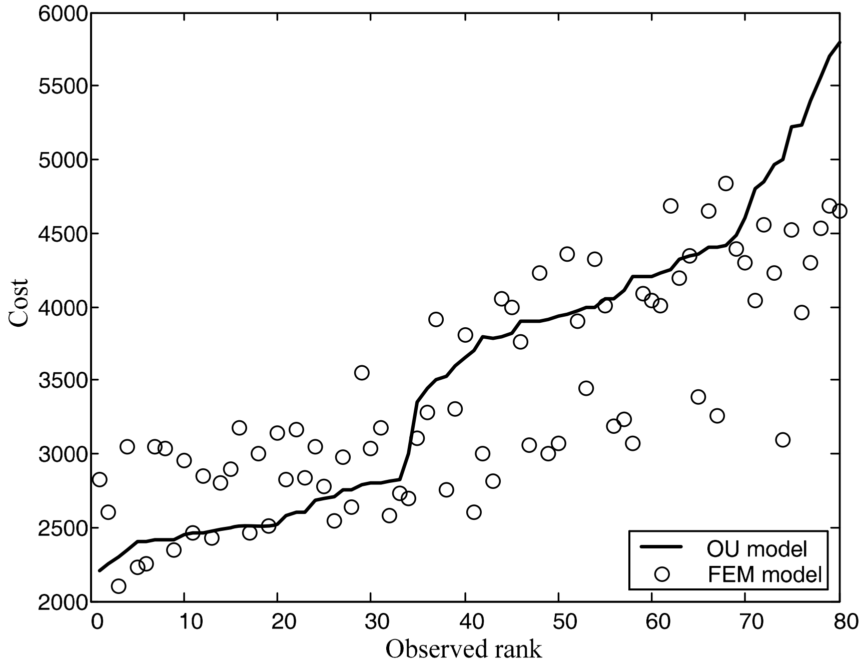

Fig. 8.13. The observed OPC

Table 8.9. The predicted and true selected sizes

g	k	s^*	$\hat{s}_1$	$\hat{s}_2$
1 (top 1.25%)	1	3	80	29
4 (top 5%)	1	3	13	6
	2	5	25	10
	3	6	37	15
	4	9	49	20
8 (top 10%)	1	3	6	3
	2	5	10	5
	3	6	15	8
	4	9	19	10
	5	11	24	12
	6	13	28	14
	7	17	33	16
	8	19	38	19

One important reason that $\hat{s}_1$ is conservative is that the true noise level is not 0.5, but 0.1729, which is much smaller. So we use linear interpolation to obtain better prediction of s. Note that if the noise level is 0, which means we know the true performance of all the designs. We only need to select the observed top-k designs (which are just the true top-k designs) to cover k truly top-g designs, i.e., $s = k$. Through previous calculation, we also know the predicted value of s when the noise level is 0.5, i.e., the $\hat{s}_1$ in Table 8.9. Through linear interpolation of size s in terms of the noise level, we obtain the prediction of s when the noise level is 0.1729 (denoted as $\hat{s}_2$ in Table 8.9). For example, when $g = 8$, $k = 1$, if the noise level is 0.5, $\hat{s}_1 = 6$; if the noise level is 0, s should be 1; now, the noise level is 0.1729, so the new estimate can be obtained from the following linear interpolation: $(6-1)/0.5\times0.1729+1\rceil=3$, which is denoted as $\hat{s}_2$ in Table 8.9. We can see that $\hat{s}_2$ is less conservative than $\hat{s}_1$, which usually is an upper bound of the true value s^*, and close to the true value s^* when $g = 8$. The only exception is when $g = 8$ and $k = 7$, the predicted $\hat{s}_2$ is smaller than s^*, but still very close. After goal softening, if we want to find at least one of the top-10% designs with high probability, the prediction $\hat{s}_2$ says we only need to investigate the observed top-3 designs. Comparing with brute force, we reduce the detailed performance evaluation by more than 25 folds (from 80 to 3). Through these numerical results, we see OO can help to save the computing budgets even when the objective function is not a stochastic simulation but a deterministic complex calculation. The results also justify the application of the UAP table when the representative set Θ_N is smaller than 1000.

2.3 Conclusion

In this section, we have considered a turbine blade manufacturing process optimization problem. The objective function can only be accurately evaluated through a complex but deterministic calculation. By using a crude model, which is more than 10000 times faster than the detailed FEM model, together with the idea of goal softening, we are able to save the computing budgets by more than 25 folds, comparing with brute-force calculation. This justifies that we can apply OO to solve the problem, taking the deterministic but complex error between the crude model and the detailed model as random noise. It should be noted that we omit many technical details to simplify the above discussion, such as the equations to describe the forging process in the FEM, and parameter settings of the 80 designs that are randomly sampled. Readers can refer to (Yang 1998) for more details. (Yang 1998) also shows that we can increase the accuracy of the cost prediction by incorporating more information in the crude model. This in turn helps to reduce the selected set thus required, and further save the computing budgets. We also show how to reduce the selected set size by interpolation. Another way to reduce the selected set size can be found in Section 3.3 below.

3 Performance optimization for a remanufacturing system

In this section we consider a remanufacturing system (Song et al. 2005a, 2005b). The goal is to manage the number of machines in the repair shop and the number of new parts to order in the inventory, so that the maintenance cost is minimized and the average maintenance time for an asset is not too long. Since there are two considerations in this problem, we can regard the maintenance cost as the objective function, and the requirement on the maintenance time as the constraint. In this way, we have a constrained optimization problem. The corresponding problem formulation will be discussed in Section 3.1 in details. Due to the time-consuming simulation-based evaluation of both the objective function and the constraint, we apply the constrained ordinal optimization as introduced in Chapter V. The application procedure is shown in Section 3.2. The experimental results are also presented, which is promising because we save the computing budgets by 25 folds. However, alternatively we can regard both the maintenance cost and the maintenance time as objective functions. We have then a two-objective function simulation-based optimization problem. We apply the vector ordinal optimization as introduced in Chapter IV. Especially, we show how to incorporate the problem information to further save the computing budgets in VOO. The details are presented in Section 3.3. We make a brief conclusion in Section 3.4.

3.1 Problem formulation of constrained optimization

Due to the consideration of saving the production cost and reducing environmental pollution, the study on remanufacturing system has attractted more and more interest recently (Guide et al. 1999; Guide 2000). The basic idea of remanufacturing system is to re-use the parts (sometimes after repair) from the old products to produce new products. This idea is especially useful for very expensive assets (such as aircraft jet engines) which consist of many parts. Rejecting old parts directly not only causes environmental pollution easily but also increases the production cost of a new asset. Thus the old parts are usually recycled after some repair.

A detailed model of a remanufacturing system is shown in Fig. 8.14. Due to random failure, the asset is shipped to this remanufacturing system. After disassembled into parts and inspected, the parts still in serviceable condition will be directly sent to a certain place and wait to be reassembled into new assets. The other parts need some repair and are sent to the repair shop. After the repair, the parts enter an inventory, and are then assembled into new assets together with the parts in serviceable condition, then leave the system. Since the parts of the same type are not distinguished from each other during the assembling, the inventory is also called the rotable inventory in practice (Kleijn and Dekker 1998). Due to the random arrival of the asset to this system and the uncertainties in the waiting time and repair time in the repair shop, sometimes there might not be enough parts when assembling a new asset. To avoid this "lack of synchronization", new parts can be ordered to the inventory. The parameters we can control are the number of machines in the repair shop and the number of new parts to order in the rotable inventory. We care about two performances metrics of the system. One is the average maintenance cost of an asset. The other one is the average maintenance (remanufacturing) time of an asset. These two performances are obviously related to one another. When there are more machines in the repair shop or more new parts are ordered in the inventory, the maintenance time can be reduced but the maintenance cost increases. The question is how we can minimize the maintenance cost[7], given the requirement on the maintenance time.

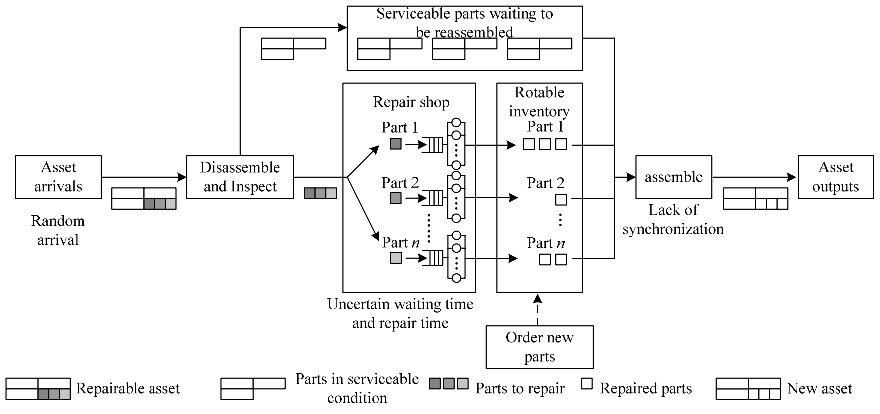

Fig. 8.14. Detailed model of the remanufacturing system

[7] Actually the opportunity to do preventive maintenance or not while the engine is disassembled is another decision which we will not consider in this example.

We mathematically formulate the problem as follows. Consider a planning horizon with m seasons (there are 3 months in each season). Suppose there are n parts in each asset, each of which requires a specific type of machine to repair. Let $C_{i,j}$ be the number of machines of type i that are used in season j. And let $\Delta I_{i,j}$ be the number of new parts ordered at the beginning of season j, which will be shipped to the inventory and become available in the next season. The maintenance cost consists of the machine cost ($\alpha\sum_{i=1}^{n}\sum_{j=1}^{m}C_{i,j}$), the cost for ordering new parts ($\sum_{i=1}^{n}\sum_{j=1}^{m}\Delta I_{i,j}$), and the inventory holding cost ($\beta\sum_{i=1}^{n}\sum_{j=1}^{m}I_{i,j}$), i.e.,

$$J(C,\Delta I)=\alpha\sum_{i=1}^{n}\sum_{j=1}^{m}C_{i,j}+\sum_{i=1}^{n}\sum_{j=1}^{m}\Delta I_{i,j}+\beta\sum_{i=1}^{n}\sum_{j=1}^{m}I_{i,j},$$

where α and β are positive real numbers, C and ΔI are n-by-m matrix, with $C_{i,j}$ and $\Delta I_{i,j}$ as the components. $C_{i,j}$ should not exceeds a specific value in two neighboring seasons, i.e.,

$$\left|C_{i,j-1}-C_{i,j}\right|\le\Delta C_i,$$

and $C_{i,j}$ should be controlled within a reasonable range, i.e.,

$$C_i^{\min}\le C_{i,j}\le C_i^{\max}.$$

Similarly, the order quantity of the parts cannot exceed a specific value, i.e.,

$$\Delta I_{i,j}\le\Delta I_i^{\max}.$$

Then it is obvious that the amount of part i in season j in the rotable inventory satisfies

$$I_{i,j}=I_{i,j-1}+p_{i,j-1}+\Delta I_{i,j-1}-q_{i,j-1},$$

where $p_{i,j-1}$ represents the number of part i that were finished in season j-1, $q_{i,j-1}$ represents the number of part i that were used in season j-1. Let $a(k)$ be the inter-arrival time between the k-th and the (k-1)-th asset. The first asset arrives at time $a(1)$, and the k-th asset arrives at time $\sum_{i=1}^{k}a(i)$.

After the *k*-th asset is disassembled and inspected, some parts are in serviceable condition and will be assembled with some other parts from the inventory into a new asset, then leave the system at time $\eta(k)$. The maintenance time for this asset is defined as

$$T(k) = \eta(k) - \sum_{i=1}^{k} a(i).$$

During the planning time horizon, the probability that the maintenance time exceeds a given limit T_D is

$$\text{Prob}\left[T(k) > T_D / \eta(k) \le T_C\right].$$

Suppose the constraint is that this probability should not be large, say less than P_0. Then the constrained optimization problem is

$$\min_{C,\Delta I} J(C,\Delta I) \text{ s.t. } \text{Prob}\left[T(k) > T_D / \eta(k) \le T_C\right] < P_0 .$$

Although the above problem formulation might be simpler than the real system, this formulation preserves the basic characteristic of the real system, especially the difficulties. First, both the number of the machines and the number of new parts can only take discrete values. There are $n \times m$ variables in C and ΔI each, and thus the size of the design space is close to

$$\prod_{i=1}^{n} \left(\Delta I_i^{\max} \min\left\{2\Delta C_i, C_i^{\max} - C_i^{\min}\right\}\right)^m ,$$

which grows exponentially as n and m increases. Simulation is the only way to do detailed performance evaluation for each (C, ΔI). Second, both the objective function and constraint are simulation-based. To obtain an accurate performance evaluation of the objective function and the constraint, we need ~1000 replications, which will take 30 minutes for each design by using the Enterprise Dynamics Software (Song et al. 2005a). If we want to accurately evaluate the feasibility of 1000 randomly sampled designs, we will need 500 hours, which is a very long time. In the next subsection, we apply constrained ordinal optimization to deal with these difficulties.

3.2 Application of COO

As introduced in Chapter V, the idea of COO is to use a feasibility model to quickly screen out the feasible designs (probably with some mistakes), and apply a crude model within these designs that are predicted as feasible to find some truly good enough and truly feasible designs. In order to apply COO to the remanufacturing system, we need to find an imperfect feasibility model for the constraint and a crude model for the performance.

3.2.1 Feasibility model for the constraint

By definition, any method that can predict the feasibility of a design with reasonable accuracy (say higher than 0.5) can be a feasibility model used in COO. This gives us a lot of freedom, such as heuristics and experiences. Of course, a feasibility model with higher accuracy will make a smaller number of mistakes, thus can further save the computing budgets. In this example, we use a machine learning method to obtain a feasibility model. The idea is as follows: First, we randomly sample a small number of designs, and then use brute-force simulation to accurately determine the feasibility of these designs. Input these designs and the corresponding feasibility as the training data, and use a machine learning method to discover the relationship between the parameter setting in the design and the feasibility. When the training finishes, we obtain a model. When a new design is input to this model, a predicted feasibility will be output. In this remanufacturing system, a feasibility model was found in this way. For technical details, such as what training method is used, and what the feasibility model looks like, please refer to (Song et al. 2005a). The feasibility model is very fast (0.003 second to predict the feasibility of a design on the average) and has a high accuracy, 0.985, which means if we randomly sample 1000 designs that are predicted as feasible by this feasibility model, an average of 985 designs are truly feasible. Although COO can work with a feasibility model with much less accuracy, such a high accuracy does allow us to save the computing budgets by 25 folds, as will be shown later in this subsection.

3.2.2 Crude model for the performance

After the designs predicted as feasible are screened out by using the feasibility model, we need a crude model to sort these designs according to the observed performance. The crude model should be computationally fast, and only need to give a rough estimate of the performance of the design. In one extreme case, blind pick does not need the estimate of the performance.

Since no problem information is utilized in the blind pick, the required size of the selected set is an upper bound of the case when other crude models are used, e.g., using a single replication of the simulation to estimate the performance.

The application procedure of the COO (feasibility model with blind pick) is summarized in Box 8.2.

Box 8.2. Application procedure of COO (feasibility model with blind pick) in the remanufacturing system

Step 1: Uniformly randomly sample N designs from the entire design space.
Step 2: Use the feasibility model to screen out the predicted feasible designs.
Step 3: User defines the size of the good enough set g and the required alignment level k.
Step 4: Using the accuracy of the feasibility model, we can calculate the size of the selected set s.
Step 5: Blind pick s designs from the predicted feasible list.
Step 6: The COO theory ensures that there are at least k truly good enough and feasible designs in these s selected designs with high probability.

3.2.3 Numerical results

To get an idea of how much computing budget we can save by using COO, we show the following experimental results. Consider a planning for 8 seasons (24 months), $m = 8$, $T_C = 720$ days. Suppose there is one part that needs to be repaired (i.e., $n = 1$) after disassembly. The inter-arrival time in season i contains exponential distribution, i.e.,

$$\text{Prob}\left[a(k)=t\right]=\lambda_i e^{-\lambda_i t},$$

where the time unit is day, and the λ_i in the 8 seasons take the values of 3.5, 3.0, 2.5, 2.0, 2.5, 3.0, 3.5, and 3.0. It takes 5 days to disassemble and check each asset. The repairing time of the parts satisfies the triangular distribution, with a minimum of 30 days, a maximum of 90 days, and an average of 60 days. It takes 7 days to reassemble the parts. In this case, C and ΔI are both 8 dimensional row vectors, and the elements within are C_i and ΔI_i, representing the number of machines in season i, and the order quantity at the beginning of season i. $11 \le C_j \le 40$, $0 \le \Delta I_j \le 7$, $C^{\max}=40$, $C^{\min}=11$, $\Delta C = 5$, $\Delta I^{\max} = 7$, $\alpha = 1$, $\beta = 0.2$, $T_D = 100$ days. The size of the design space is 5.8×10^{14}. The requirement on the maintenance time is:

$$\mathrm{Prob}\left[T(k) > T_{\mathrm{D}} / \eta(k) \le T_{\mathrm{C}}\right] < 0.05\,,$$

i.e., P_0=0.05.

To get a rough idea on how many designs are feasible, we uniformly randomly sample 1000 designs and use brute force simulation to obtain the true performance and feasibility of these designs, as shown in Fig. 8.15. We can see that a lot of designs are not feasible. If we do not have a feasibility model and directly apply OO in this problem, there will be a lot of infeasible designs in the selected set, which leads to a large selected set.

We regard the truly top 50 feasible designs as good enough. For different alignment probability, we use Eq. (5.6) in Section V.1 to calculate the size of the selected set such that there is at least 1 truly good enough and feasible design in the selected set with a probability no less than the required alignment probability. These sizes are listed in Table 8.10.

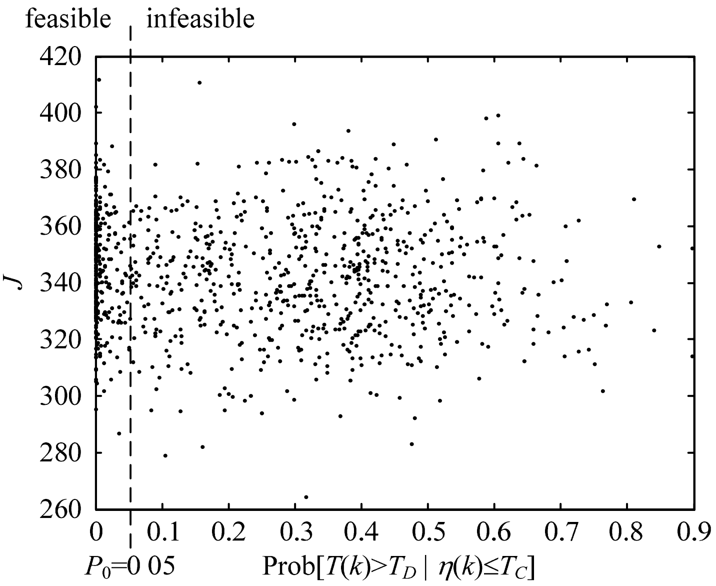

Fig. 8.15. The true performance and feasibility of 1000 randomly sampled designs

Table 8.10. Selected set size *s* for the remanufacturing system

Required AP	*s*
≥0.50	10
≥0.70	16
≥0.95	39

Suppose the required AP is 0.95. Since the designs are randomly sampled. The alignment level between the selected set and the good enough set might be different in different experiments. We show one instance in Table 8.11, where only the indexes of the designs are shown. In this instance there are 3 truly good enough and feasible designs found. Compared with the brute force simulation, which needs to accurately evaluate the performance and the feasibility of all the 1000 designs, COO saves the computing budgets by 25 folds in this example, by reducing from $N = 1000$ to $s = 39$.

Table 8.11. One instance of the alignment between *G* and *S*

Set	Plans
S	{404, 858, 744, 766, 245, ***763***, 241, 466, 48, 532, 408, 906, 186, 39, 597, 577, 589, 351, 567, 406, 948, 882, 988, 402, 924, 464, 667, 530, 984, 906, 633, 357, ***317***, 907, 119, 305, ***857***, 737, 646}
G	{90, 270, 450, 630, 810, 990, 157, 337, 517, 697, 877, 1, 194, 374, 554, 734, 914, 136, 316, 496, 676, 856, 29, 209, 389, 569, 749, 929, 184, 364, 544, 724, 904, 43, 223, 403, 583, ***763***, 943, 146, 326, 506, 686, 866, 137, ***317***, 497, 677, ***857***, 143}
$G \cap S$	***{317, 763, 857}***

3.3 Application of VOO

As aforementioned, in practice we sometimes do not know the appropriate value of P_0, which is the threshold for the probability that a maintenance time exceeds the given value. What we know is that the maintenance time is an important aspect of the system performance that should be considered during the optimization. We will here regard both the maintenance cost and the maintenance time as objective functions. More specifically we have two objective functions. One is the probability that the maintenance time exceeds a given limit, i.e.,

$$J_1(C, \Delta I) = \text{Prob}\left[T(k) > T_{\text{D}} / \eta(k) \leq T_{\text{C}}\right].$$

The other one is still the maintenance cost, i.e.,

$$J_2(C,\Delta I) = \alpha\sum_{i=1}^{n}\sum_{j=1}^{m} C_{i,j} + \sum_{i=1}^{n}\sum_{j=1}^{m}\Delta I_{i,j} + \beta\sum_{i=1}^{n}\sum_{j=1}^{m} I_{i,j}.$$

Then we have a two-objective optimization problem

$$\min_{C,\Delta I} J(C,\Delta I) = \min_{C,\Delta I}\left(J_1(C,\Delta I), J_2(C,\Delta I)\right)^{\tau}.$$

Both objective functions can only be accurately evaluated by simulations. We will apply the vector ordinal optimization introduced in Chapter IV to solve this problem.

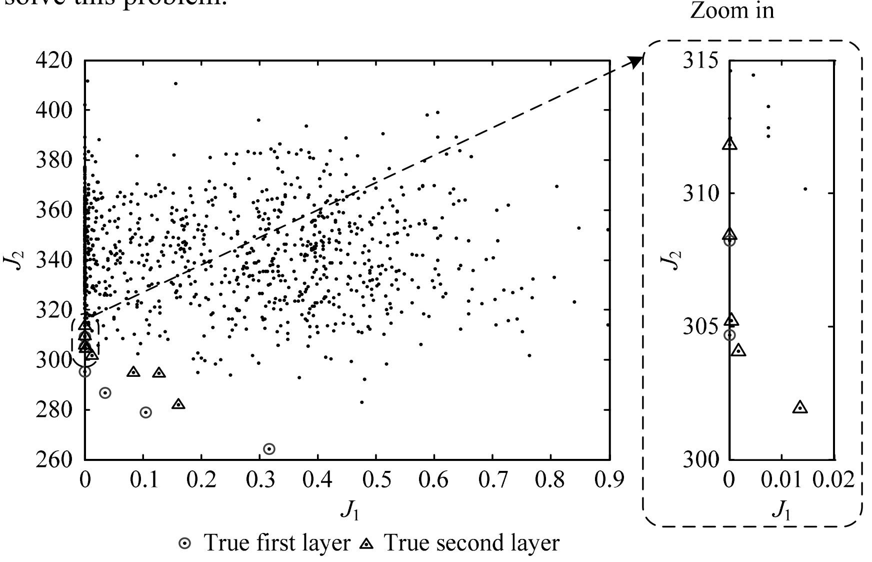

Fig. 8.16. The true performances of the designs

Since we have already used brute force simulation to obtain the true performance and feasibility of 1000 randomly sampled designs in Fig. 8.15, we show the first two layers of these 1000 designs in Fig. 8.16. There are 6 designs in the first layer (marked by circles), and 8 designs in the second layer (marked by triangles) shown in Fig. 8.16. We also show the observed performance curve in the vector case (VOPC) in Fig. 8.17. Of course, in practice we do not know this true VOPC. Instead, we use a crude model (a single replication) to get the rough estimation of the performance. Fig. 8.18 shows one instance, where there are 4 designs in the observed first and the observed second layer, respectively, which is different from Fig. 8.16. The VOPC is similar to Fig. 8.17, which belongs to the steep type. We estimate the noise level by 10 independent simulations of a design, and find the normalized noise level is 0.1061 for J_1 and 0.0078 for J_2, which is a small noise level. By looking at the VOO-UAP table Table 4.1 in Chapter IV, we find the coefficients in the regression function are: $Z_1 = -0.7564$, $Z_2 = 0.9156$, $Z_3 = -0.8748$, and $Z_4 = 0.6250$. Define the designs in the truly first two layers as good enough designs (there are 14 design in total). Because the VOO-UAP table is developed for 10,000 designs and 100 layers in all,

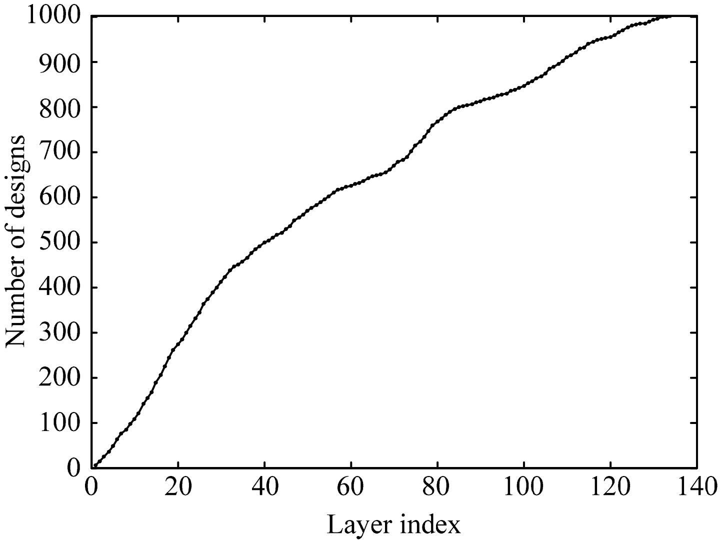

Fig. 8.17. The true VOPC

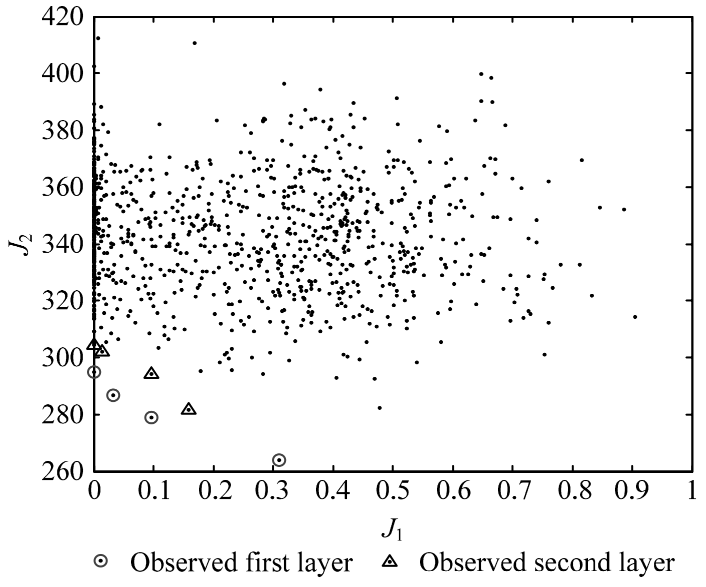

Fig. 8.18. The observed performance of the remanufacturing system

we need to normalize the values of g, s, and k before the calculation. There are 134 layers in Fig. 8.17[8], so $g'=[100/134\times2]=1$, $k'=10000/1000\times k=10k$ ($1\le k\le14$), and $s=[134/100\times s']$. For different values of k, we denote the values of s predicted in this way as $\hat{s}_1$. Since the noise levels in J_1 and J_2 are smaller than 0.5, which is used in the VOO-UAP table, the values of $\hat{s}_1$ might be conservative. So, we also use some simulation-based method to obtain less conservative prediction of the values of s, which are denoted as $\hat{s}_2$. More details of this simulation-based method are presented in details later.

On the other hand, we use 1000 independent simulations to estimate how many observed layers are needed to contain some truly good enough designs with a probability no less than 0.95. Let these values be s^*. For different values of k, we show the values of s^*, the predicted values of s using the regression function (denoted as $\hat{s}_1$), and the predicted values of s using simulation-based method (denoted as $\hat{s}_2$) in Table 8.12.

Table 8.12. The predicted value and true value of s

k	s^*	$\hat{s}_1$	$\hat{s}_2$	$\left\lvert\bigcup_{i=1}^{\hat{s}_2}\mathcal{L}_i\right\rvert$
1	1	7	1	6
2	1	11	1	6
3	1	15	2	14
4	1	20	2	14
5	2	24	2	14
6	2	28	3	24
7	2	32	3	24
8	3	36	4	36
9	3	40	5	48
10	5	44	5	48
11	6	48	6	62
12	8	52	8	85
13	9	56	9	96
14	13	59	12	141

Table 8.12 shows that the predicted value $\hat{s}_1$ is always an upper bound of the true value s^*, but the difference might be large. This means $\hat{s}_1$ is conservative, because the small noise level in the VOO-UAP table is 0.5,

[8] The number of observed layers is a random number, which varies around the true number of layers.

but the noise level in this problem is 0.1061 and 0.0078, respectively. To obtain a less conservative estimate of s^*, we use the following method. Based on the rough estimate obtained through a single simulation, regard the estimate as the true value, find the observed first two layers, and regard these designs as the "truly good enough designs". Then add the normally distributed noise $N(0,0.2122^2)$ and $N(0,0.0156^2)$ [9], and find the observed first s layers. Repeat this procedure for 1000 times, and obtain an estimate of the alignment probability for each (s, k) (in which $s = 1,2,\ldots 20$, and $k = 1,2,\ldots,14$). Select the smallest s such that the alignment probability is no less than 0.95 as the estimate of the true values s^*, and denote as $\hat{s}_2$, also shown in Table 8.12. We can see that $\hat{s}_2$ is a good estimate of s^*, and a tight upper bound of s^*, except for the case of $k = 14$. In the case of $k = 14$, the difference is only 1. From Table 8.12, we can see that VOO saves the computing budgets by at least one order of magnitude in most cases. Only when $k = 14$, the number of designs in the observed first $\hat{s}_2$ layers increases drastically. That is because it is a difficult job to find *all* the designs in the truly first two layers.

To summarize, the application procedure of VOO in this remanufacturing system is as follows.

Box 8.3. Application procedure of VOO in the remanufacturing system

Step 1: Randomly sample N designs.
Step 2: Use the crude model to quickly estimate the performance of the designs.
Step 3: Layer down the designs according to the observed performance.
Step 4: User defines the good enough set and the required alignment level k.
Step 5: Estimate the observation noise level and the problem type (VOPC).
Step 6: Use the VOO-UAP table in Section IV.2 to calculate the number of observed layers to select.
Step 7: Select the observed first s layers as the selected set.
Step 8: Then the VOO theory ensures that there are at least k truly good enough designs in the selected set with high probability.

[9] The standard deviations of the additive noises are twice the standard deviations of the noises in the two objective functions, respectively.

3.4 Conclusion

In this section, we consider the performance optimization of a remanufacturing system. By formulating the requirement on the maintenance time as a constraint or an objective function, we have a simulation-based constrained optimization problem or a two-objective optimization problem. We apply COO and VOO to solve the problems, respectively. Besides the general application procedure of COO and VOO as introduced in Chapter V and IV, we also discuss how we can incorporate the problem information into a feasibility model, using a machine learning method in COO; and how we can use problem information to improve the estimate of the number of observed layers to select in VOO. In both problems, COO and VOO save a lot of computing budgets. Note that we only use blind pick in Section 3.3. By using horse race selection rule, we can further improve the performance of COO by selecting a smaller number of predicted feasible designs. Interested readers may refer to (Song et al. 2005b) for more details.

Exercise 8.1: Can we apply COO to solve simulation-based multi-objective optimization problems? If so, please explain how. If not, please explain why.

Exercise 8.2: Can we apply VOO to solve simulation-based constrained optimization problems? If so, please explain how. If not, please explain why.

4 Witsenhausen problem

A celebrated problem in system and control is the so-called Witsenhausen problem. In 1968, H. S. Witsenhausen (Witsenhausen 1968) posed an innocent looking problem of the simplest kind. It consists of a scalar linear dynamic discrete time system of two time stages (thus involving two decisions at time stages one and two). The first decision is to be made at time one with perfect knowledge of the state, and there is a quadratic cost associated with the decision variable. The second decision can only be made based on noisy Gaussian observation of the state at time stage two, however, there is no cost associated with the decision. The performance criterion is to minimize the quadratic terminal state after the two decisions. Thus, it represents the simplest possible Linear-Quadratic-Gaussian (LQG) control problem except for one small detail: ***Instead of the usual assumption of one centralized decision maker who remembers at time stage two what s/he knows at time stage one, we do not have perfect memory or recall.*** In fact, we have a decentralized team problem with two decision makers (DMs), DM1 and DM2 who do not have complete knowledge of what the other knows. Here the possibility for optimization is clear. DM1 knows the state of the system perfectly. S/he can simply use his/her control variable to cancel the state perfectly and leave DM2 nothing more to do. However, his/her action entails a cost. On the other hand, DM2 has no cost to act, but, without perfect memory, s/he has no perfect knowledge of the state of the dynamic systems at time stage two. A simple approach would be to strike a compromise using linear feedback control law for each decision maker, which is also known to be optimal under the traditional centralized LQG system theory for problems with perfect memory. In fact, it is easy to prove that such a solution is a person-by-person optimal solution in equilibrium, i.e., if DM1 fixes his/her linear feedback control law, the best response by DM2 is a linear feedback control law and vice versa. However, Witsenhausen demonstrated that, without perfect memory, there exists a nonlinear control law for both DM1 and DM2, which involves signaling by DM1 to DM2, using its control action (The idea of signaling will be explained in more details in Section 4.1.), that outperforms the linear person-by-person optimal control law. In other words, the Witsenhausen problem presents a remarkable counterexample which shows that the optimal control law of LQG problems may not always be linear when there is imperfect memory. At the time, this was totally surprising since the problem seemed to possess all the right mathematical assumptions to permit an easy optimal solution. However, the globally optimal control law for such a simple LQG problem (or team decision problem) was unknown. The discrete version of the problem was known to be NP-complete (Papadimitriou and Tsitsiklis 1986). Many attempts and papers on the problem followed in the next thirty and more years before the problem was understood and a numerical solution of the globally optimal control law obtained in (Lee et al. 2001).

The difficulty of the problem constitutes the essence of information structure (who knows what and when) in decentralized control, which is a subject worthy of a separate book. We shall not go into the matters here. However, we shall use the Witsenhausen problem here to illustrate the process of search in the space of control laws, using OO to get good enough solutions. This is because there are so much data accumulated with this problem and we can easily assess the "good enough"-ness of any results thus obtained via OO.

First, we show the mathematical problem formulation in Section 4.1. Since the optimal control laws associated with the Witsenhausen problem have not been obtained analytically yet, it is important to discover the structure of the space of the control laws numerically. This is a common problem faced by many practical engineering problems, where finding the optimal design needs tremendous computing time, even if not computationally infeasible. By using OO, we were able to discover some structure information of the design space in the Witsenhausen problem efficiently, based on only noisy performance observation. The information not only produced a pair of control laws that were 47% better than the best solution known by that time (Banal and Basar 1987), but also helps to finally achieve the best-so-far numerical solution (Lee et al. 2001). We introduce the details for this also in Section 4.1. In Section 4.2, we consider the constraint on memory space when solving the Witsenhausen problem, and show how to find simple and good enough control laws using OO and OBDD, which were introduced in Chapter VI. With minor performance degradation (less than 5%), we save the memory space to store the control law by over 30 folds. We make a brief conclusion in Section 4.3.

4.1 Application of OO to find a good enough control law

The Witsenhausen problem can be described as follows. It is a two-stage decision making problem. At stage 1, we observe the initial state of the system x. Then we have to choose a control $u_1=\gamma_1(x)$ and the new state will be determined as $x_1=x+u_1=\mathrm{x}+\gamma_1(x)$. At stage 2, we cannot observe x_1 directly. Instead, we can only observe $y=x_1+v$, where v is the additive noise. Then we have to choose a control $u_2=\gamma_2(y)$ and the system state stops at $x_2=x_1+u_2$. The cost function is $E[k^2(u_1)^2+(x_2)^2]$ with $k^2>0$ as a constant. The problem is to find a pair of control functions (γ_1, γ_2) which minimize the cost function. The trade off is between the costly control of γ_1 which has perfect information and the costless control γ_2 which has noisy information. We consider the famous benchmark case when $x\sim N(0,\sigma^2)$ and $v\sim N(0,1)$ with $\sigma = 5$ and $k = 0.2$.

Witsenhausen made a transformation from (γ_1, γ_2) to (f, g), where $f(x)=x+\gamma_1(x)$ and $g(y)=\gamma_2(y)$. Then the problem is to find a pair of functions (f, g) to minimize $J(f, g)$ where

$$J(f,g)=E\left[k^2\left(f(x)-x\right)^2+\left(f(x)-g\left(f(x)+v\right)\right)^2\right]. \tag{8.4}$$

The first term in Eq. (8.4), $E[k^2(f(x)-x)^2]$, represents the cost shouldered by player one in the first time stage, so it is also called the stage one cost. The second term, $E[(f(x)-g(f(x)+v))^2]$, represents the cost shouldered by player two in the second time stage, so it is also called the stage two cost. Witsenhausen (Witsenhausen1968) proved that: 1) For any $k^2>0$, the problem has an optimal solution. 2) For any $k^2<0.25$ and $\sigma=k^{-1}$, the optimal solution in linear control class with $f(x)=\lambda x$ and $g(y)=\mu y$ has $J^*_{\text{linear}}=1-k^2$, and $\lambda=\mu=0.5\left(1+\sqrt{1-4k^2}\right)$. In the benchmark case that we consider, k=0.2, $J^*_{\text{linear}}=0.96$. 3) There exist k and σ such that J^*, the optimal cost, is less than J^*_{linear}, the optimal cost achievable in the class of linear controls. Witsenhausen gave the following example. Consider the design: $f_W(x)=\sigma\,\text{sgn}(x)$, $g_W(y)=\sigma\tanh(\sigma y)$, where sgn(•) is the sign function, then the cost function J is J_W=0.4042. 4) For given $f(x)$ satisfying $E[f(x)]=0$ and $\text{var}[f(x)]\leq 4\sigma^2$, which are the conditions that the optimal $f^*(x)$ should satisfy, the optimal g^*_f associated with function f is

$$g^*_f=\frac{E\left[f(x)\varphi\left(y-f(x)\right)\right]}{E\left[\varphi\left(y-f(x)\right)\right]}, \tag{8.5}$$

where $\varphi(\bullet)$ is the standard Gaussian density function.

Now the problem becomes that of searching for a single function f to minimize $J\left(f,g^*_f\right)$. Although the problem looks simple, no analytical method is available yet to determine the optimal f^*. The numerical optimal solution only came after over thirty years later and after many attempts (Lee et al. 2001). In the following, we will demonstrate how we should apply OO to search for good control laws for the Witsenhausen problem. Before we present the numerical details, we should discuss what a crude model is and how we can apply OO to discover the property of the good enough designs, which helps to narrow down the search space.

4.1.1 Crude model

Following the properties of the optimal control laws shown by Witsenhausen, each "design" in the Witsenhausen problem is a control function f, which satisfies $E[f(x)]=0$ and $\text{var}[f(x)]\leq 4\sigma^2$. Because f is in general a one-dimension real function, and there are in principle infinite number of such functions, it is important to find an appropriate representation for such functions in a digital computer. The idea is to discretize the function f. Lee et al. showed that it is reasonable to assume the optimal function f^* is symmetric about the origin, i.e., $\gamma_1(y_1) = -\gamma_1(-y_1)$ (Lee et al. 2001). In the following discussion, we only consider $f(x)$ for $x\geq 0$. We divide the x-space

$[0,\infty)$ evenly in probability, i.e., we divide x-space into n intervals, $I_1,\ldots,I_n$, where $I_i=[\sigma t_{(0.5+0.5(i-1)/n)}, \sigma t_{(0.5+0.5i/n)})$, t_α is defined by $\Phi(t_\alpha)=\alpha$ where Φ is the standard normal distribution function. Prob$[x\in I_i]=0.5/n$ because x has a normal distribution $N(0,\sigma^2)$. Then for each interval I_i, a control value f_i is uniformly picked from $(-3\sigma, 3\sigma)$, i.e., $f_i\sim U(-15, 15)$. To calculate the performance $J\left(f,g_f^*\right)$, we should calculate the optimal associated function g_f^* through Eq. (8.5) and then we will obtain $J\left(f,g_f^*\right)$ through Eq. (8.4). However, both Eq. (8.4) and Eq. (8.5) involve expectations, which mean a large number of Monte Carlo simulations might be required to calculate the performance J accurately. Actually based on the results described in this subsection, (Lee et al. 2001) developed a step-function representation of the function f and then obtained a way to calculate J through numerical integration instead of Monte Carlo simulation. Although numerical integration is much faster than Monte Carlo simulation, it requires a long-time numerical integration to make the result very accurate. We will come back to this at the end of this subsection. Right now, let us assume Monte Carlo simulation is the only way to accurately evaluate J, which is true when the results of this subsection were developed in 1999.

The question now is how to find a crude model which is computationally fast and can give a rough estimate of J. We simplify the calculation from two aspects. First, instead of calculating the accurate g_f^*, we calculate an approximation of g_f^*,

$$\hat{g}_f=\frac{\sum_{i=1}^{100} f(x_i)\varphi\left(y-f(x_i)\right)}{\sum_{i=1}^{100}\varphi\left(y-f(x_i)\right)}.$$

Second, instead of using a large number of Monte Carlo simulations to accurately calculate $J\left(f,\hat{g}_f\right)$ through Eq. (8.4), we use only 100 replications to get an estimate $\hat{J}\left(f,\hat{g}_f\right)$. In this way, we get a crude model $\hat{J}\left(f,\hat{g}_f\right)$ of the true performance $J\left(f,g_f^*\right)$.

4.1.2 Selection of promising subsets

After the discretization of function f, the design space in the Witsenhausen problem is still extremely large. To overcome this difficulty, our basic idea is to divide the entire design space into smaller subsets and choose promising subsets for further searching. The No-Free-Lunch Theorem tells us that every other optimization method can be as efficient (inefficient) as blind pick, without any problem information. To achieve a higher efficiency than that of blink pick, we need to discover some structure information of the design space, e.g., which subset contains more good enough designs than others. In particular, if we sample a set of designs for their performances (however noisily or approximately), we should be able to catch a glimpse, from the samples, of what are "good" subsets to search, and gradually restrict the search there. This is like traditional hill climbing, except that we move from one subset to another instead of moving from point to point in the search space. The key question here is to establish a procedure of comparing two subsets based upon sampling and then to narrow down the search space step by step. We will show how OO helps to do this comparison in this subsection. By the OO-based comparison, we find three restrictions which narrow down the search space to a subset that contains more good enough designs than without the restrictions. The three restrictions are: 1) For each interval I_i, control f is in $(-0.5\sigma, 2.5\sigma)$ because we are searching primarily in the positive quadrant; 2) f is a non-decreasing function; 3) f has two steps. In the rest of this subsection, we will use numerical results to show how these three restrictions are discovered, and how this finally helps us to find a control law 47% better than the best solution known by that time, and later on helps to discover the best-so-far control law. First, we start from a general discussion on how to narrow down the search space using OO.

The idea behind narrowing down the search space is to identify which subset of the search space contains more good enough designs than the others. Of course, if we know the true performance of all the designs, this problem will be trivial. However, in practice we only have noisy performance observations and can only do this performance estimation for a small portion of the entire design space. So we need a function to represent the goodness of a subset, i.e., the number of good enough designs in this subset, and we hope this function can be efficiently estimated when only the noisy performance observations of some of the designs are available. The performance distribution function satisfies these constraints. Suppose Θ_1 and Θ_2 are two subsets of a large design space Θ. If we know the true performance of all the designs in Θ_1 and Θ_2, we can obtain a performance density function (Fig. 8.19(a)), which looks like a probability density function, by discretizing the performance into many small intervals, counting the number of designs the performances of which fall in each interval, and normalizing this number by the total number of designs. By integrating the performance density function, we obtain the performance distribution function (PDF) (Fig. 8.19(b)), which is non-decreasing and looks like a probability distribution function. Now, suppose we know the PDF of Θ_1 is F_1(t) and the PDF of Θ_2 is F_2(t), Let us focus on the top-5% designs in each subset. If $F_1(t_1)=0.05$, $F_2(t_2)=0.05$, and $t_1<t_2$, (as shown in Fig. 8.20) which means the top-5% designs in Θ_1 are with better performances than those in Θ_2. Then we should continue our search in Θ_1. Of course, in practice we only have the estimate of F_1 and F_2. As we will demonstrate by the numerical examples in the following, however, the estimate of F_1 and F_2 still helps us to find the promising subset.

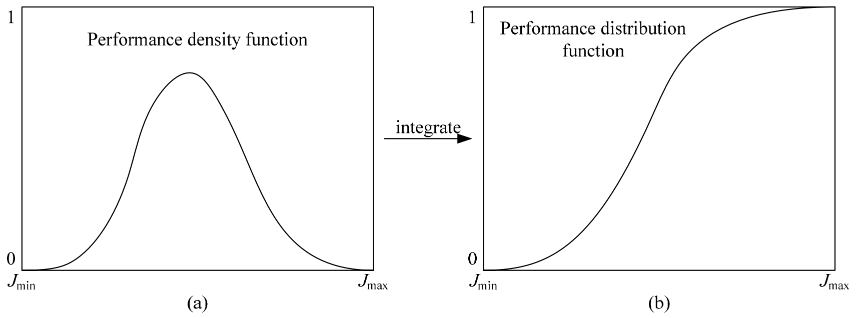

Fig. 8.19. (a) Performance density function and (b) performance distribution function

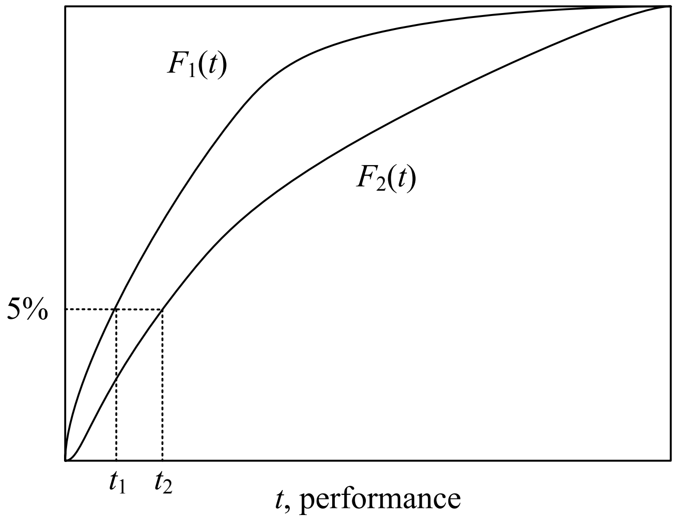

Fig. 8.20. Comparing two subsets based on PDFs

Now we are ready to summarize our sampling and space-narrowing procedure. For a search space Θ, we first define two or more subsets (there might be intersection among these subsets) and find the corresponding observed PDFs. By comparing the observed PDFs, we can estimate which subset(s) is (or are) good. We can then further narrow down our search into smaller subsets. In the following, we demonstrate how this procedure helps to discover the three properties of the good designs in the Witsenhausen problem and finally a design which was the state of the art when the result was published in year 1999 (Deng and Ho 1999).

The number of intervals, n, used in the construction of the discretized f, determines the size of the search space. In principle, n can be any positive integer. However, for a large n, the designs with good performances are only a very small portion of the entire search space. In this case, it is very difficult to find a good enough design. The first question here is what the appropriate value of n should be. For values of n=1, 2, 5, and 10, we randomly pick 5000 functions f in each case, use the crude model in Section 4.1 to estimate the performances, and observe the PDF respectively (shown in Fig. 8.21). Since we are doing minimization, we only care about the performances of the top-5% designs in each case. In Fig. 8.21, we only show the part of the PDFs that are close to the origin. There are 4 curves, from the left to the right, representing the PDFs of the subsets when n=1, 2, 5, and 10, respectively. We compare the performance of the top-5%-th design in each subset and find that the cases of n=5 and 10 are of much larger costs (both greater than 1.0) than those of n=1 and 2 (both smaller than 1.0). Thus, n=1 and 2 are better subsets than n=5 and 10. The problem is the cases of n=1 and 2 are indistinguishable. Because if we only care about top-5% designs in both cases, many top-5% designs in the case of n=2 are with larger costs than the case of n=1. However, if we consider the top-0.1% designs, the case of n=2 is better than n=1. Through this comparison, we find that n=1 or 2 are good choices, but we cannot determine which one is better yet. The fact that n=1 is a good choice indicates that the class of constant control functions (with discontinuity at the origin due to symmetry) is a good representation of the search space for control function f. This means that the pair of controllers described by Witsenhausen which outperform the optimal linear control law are already very good. In addition, by comparing the observed best f among the randomly sampled 5000 functions $f$or each n, we observe that the control values of the observed best controllers for different n are located in [−2, 12). Based on this observation, we make the following restriction (R1): *For each interval I_i, control f is in $(-0.5\sigma, 2.5\sigma)$, i.e., $f \sim U(-2.5,12.5)$.*

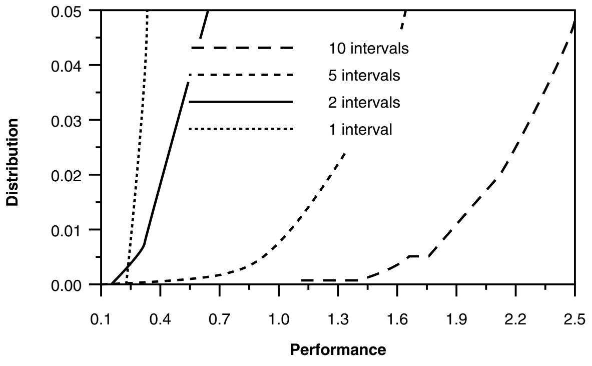

Fig. 8.21. Observed PDFs when there are n=1, 2, 5 and 10 intervals in $f(x)$, $x\geq 0$ (Deng and Ho 1999) © 1999 Elsevier

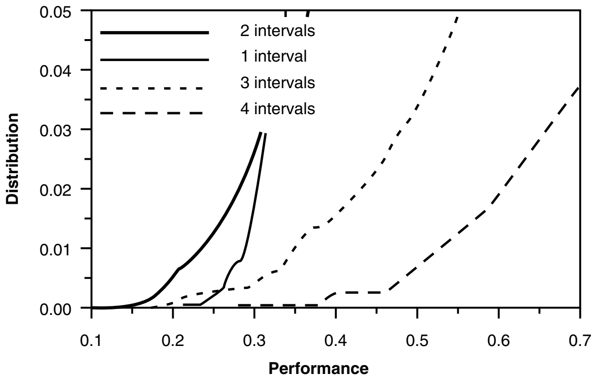

Fig. 8.22. Observed PDFs with restriction R1 (Deng and Ho 1999) © 1999 Elsevier

With this restriction, we repeat our experiment for n=1, 2, 5, and 10 and find the cases of n=1 and 2 still outperform the cases of n=5 and 10. We then do a further comparison among the cases of n=1, 2, 3, and 4. We show the observed PDFs and the observed best-control functions in Fig. 8.22 and Fig. 8.23. In Fig. 8.22, the top-5%-th design in the cases of n=1 and 2 are of costs smaller than 0.4, which is smaller than the costs of the top-5%-th designs in the cases of n = 3 and 4 (with costs larger than 0.5). If we compare the performances of the top-3%-th designs in each case, the case of n = 2 will be the best subset among the four cases. A more interesting phenomenon we may observe from Fig. 8.23 is that the observed best controllers for both n = 3 and 4 have the two-interval shape as the one of n = 2. All these observations indicate that the right direction of search should be toward the two-interval functions. Since the observed best controllers (in Fig. 8.23) display some increasing property, we make a further restriction (R2): *The control f is a non-decreasing function in (–0.5σ, 2.5σ).*

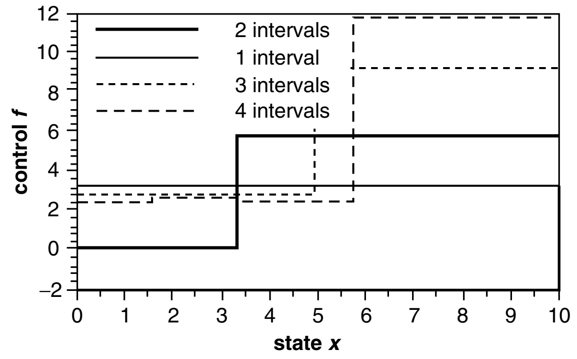

Fig. 8.23. Observed best control *f* with restriction R1 (Deng and Ho 1999) © 1999 Elsevier

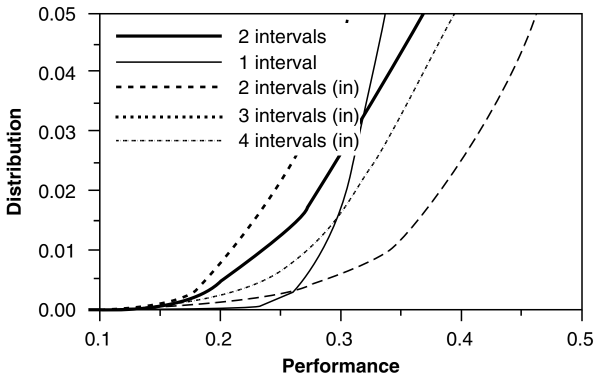

Fig. 8.24. Observed PDFs with restriction R2 (Deng and Ho 1999) © 1999 Elsevier

To test whether restriction R2 helps us to find subsets containing more good designs, we compare the observed PDFs before and after the restriction R2 is applied in Fig. 8.24. The curves with the legend "interval(in)" are those with restriction R2. Fig. 8.24 shows that with restriction R2, the top-5% designs in the two-interval controllers have the best performances. This indicates that the specification of the non-decreasing control function is in the right direction. Actually it was shown by Witsenhausen that the optimal function *f* should be non-decreasing (Witsenhausen 1968). It is conceivable that the optimal control function may possess significant discontinuity. Thus, the 3rd restriction (R3) is made as follows: *The control f is a two-value non-decreasing step function in (–0.5σ, 2.5σ).*

In the previous experiments, to make a quick estimate of the PDF for each value of *n*, when the number of intervals n is given, we fix the discretization of the *x*-space as explained in Section 4.1. For example, for *n*=2, we fix the jump points at $x=\sigma t_{0.75}$. Now, we have already identified that n=2 is a good choice, as stated in R3, so we will determine the jump point of the two-value functions. As Fig. 8.23 shows, the jump point may not be at $\sigma t_{0.75}$. We consider 10 possible jump points: $\sigma t_{0.55}$, $\sigma t_{0.60}$, $\ldots\sigma t_{0.95}$. For each jump point, we randomly sample 5000 functions f that satisfies R3. The observed PDFs, associated with different jump points, are presented in Fig. 8.25. In Fig. 8.25, the legend "2 int. (a)" represents the jump point as σt_a. We see that the best jump point is around $\sigma t_{0.90}$. The best observed control function f among 5000 samples in the space associated with $\sigma t_{0.90}$ is

$$f_{DH}(x)=\begin{cases}3.1686, & 0\le x<6.41,\\ 9.0479, & x\ge 6.41.\end{cases}$$

The subscript "DH" is to denote that this function was first found by M. Deng and Y.-C. Ho in 1999 (Deng and Ho 1999). We use 10000 replications to obtain an accurate estimate of the true performance of this function, and obtain the value 0.1901 with variance 0.0001, which is 47% better than the best solution know by that time which was found by Banal and Basar with performance J_{BB}=0.3634 (Banal and Basar 1987).

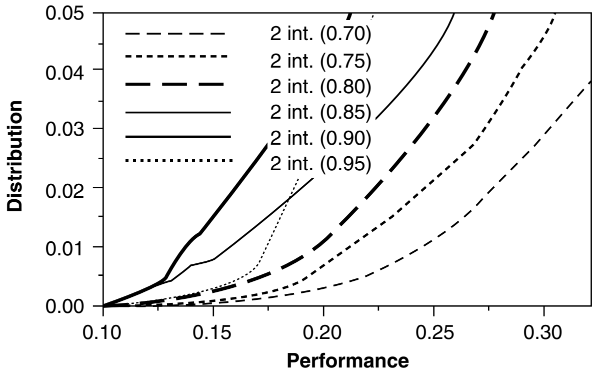

Fig. 8.25. Observed PDFs with restriction R3 (Deng and Ho 1999) © 1999 Elsevier

The after-the-fact reasoning behind the superiority of the two-value controllers is as follows. Witsenhausen proposed the signaling concept when he reported this famous counter example in (Witsenhausen 1968). The idea is that DM1 knows the state of the system perfectly but has an action cost. So instead of using his/her own control variable to cancel the state perfectly, DM1 cancels (or enhances) parts of the state (x), which makes the state of the system concentrated on either a given negative or positive point x_1. DM1 uses x_1 as a signal to tell DM2 how to set his/her control variables, such as positive or negative values of x_1. Under moderate noise conditions, DM2 can ascertain the sign of x_1 with high probability. DM2 has no action cost. If DM2 can interpret DM1's signal x_1 correctly, and thus takes the correct action to cancel almost all the state x_1, the resulting state of the system, x_2, will cause little cost, i.e., a small stage 2 cost. However, DM2 only has noisy observation of x_1, and may misinterpret x_1. To reduce the probability of misinterpretation, DM1 needs to take large actions to make the signaling levels far apart from each other, which causes a large stage 1 cost. Finding the optimal $f^*(x)$ in the class of step functions (as used in this section) amounts to finding the optimal number of steps/intervals in $f(x)$ and their placements so as to balance the tradeoffs between the first and second stage costs. The step function f_W proposed by Witsenhausen is a one-interval function (the single jump point is at x=0 and with signaling level σ). Banal and Basar further optimized the signaling level of this one-interval function and obtained f_{BB} (the single jump point is still at x=0 but with signaling level $\sigma\sqrt{2/\pi}$). The signaling scheme in f_{DH} allows DM1 to use four signal levels, i.e., more positive (9.0479), less positive (3.1686), less negative (–3.1686), and more negative (–9.0479). Hence, there is a reduction in the magnitude of $(x\text{-}f_{DH}(x))$. Meanwhile, the signaling levels are placed sufficiently far apart so that DM2 can still distinguish DM1's signal with small errors.

In the above discussion, we continuously narrow down the search space by comparing the observed PDFs of different subsets. The ideas of goal softening and ordinal comparison allow us to discover some properties of the good designs, i.e., the three restrictions we found. This finally led us to a design 47% better than the best solution known by that time (Banal and Basar 1987) when this result was published in 1999. These results indicate that a step function may be an appropriate representation of the function f. This idea was further explored in (Lee et al. 2001). In (Lee et al. 2001), by describing the function f as a step function, Lee et al. achieved a fast and accurate computational scheme for the cost J which eliminates the need of simulation, but requires numerical integration. Also they observed that the jump points should be located around the average of two adjacent values of function f (which is also called the signaling levels). Furthermore, additional improvement can be made by adding small segments to approximate a slight slope for each step in function f. Finally they achieved the following function f

$$f_{\text{LLH}}(x) = \begin{cases} 0.00 & 0.00 \le x < 0.65 \\ 0.05 & 0.65 \le x < 1.95 \\ 0.10 & 1.95 \le x < 3.25 \\ 6.40 & 3.25 \le x < 4.58 \\ 6.45 & 4.58 \le x < 5.91 \\ 6.50 & 5.91 \le x < 7.24 \\ 6.55 & 7.24 \le x < 8.57 \\ 6.60 & 8.57 \le x < 9.90 \\ 13.10 & 9.90 \le x < 11.25 \\ 13.15 & 11.25 \le x < 12.60 \\ 13.20 & 12.60 \le x < 13.95 \\ 13.25 & 13.95 \le x < 15.30 \\ 13.30 & 15.30 \le x < 16.65 \\ 19.90 & 16.65 \le x. \end{cases}$$

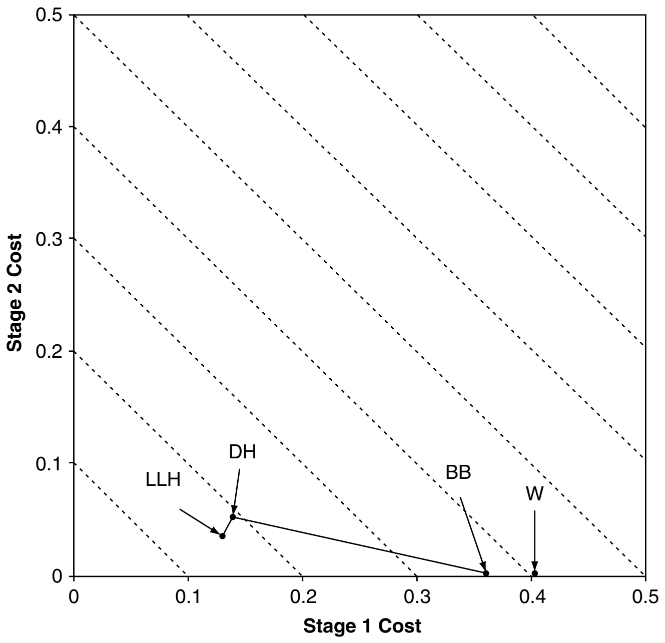

Fig. 8.26. Historical improvements on the Witsenhausen problem (benchmark: $k = 0.2$ and $\sigma = 5$) (Lee et al. 2001) © 2001 IEEE

The corresponding cost is $J_{\mathrm{LLH}} = 0.167313205338$. This is the best-so-far solution in the past over-thirty years. In summary, we show the historical improvements on the Witsenhausen problem in Fig. 8.26, from which we can see J_{LLH} has a good balance between stage 1 cost and stage 2 cost. It was claimed that this is the "optimal" solution in the sense that all properties of the optimal solution have been discovered, any further improvement is relatively small and can only be achieved numerically, say by further dividing each step of function $f_{\mathrm{LLH}}(x)$ into smaller steps and using local search to improve.

4.2 Application of OO for simple and good enough control laws

Although f_{LLH} is the best-so-far solution to the Witsenhausen problem, it is obviously more complex than the 1-step function f_{W} and f_{BB}, the 2-step function f_{DH}, and the following 3.5-step function $f_{3.5}$

$$f_{3.5}(x) = \begin{cases} 0 & 0 \le x < 3.25 \\ 6.5 & 3.25 \le x < 9.90 \\ 13.2 & 9.90 \le x < 16.65 \\ 19.9 & 16.65 \le x. \end{cases}$$

from which f_{LLH} was obtained by adding small segments to each step. Each time when a better $f(x)$ was reported, the incremental improvement becomes smaller and smaller (from $J_{\mathrm{W}} = 0.4042$ to $J_{\mathrm{BB}} = 0.3634$ to $J_{\mathrm{DH}} = 0.1901$ to $J_{3.5} = 0.1714$ to $J_{\mathrm{LLH}} = 0.1673$), and the function f becomes more and more complex. It seems that there is a trade-off between the performance of f and the complexity of f. Since we have not quantified the complexity of f, this statement is very informal. An interesting question is whether we can find simple f's with similar performance to the best-so-far solution. Though f_{W}, f_{BB}, f_{DH}, and $f_{3.5}$ are intuitively simpler than f_{LLH}, these functions were not obtained for being simple and good. It is not clear yet how to find a simple and good enough f in a systematic way. This is where the OO methodology of Chapter VI can help. In this subsection, we use the Kolmogorov complexity (KC) to measure the complexity of a function f. The KC of a function can hardly be calculated in general, but can be estimated through the OBDD-based representation of the function (where OBDD stands for Ordered Binary Decision Diagram). In Chapter VI, we combine OO and OBDD to get a systematic method of finding simple and good enough solutions. We will use this method to find a simple and good enough control law for the Witsenhausen problem. Comparing with the best-so-far function f_{LLH}, with minor performance degradation (within 5%), we reduce the complexity of f (i.e., the memory space to store f) by over 30 folds. Although in this specific example, most digital computers in engineering practice can store f_{LLH} with no difficulty, the importance of studying this problem is to demonstrate how to use the method introduced in Chapter VI to find simple and good enough designs, especially when the memory space is limited.

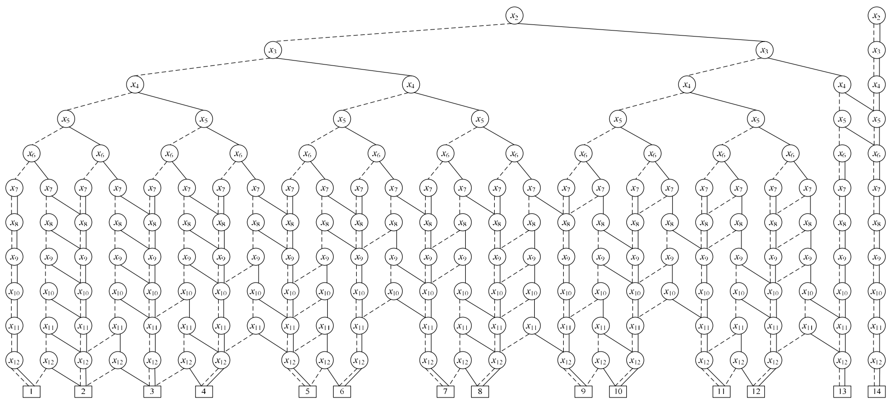

Fig. 8.27. One PROBDD that describes f_{LLH}. The 14 boxes in the bottom represent the 14 output in f_{LLH}, from 0.00 to 19.90, represented by 11-bit binary sequence

We start from quantifying the complexity of the best-so-far function f_{LLH}. As introduced in Chapter VI, KC supplies a measure of this complexity. The idea is to write a program to represent f_{LLH}. When the value of x is input, a computer should output the value of $f_{\mathrm{LLH}}(x)$ by executing this program, which should work for all the values of x. The length of the shortest program that can represent f_{LLH} in this way is defined as the KC of f_{LLH}. $f(x)$ is defined over all the real numbers. However, digital computers have only finite input. Lee et al. showed that (Lee et al. 2001) it is reasonable to approximate $f(x)$ by only focusing on the domain of $[-5\sigma, 5\sigma]$, i.e., $-25 \le x \le 25$, because $x \sim N(0,\sigma^2)$ and the value of $f(x)$ for $x \notin [-5\sigma, 5\sigma]$ are very insignificant to the overall cost objective. In the following discussion, when representing function $f(x)$ by a program, we only consider input $0 \le x \le 25$. In f_{LLH}, the input has a resolution of 0.01, thus we need $\lceil \log_2(2500) \rceil = 12$ bits to encode the input. As for the output, we can similarly calculate that $\lceil \log_2(1990) \rceil = 11$ bits are needed to encode the output. As explained in Chapter VI, we can also represent f_{LLH} by a Partially Reduced OBDD (PROBDD) with 12-bit input and 11-bit output. We show one such PROBDD in Fig. 8.27. There are 182 nodes (excluding the 14 boxes in the bottom) in this PROBDD. Following the calculations introduced in Chapter VI, we know it takes 24192 bits[10] to store this PROBDD. The details of the calculation are shown in Table 8.13. (In the table, f_{sg} is a *simple* and *good enough* solution which will be discussed later.) The size 24192 is much less than 57500, the size of lookup table representation.

Exercise 8.3: In the look-up table representation, we describe f_{LLH} by listing all the $(x, f_{LLH}(x))$ pairs in sequence. Please tell why 57500 bits all together is needed.

Table 8.13. The complexity and performance of the milestone f's

f	Base (input)	# of input bits	Base (output)	# of output bits	# of nodes	# of boxes	# of rules (r)	d	$4rd$ (bits)	J
f_W	–	0	1	3	0	1	6	4	96	0.4042
f_{BB}	–	0	0.0001	16	0	1	32	5	640	0.3634
f_{DH}	0.01	12	0.0001	17	32	2	132	7	3696	0.1901
f_{sg}	0.01	12	0.001	15	24	4	168	7	4704	0.1746
$f_{3.5}$	0.01	12	0.1	8	62	4	188	7	5264	0.1714
f_{LLH}	0.01	12	0.01	11	182	14	672	9	24192	0.1673

This justifies that PROBDD supplies a more compact representation of the function f_{LLH}. Since the KC of a function cannot be calculated in general, we suggest in Chapter VI to use "4*rd*" as an estimate of the KC of a function f, where r is the number of rules to implement the PROBDD that represents f, and d is the number of bits to encode each of the 4 elements in a rule. Following this measurement, we also estimate the KC of f_W, f_{BB}, f_{DH}, and $f_{3.5}$, and show the results in Table 8.13. We show the complexity and

[10] In (Jia 2006) and (Jia et al. 2006b), the complexity of f_{LLH} is estimated as 71311 bits, which is different from the results shown here. There are several reasons that cause the difference. First, (Jia et al. 2006b) used 15 bits to encode the input and 14 bits to encode the output. Second, due to more bits used in input and output, the PROBDD obtained in (Jia et al. 2006b) contains more nodes than the one shown in Fig. 8.27. Third, instead of using the formula "4*rd*" to estimate the complexity (Please refer to Chapter VI for more details), (Jia et al. 2006b) uses another formula to estimate the complexity. Although the values of the estimates are different, (Jia 2006) also estimates f_{sg}, which will be introduced later in this subsection, much simpler than f_{LLH}. At the end of this subsection, we will use an example to show how different computers may lead to different KCs. In practical application, this will not be a problem, as the computer is given and fixed.

performance of these functions in Fig. 8.28. This also justifies that 4*rd* is a reasonable estimate of the complexity of the function, because Fig. 8.28 shows that when the performance improves, the complexity also increases, which is consistent with our intuition.

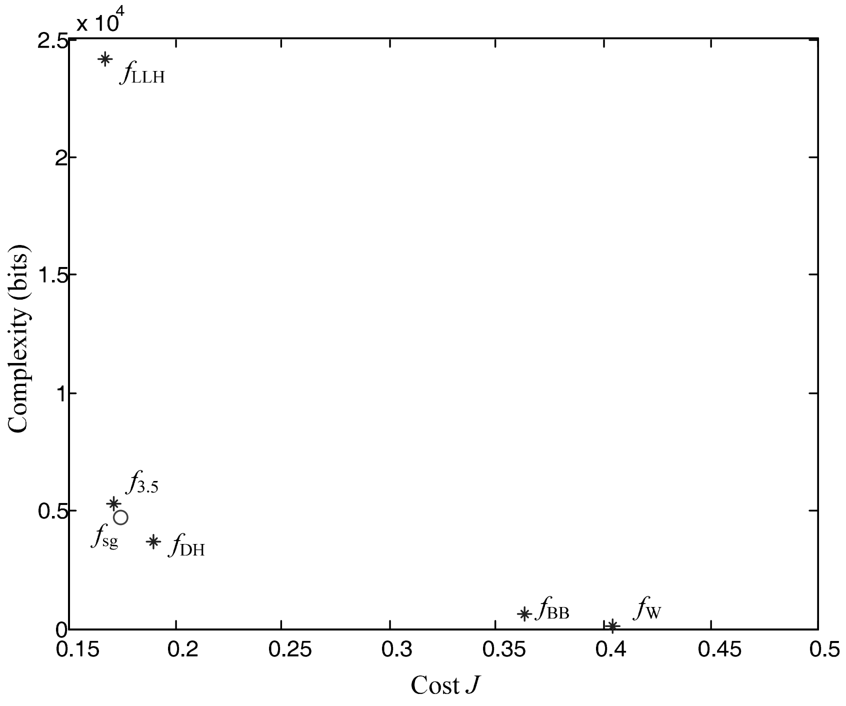

Fig. 8.28. The complexity and performance of the milestone *f* 's

After comparing milestone solutions to the Witsenhausen problem, we will return to the main topic of this subsection: to find simple functions with good performances (say similar to that of f_{LLH}) using method introduced in Chapter VI. For this sake, we add a constraint $\hat{C}(f) \le 10000$ where $\hat{C}(f)$ represents the estimate of the complexity of *f* as explained above. This constraint can be interpreted as the given memory space of 10000 bits. As explained above, we need 24192 bits to store f_{LLH} using PROBDD, and 57500 bits using lookup table, so f_{LLH} cannot be stored. Following the method introduced in Chapter VI, by randomly generating 1000 control functions satisfying that $\hat{C}(f) \le 10000$, we randomly sample solutions to the Witsenhausen problem that can be stored within the

10000-bit memory space. Using numerical integration with a large step-size to estimate the cost J of each such function as in (Lee et al. 2001), we obtain an observed Ordered Performance Curve as shown in Fig. 8.29.

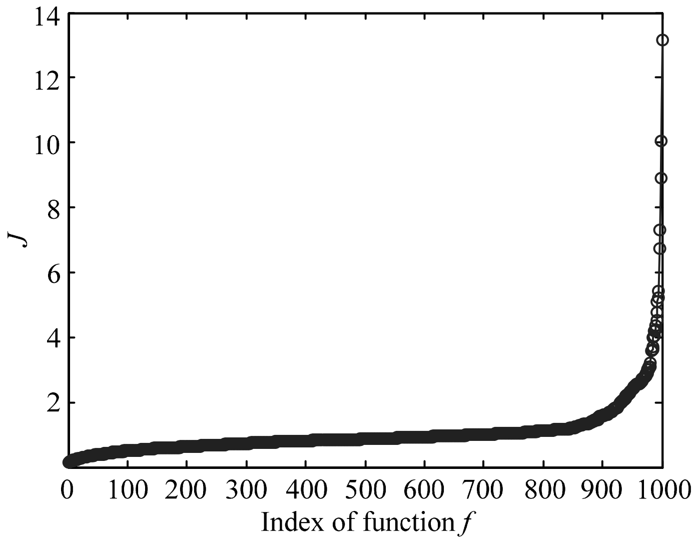

Fig. 8.29. The observed OPC of the functions that can be stored within 10000 bits in the Witsenhausen Problem (Jia et al. 2006b) © 2006 IEEE

This indicates that our problem belongs to the flat type of OPC. Also the noise level is estimated as 0.0056 after normalization. Suppose we want to find at least one of the top-5% designs that can be stored in 10000 bits. Using the UAP table in Section II.5, we calculate that we should select the observed top-37 designs. The OO theory ensures that there is at least one of the top-5% simple functions contained in the observed top-37 simple functions with a high probability. To test this, we run the above procedure by 100 replications. Each time, we randomly sample 1000 simple functions, and use long-time numerical integration to calculate the accurate performances. We find that there is always at least one truly top-5% simple function found each time, i.e., the observed alignment probability is 1 in the experiments.[11] There are actually 24.5 truly top-5% simple functions

[11] The top-5% here is w.r.t. all the simple functions. Since we do not sample complex functions (i.e., functions that need more than 10000 bits to store) in the experiments, we may not find a truly top-5% function (w.r.t. both simple and complex functions) in each replication.

found on average, with an average cost of 0.2096. The best function found in the 100 replications is

$$f_{\mathrm{sg}}(x)=\begin{cases}3.125, & 0\le x<6.25;\\ 9.375, & 6.25\le x<12.5;\\ 15.625, & 12.5\le x<18.75;\\ 21.875, & 18.75\le x,\end{cases}$$

where the subscript "sg" represents "simple and good enough". The cost of $f_{\mathrm{sg}}(x)$ is 0.1746 (estimated by the numerical integration with step size 0.001). $\hat{C}\left(f_{\mathrm{sg}}\right)=4704$. The complexity and performance of f_{sg} is also shown in Table 8.13 and Fig. 8.28. Although function $f_{\mathrm{sg}}(x)$ is not as good as the best-so-far function $f_{\mathrm{LLH}}(x)$, it is already better than $f_{\mathrm{DH}}(x)$ with cost 0.1901, $f_{\mathrm{BB}}(x)$ with cost 0.3634, $f_{\mathrm{W}}(x)$ with cost 0.4042, and of course the best linear function with cost 0.96. Considering the fact that in the procedure of finding $f_{\mathrm{sg}}(x)$, we did not utilize much problem information as in (Lee et al. 2001), for example, the placement of the jump points and the slight slope in each step, we are quite satisfied.

We have mentioned in Chapter VI that the KC depends on the computer that executes the program. Some computer has a longer list of commands. This allows us to use a shorter program to implement the same function. For example, in some computers the numbers 0.05, 0.10, 0.15, and 0.20 are not stored in the normal form, i.e., the binary numbers converted directly from the decimals. Instead, these computers record a base 0.05, and record the numbers as 1, 2, 3, and 4. In this way, we only need to save one decimal fraction accurately, thus save the memory space. This technique is commonly adopted in our laptops and desktops. By using this technique, we can further reduce the complexity of function f's from Table 8.13 to Table 8.14. In such a computer, f_{sg} only requires 600 bits. With minor performance degradation (within 5%), we save the memory space by over 30 folds (from 22176 bits to 600 bits). We also show the new complexity and performance in Fig. 8.30. In Fig. 8.30, f_{BB} dominates f_{W} in the sense that they have the same complexity but f_{BB} has better performance. f_{sg} dominates f_{DH} in both the complexity and the performance. The f_{LLH}, $f_{3.5}$, f_{sg}, and f_{BB} are the Pareto frontier.

Table 8.14. The complexity and performance of the milestone f's when allowing to change the base in input and output

f	Base (input)	# of input bits	Base (output)	# of output bits	# of nodes	# of boxes	# of rules (r)	d	$4rd$ (bits)	J
f_{W}	–	0	5	1	0	1	2	3	24	0.4042
f_{BB}	–	0	3.9894	1	0	1	2	3	24	0.3634
f_{DH}	6.41	1	0.0001	17	1	2	70	6	1680	0.1901
f_{sg}	6.25	2	3.125	3	3	4	30	5	600	0.1746
$f_{3.5}$	0.05	9	0.1	8	48	4	160	7	4480	0.1714
f_{LLH}	0.01	12	0.05	9	182	14	616	9	22176	0.1673

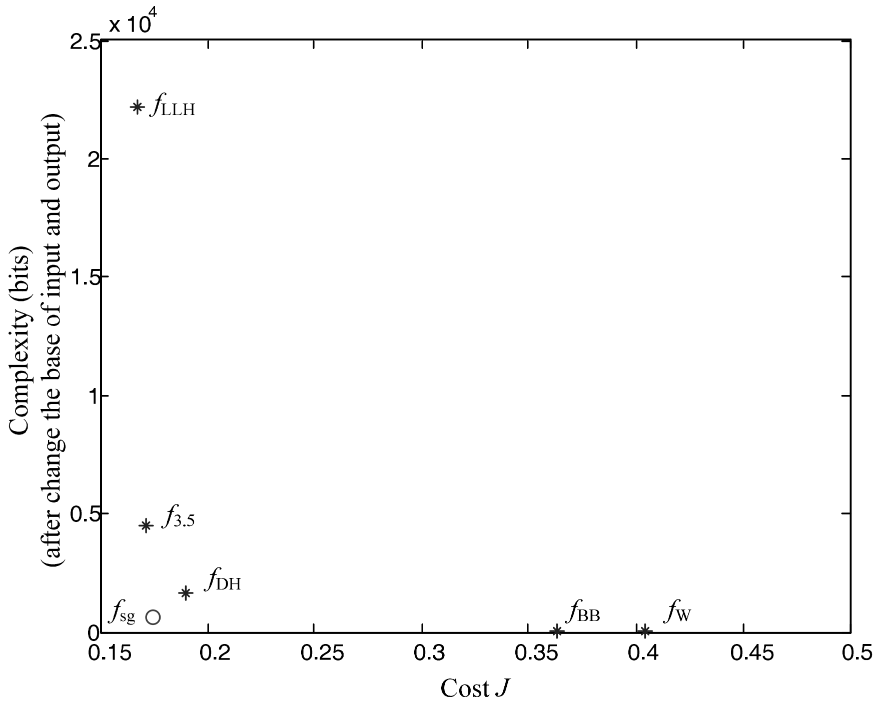

Fig. 8.30. The complexity and performance of the milestone f's after changing the base of input and output

4.3 Conclusion

In this section we have considered the famous Witsenhausen problem in team decision theory. By comparing the observed PDF of each subset, we find an easy way to discover several properties of good designs, using numerical experiments, thus can find the promising subsets efficiently. When applied in year 1999, this method finds a function for the Witsenhausen problem, which is better than the best function known by that time. Following the step-function formulation, a best-so-far function was found in 2001 by Lee et al. However, the solution seems involving a high degree of descriptive complexity. By combing OO and OBDD as introduced in Chapter VI, we are able to find simple and good enough functions with high probability. Applying this idea to the Witsenhausen problem, with minor performance degradation, we save the memory space by over 30 folds.