INFORMATICA, 1995, Vol. 6, No. 2, 193–224

AVERAGE COMPLEXITY AND THE BAYESIAN HEURISTIC APPROACH TO DISCRETE OPTIMIZATION

Jonas MOCKUS

Institute of Mathematics and Informatics
2600 Vilnius, Akademijos St. 4, Lithuania

Abstract. We apply some concepts of Information-Based Complexity (IBC) to global and discrete optimization. We assume that only partial information on the objective is available. We gather this partial information by observations. We use the traditional IBC definitions and notions while defining formal aspects of the problem. We use the Bayesian framework to consider less formal aspects, such as expert knowledge and heuristics,

We extend the traditional Bayesian Approach (BA) including heuristics. We call that a Bayesian Heuristic Approach (BHA).

We discuss how to overcome the computational difficulties using parallel computing. We illustrate the theoretical concepts by three examples: by discrete problems of flow-shop sheduling and parameter grouping, and by a continuous problem of batch operations scheduling.

Key words: complexity, average, Bayesian, optimization, heuristic, global, discrete.

1. Outline of IBC

1.1. General objective. We introduce some concepts of Information Based-Complexity (IBC) (Packel and Wozniakowski, 1987). We follow Traub and Wozniakowski (1992) while using the IBC notation and definitions. Let

$$S\colon\ F \to G, \tag{1}$$

where F is a subset of a linear space and G is a normed linear space. We wish to compute an approximation to $S(f)$ for all f from F.

1.2. Information operations. Typically, f is an element from an infinite-dimensional space and it cannot be represented on a digital computer. We

therefore assume that only partial information[1] about f is available. We gather this partial information about f by computing information operations $L(f)$, where $L \in \Lambda$ [2]. The class Λ denotes a collection of information operations that may be computed.

For each $f \in F$, we compute a number n of information operations from the class Λ. Let

$$N(f) = \big(L_1(f), L_2(f), \ldots, L_n(f)\big), \quad L_i \in \Lambda \tag{2}$$

be the computed information about f. We stress that L_i can be chosen adaptively. That is, the choice of L_i may depend on the already computed $L_1(f)$, $L_2(f), \ldots, L_{i-1}(f)$. We do not consider termination criteria, thus we fix the number n of information operations.

$N(f)$ is called the information about f, and N is the information operator. In general, N is many-to-one, and that is why it is impossible to recover the element f, knowing $y = N(f)$ for $f \in F$. For this reason, the information N is called partial.

1.3. Algorithm of approximation. Having computed $N(f)$, we approximate $S(f)$ by an element $U(f) = \phi(N(f))$, where ϕ: $N(F) \to G$. A mapping ϕ is called an algorithm.

The definition of error of the approximation U depends on the setting. In the worst case setting

$$e(U) = \sup_{f \in F} \big\|S(f) - U(f)\big\|, \tag{3}$$

and in the average case setting, given a probability measure μ on F,

$$e(U) = \int_{f \in F} \big\|S(f) - U(f)\big\| \mu(df). \tag{4}$$

1.4. Cost of computing. Suppose that for each $L \in \Lambda$ and for each $f \in F$, the computation of $L(f)$ costs a unit. We call that an observation cost. Let the cost (N, f) denote the cost of computing the information $N(f)$. Note that the

[1] For simplicity, we do not consider contaminated information.

[2] Later we call those operations as observations.

$\text{cost}(N, f) \geqslant n$, and the inequality may occur since adaptive selection of L_i may require some operations in addition to the cost of n observations.

Knowing $y = N(f)$, we compute $U(f) = \phi(y)$ by combining the information $L_i(f)$. Let the $\text{cost}(\phi, y)$ denote the cost of operations from those to be computed $\phi(y)$.

The cost of computing $U(f)$, the $\text{cost}(U, f)$ are given by

$$\text{cost}(U, f) = \text{cost}(N, f) + \text{cost}(\phi, N(f)). \tag{5}$$

In the average case setting

$$\text{cost}(U) = \int_F \text{cost}(U, f)\mu(df). \tag{6}$$

Later on we shall divide the $\text{cost}(U, f)$ into two parts: the cost of observations n and the cost of auxiliary computations a necessary for adaptive selection of L_i and for computation of the algorithm $\phi(y)$. Usually we are certain how to define the observation cost but are not certain about the cost of auxiliary computations. Therefore later on we shall consider the cost of auxiliary computations just in general terms, such as "low", "high" etc.

In the IBC framework the ε-complexity is defined as a minimal cost among all U with error at most ε

$$\text{comp}(\varepsilon) = \inf\{\text{cost}(U)\colon\ U \ \text{ such that } \ e(U) \leqslant \varepsilon\}. \tag{7}$$

1.5. Radius of information. The traditional IBC theory makes a distinction between information and algorithm. As we have already explained, the approximation $U(f)$ is computed by combining information operators from the class Λ. Let $y = N(f)$ denote the computed information. The set $S(N^{-1}(y))$ consists of all elements from G which are indistinguishable from $S(f)$. Since $U(f)$ is the same for any f from the set $N^{-1}(y)$, the element $U(f)$ must serve as an approximation to any element g from the set $S(N^{-1}(y)$. The quality of approximation $U(f)$ depends on the "size" of the set $S(N^{-1}(y)$. Therefore IBC defines the radius of information $\text{rad}_{\text{inf}}(N)$ considering the set $S(N^{-1}(y)$ for $y \in N(F)$. In the worst case setting that means the maximal radius

$$\text{rad}_{\text{inf}}(N) = \sup_{y \in N(F)} \text{rad}\left(S(N^{-1}(y))\right). \tag{8}$$

Here $\operatorname{rad}(A) = \inf_{x \in G} \sup_{a \in A} \|x - a\|$.

In the average case setting $\operatorname{rad}_{\inf}(N)$ means the average radius. A formal definition of the average radius is more complicated.

1.6. Bayesian risk. We shall not use the information radius while considering the optimization problems. We mentioned the information radius just as an elegant concept of IBC. We shall minimize the Bayesian risk $R(\mu)$ defined by the condition

$$R(\mu) = \inf_{U} e(U). \tag{9}$$

We stress that we take the infimum over all possible U. We see that U can be identified with a pair (N, f), where N is the information and ϕ is the algorithm that uses this information. It means that we take the infimum over all the information N consisting of information operators from the class Λ, and over all the algorithms ϕ that use N. Now we apply those concepts to a global optimization example.

2. IBC and global optimization

2.1. Observations. Let F be a family of continuous real functions f defined on an m- dimensional cube $A = [-1, 1]^m \subset R^m$, thus $G = R^m$. Let

$$S(f) = x^*(f) = x^* = \arg \min_{x \in A} f(x). \tag{10}$$

The information is obtained by the information operator

$$N(f) = \big(f(x_1), f(x_2), \ldots, f(x_n)\big) \tag{11}$$

with the points x_i adaptively chosen and the number n given.

We call a pair $z_i = (y_i, x_i)$, where $y_i = f(x_i)$, an observation. We call the vector $z(n) = (z_1, z_2, \ldots, z_n)$ an observation vector.

Denote the final decision by x_{n+1} and the result of the final decision by $y_{n+1} = f(x_{n+1})$.

We consider adaptive information, thus the point of the next observation x_{i+1} may depend on the already observed results $z(i) = (z_1, z_2, \ldots, z_{i-1})$. The observation vector $z(n) = l(N, f)$ is the information about f obtained after n observations. In general, $z(n)$ is many-to-one, and that is why it is impossible to recover the element f, knowing $z(n) = N(f)$ for $f \in F$.

2.2. Decision algorithms. Having computed $z(n) = N(f)$, we approximate $S(f)$ by an element $x_{n+1} = U(f) = \phi(N(f))$, where ϕ: $z(n) \to A$. We call x_{n+1} the final decision.

If we wish to make each observation in an optimal way we have to define the algorithms of sequential decisions, too. We call a mapping ϕ_i^n: $z(i) \to A$ an algorithm of sequential decision if $i < n$. Thus we define each observation $x_{i+1} = \phi_i^n(z(i))$, $i = 1, \ldots, n$ [3].

2.3. Errors. The error of the approximation U in the average case setting

$$e(U) = \int_F w\big(S(f), U(f)\big)\mu(df). \tag{12}$$

We assume a linear loss function:

$$w\big(S(f), U(f)\big) = f\big(U(f)\big) - f\big(S(f)\big). \tag{13}$$

Here $f(S(f)) = f(x^*)$ is the global minimum of f, and $f(U(f)) = f(x_{n+1})$ is its approximation after n observations.

The second component $f(S(f))$ of expression (14) does not depend on the decision algorithms ϕ and ϕ_i^n. Therefore we omit this component

$$w(U(f)) = f\big(U(f)\big) = f(x_{n+1}). \tag{14}$$

In the average case setting, given a probability measure μ on F,

$$e(U) = \int_{f \in F} f\big(U(f)\big)\mu(df). \tag{15}$$

3. Bayesian approach (BA)

3.1. Sequential decisions. After the observation i we choose the next $x_{i+1} \in A$ the infimum to be attained:

$$x_{i+1} = \arg \inf_{x \in A} \int_{N^{-1}(z(i))} f\big(U(f)\big)\mu\big(df | z(i)\big). \tag{16}$$

[3] The notion of sequential decisions is indirectly included while defining the adaptive information operations L_i in the traditional IBC framework (Packel and Wozniakowski, 1987.

Condition (16) defines the sequential decision rule ϕ_i^n: $z(i) \to A$.

Using the statistical language, we may say that the measure $\mu|z(i)$ is an a posteriori measure that represents our belief in the distribution of elements $f \in N^{-1}(z(i))$ which are indistinguishable after the information $z(i) = N_i(f)$ has been obtained. Here $N_i(f) = (f(x_1, \ldots, f(x_i))$ denote information operator (11) truncated at the observation $i \leqslant n$.

We see that when defining algorithm (16) we have to solve a kind of multi-dimensional nonlinear dynamic programming problem. The reason is that making a decision at some stage $i < n$ we have to predict what information we will get later. It means that we should predict all the pairs $z_j = (x_j, y_j)$, $i < j < n+1$. We cannot expect to obtain the complete multi-stage solution. Thus, we consider various ways to simplify the problem.

3.2. Simplified decisions. Assuming that the current observation is the last one ($i = n$) we may reduce the sequential decision rule (16) to a sequence of non-sequential problems [4]. We call the non-sequential decision based on the assumption that $i = n$ as a "zero-step" approximation (Mockus J. and L., 1991)

In doing so we may oversimplify. If the minimal conditional expectation happens to be at some observation point, then the optimization stops. An example is a Wiener process.

Consider a family of continuous functions equipped with the classical Wiener measure μ (Žilinskas and Zygliavsky, 1992). Usually

$$\arg \min_{x \in A} \mu_i(x) = x_{j(i)}, \tag{17}$$

where $\mu_i(x)$ is the conditional expectation of $f(x)$ given $z(i)$, and

$$x_{j(i)} = \arg \min_{1 \leqslant j \leqslant n} f(x_j). \tag{18}$$

From (16) and (17) it follows that

$$x_{i+1} = x_{j(i)}. \tag{19}$$

It means that the optimization stops here, (see condition (17)). And we may be far away from any minimum.

[4] Corresponding to the optimal error algorithm ϕ defined by Packel and Wozniakowski (1987).

Therefore, further we consider a "one-step" approximation assuming that the next observation is the last one: $i = n - 1$. It means that we use sequential decisions, but we look only one step ahead. . Theoretically we can see two steps ahead, three steps ahead, and so on. However we do not see any specific advantages to see more than one step ahead. Besides, we do not know any practical means to do that.

4. A posteriori measure

4.1. Traditional case. If $N(f)$ is a measurable mapping and if F is a measurable subset of a separable Banach space (Packel and Wozniakowski, 1987), then we may define the a posteriori measure $\mu|z(i)$ by an a priori measure μ. A definition of the measure $\mu|z(i)$ by the measure μ is useful for theoretical investigation. This way we keep Kolmogorov's consistency conditions and so get a powerful analytical tool. For example, we may show how the sample behavior depends on the a priori measure μ, etc. The conditions on μ providing the continuity, differentiability, and the Lipschitzian properties of sample functions are well known (Cramer and Leadbetter, 1967).

Namely, the sample functions $f(x)$ of some stochastic process ξ are continuous (mod P) on $[0,1]$ if the difference

$$\rho_h(t) = \rho(t+h, t+h) - \rho(t+h, t) - \rho(t, t+h) + \rho(t,t) \tag{20}$$

satisfies the condition

$$\rho_h(t) < \frac{K|h|}{||\log\ |h||^q}, \tag{21}$$

where $q > 3$, $K > 0$, and P is a probability measure defining the stochastic process ξ.

Those results are important while using the proper sequential decisions as defined by recurrent equations (16).

4.2. Replacing consistency conditions. It is less clear why we need to keep the consistency conditions when we apply the one-step approximation. Using the one-step approximation means that we change the statistical model after each observation. The consistency conditions mean that we update the information while keeping the same statistical model (considering the same stochastic function). Consequently, the traditional consistency conditions seem almost irrelevant using the one-step approximation. Therefore we replace them

by some weaker conditions. Namely, the continuity of Bayesian risk and simplicity of expressions (Mockus, 1989).

5. Bayesian Heuristics Approach (BHA)

5.1. Why heuristics? An important advantage of the Bayesian approach is that we may include some expert knowledge. That is the main reason why we use the Bayesian approach. Applying the traditional Bayesian setup we may include the expert knowledge while defining an a priori measure μ. Looking at many real life decision techniques we may notice that the expert knowledge is usually exploited via some expert decision rules, so called heuristics.

There are hundreds of well known heuristic optimization techniques in engineering design and many other fields (Pardalos *et al.*, 1993; Helman *et al.*, 1993). Therefore we may extend the application of the Bayesian approach if we find the ways how to include heuristic optimization into the Bayesian framework. Later on we will consider one of such ways .

5.2. Defining heuristics. Using the BHA we denote the original objective not by f but by v:

$$v = v(d),\ d \in A \subset R^m. \tag{22}$$

Denote the original optimization problem by

$$d^* = \min_{d \in D}\ v(d). \tag{23}$$

A different notation of the objective helps to recognize what approach we use: the objective v means the BHA, and the objective f means the BA, or some other traditional approach not involving any heuristics.

Denote some auxiliary objective by

$$h = h(d, v) \in H,\ d \in D = D(v) \subset A. \tag{24}$$

Call function (24) a heuristics.

Call the following mapping a randomized heuristic decision:

$$\delta_n^\pi\colon\ H \to D. \tag{25}$$

Mapping (26) defines the variable $d_n \in D$ as a function of heuristics h depending on some randomization measure π and the stage n:

$$d_n = \delta_n^\pi(h). \tag{26}$$

We call some sequence of heuristic decisions d_n π-consistent if the sequence converges to the global minimum x^*:

$$\lim_{n\to\infty} d_n = x^* \pmod{\pi}. \tag{27}$$

We consider the randomized heuristic decisions because the randomization is an easy way to get the convergence. Condition (27) holds if the probability π to hit any ε-neighborhood in a continuous case (or any point in a discrete case) is positive.

Usually we cannot specify exactly what heuristics h and what randomization π we would prefer. In such cases it is natural to consider some "mixture" of heuristics h_s and/or some mixture of randomizations π_l. However, we are not certain about the best "weights" of the mixture. The "weight" of a mixture component means the probability to use this component. It is a sort of lottery where we may "win" some particular heuristics or/and some specific randomization procedure. In this lottery the weights define the probabilities to "win".

We represent heuristics as a vector $h = (h_s,\ s = 1,\dots,S)$ and randomization measures as a vector $\pi = (\pi_l,\ l = 1,\dots,L)$. Denote the weights of the components h_s and π_l as a vector $x \in X$:

$$x = (x_i, i = 1,\dots,n),\ \sum_i = 1,\ x_i \geqslant 0,\ n = S + L. \tag{28}$$

A convenient way to represent the uncertainty about the mixture of weights is by some a priori measure μ_H on X.

Applying the Bayesian Heuristic Approach (BHA) we divide the solution of original optimization problem (23) into two parts:

- in the first part we get the optimal mixture of randomized heuristics

$$x^* = \arg\min_{x\in X} f(x), \tag{29}$$

  given an a priori measure μ_H of weights $x \in X$;
- in the second part we optimize original objective (22) using randomized heuristic decisions (26) defined by this mixture.

5.3. Optimizing heuristics. We may regard this two-part procedure as a reduction of the original problem of objective $v(d)$ optimization to an auxiliary

problem of mixture of weights x optimization. By mixture of weights we define the probabilities of using different heuristics or/and randomizations. We define by $f(x)$ the results of the repeated Monte Carlo simulation applying some consistent heuristics h to the function f using weights x. We call those techniques as Bayesian Heuristics Approach (BAH) (Mockus J. and L., 1994). We may see that the number of weight vector components is defined just by the number of different heuristics and different randomization procedures.

The dimensionality m of the original problem $\min_d v(d),\ d \in R^m$ is not directly related to the dimensionality n of the corresponding BHA problem. Usually $m > n$.

5.4. Discrete optimization. There are some additional difficulties defining an a priori measure on a set of the original discrete optimization problem. The reason is that the neighborhood relations are not uniquely defined on discrete sets. Those relations are important defining the a priori measure. It is natural to assume the statistical relation to be weak between distant points and strong between the close ones. We may apply this assumption only if we know the neighbors. The problem is that the a priori measure in discrete cases depends on the neighborhood definition which is not always clear.

We avoid this problem using the Bayesian Heuristics Approach. Here we define the a priori measure on a continuous set of weights of randomized heuristic decisions. This is the an additional advantage in applying the BHA to the discrete optimization problems.

5.5. "Teaching" heuristics. We may enhance the efficiency of the Bayesian Heuristics approach by "teaching" techniques, common in pattern recognition, neural networks, etc. By "teaching" we mean the optimization of x by some selected members of the objective function v family V. We apply those parameters to the other members of the family V later on Kuryla and Mockus (1995). The teaching is important in designing a "on-line" techniques. We investigate the effect of teaching later on, while considering the flow-shop problem, see Fig. 1.

5.6. Comparison with traditional approaches. In traditional approaches (both the IBC and the BA) we define the a priori measure μ on a set of original objective functions. The optimal decisions are determined by expressions (16).

Using the BHA we consider a different framework: we define the a pri-

ori measure μ_H on a set of randomized heuristic decisions directly, skipping condition (16). It means that we skip the most difficult part of the problem.

When applying the BA to optimize the original problem directly we:

- consider equations (16) of the same high dimensionality as that of the original problem;
- include the expert knowledge while defining an a priori measure on a set of functions to be optimized.

When applying the BHA to solve the original problem indirectly by optimizing the "mixture" of heuristics we:

- consider equations (16) of lower dimensionality that is equal to the number of different heuristics and randomization techniques which is usually not too large.
- include the expert opinion in a natural and efficient way while defining heuristics. [5]

Now we discuss in short how to "restore" the traditional BA problem, given the BHA problem. Theoretically we may describe a BHA problem by the triplet (μ_H, π, h), where μ_H is an a priori measure of mixture (28), π is a randomization measure, and h is a heuristic. We may describe the BA problem just by an a priori measure μ.

We may "restore" the BA problem in some simple cases. By restoring we understand a definition of the a priori measure μ on a set of objective functions such that the optimal solution of two parts of BHA problem (26) and (29) satisfy optimality conditions (16) of the BA problem.

In other words, restoring the BA problem we are looking for a measure μ such that gives the BA decisions the same as the BHA triplet (μ_H, π, h).

Consider a trivial example of the "irrelevant" heuristic h. We call a heuristic irrelevant if there exists no relation between h and the original objective v. In this case, under some obvious conditions, the randomized heuristic decision (26) of the BHA corresponds to a simple the Monte Carlo search. It is easy to see that the Monte Carlo search satisfies optimality condition (16) of the BA, too, if the a priori measure μ is of "white noise" type. It means that the "white

[5] In addition to that we also include the expert knowledge while defining an a priori measure on the "mixture" of heuristics. It means that the BHA provides two ways of including the expert knowledge. The traditional BA provides only one way.

noise" decisions in the traditional BA framework correspond to the decisions of irrelevant heuristics using the BHA.

We doubt the possibility of restoring BA problems from BHA problems in nontrivial cases. However we see no practical reason for such a restoration.

6. Parallel Bayesian algorithms

6.1. Outline. Bayesian algorithms need a lot of computing that we may parallelize in a natural way.

Bayesian algorithms perform three types of computations:

- the observations: here we define the objective $f(x)$ given $x = x_n \in A$;.
- the forecasting: here we optimize the Bayesian risk $R(x)$, thus defining the next observation point x_{n+1};.
- the heuristics estimation: here we define the best results of a randomized heuristics by Monte Carlo simulation.

We may apply parallel algorithms in all the three cases. Suppose that we may use K parallel processors.

6.2. Parallel risk optimization. Usually we optimize the risk $R(x)$ defined by condition (16) generating M points x by some simple covering techniques (such as Monte Carlo or LP sequences). If we got K processors, then we may calculate $R(x)$ at K different points x at the same time. Obviously $M \geqslant K$ and M should be divisible by K.

The result of risk optimization is a set $I_R(K)$ of K best points:

$$i \in I_R(K) \quad \text{if} \quad R(x_i) \leqslant R(x_j),\ j \notin I_R(K). \tag{30}$$

That is one iteration. We repeat the same procedure at each iteration.

6.3. Parallel observations. At each iteration we observe K points defined by condition (30). We perform each observation in parallel by different processors. We use all available observations defining the risk $R(x)$ in the next iteration. We continue the optimization until we reach the limit observation number N which has to be divisible by K.

We assume that the processor number K is much less as compared with the total observation number N. Otherwise we should consider different techniques.

6.4. Parallel heuristics. Denote by $f_K(x)$ the best results we have got repeating K times the "mixture" of randomized heuristics defined by the parameter vector x (Mockus A. *et al.*, 1994). For example, x_0 may denote the

"weight" of a uniform component of randomization, x_1 may denote the weight of a linear component, and x_2 may denote the weight of a quadratic component. If there are K processors, then we may perform all the K "r repetitions" at the same time. If we need more repetitions, then we may choose any repetition number divisible by K.

7. Randomizing heuristics. Let us consider discrete optimization problems. In those problems we denote decisions by m instead of the general notation d which we used in the previous sections. We keep the same notation for the original objective, namely $v(m)$.

Suppose there exists a simple function $h_i(m)$ that roughly predicts the consequences of decisions m. If we prefer a symmetric minus-one-plus-one case, then

$$\min_m h_i(m) = -1, \quad \max_m h_i(m) = +1, \quad i = 1, \ldots, I. \tag{31}$$

If a non-symmetric zero-one case seems more convenient, then

$$\min_m h_i(m) = 0, \quad \max_m h_i(m) = 1, \quad i = 1, \ldots, I. \tag{32}$$

We call such a function a "heuristics". Usually the heuristics h_i are defined as priority rules. We prefer the decision m which appears to be best now, at the stage i. We disregard future consequences of the decision m.

The so called "greedy" heuristics are an important family of heuristics. For example, (Helman *et al.*, 1993) says that "a greedy algorithm, when run on a set of systems, builds a solution by beginning with the empty set and successively adding the best remaining element while maintaining feasibility." Here the number of decision stages I is equal to the number of the system elements.

We may call another family of heuristics as the "permutation" heuristics. We can start from some initial state of the system and then perform a number of permutations. We stop, if using the given set of permutations, we can't improve the results. In such a case the number of stages may be much larger than the number of system elements.

Using the permutation heuristics, the set of decisions D_i at each stage i sometimes consists of only two decisions: after and before the permutation. The simplest permutation heuristics would be

$$h_i(m) = -v(m). \tag{33}$$

Here the conditions of heuristics normalization (31) and (32) do not hold. The objective $v(m)$ is with the sign minus, because it is convenient to regard the greater heuristics as a better one [6].

Following a similar reasoning we can design many different heuristics which improve the given feasible solution. This can be regarded as an advantage in comparison with the greedy case (when we start from the empty set). Applying the permutation heuristics we usually define the objective function for each feasible permutation.

An advantage of permutation heuristics is that we can use the expert knowledge by selecting the initial solution. Another advantage is a wide choice of different feasible permutations.

We may optimize twice. The first time we use greedy algorithms which start from the empty set. The second time we improve the solution by feasible permutation. We can eliminate the first part of optimization, if we have a good expert solution.

8. Algorithm of randomized heuristics. We may take advantage of the expert knowledge by relating the probabilities $r_i(m)$ to the heuristics $h_i(m)$. The usual assumption is that the probabilities $r_i(m)$ are proportional to heuristics $h_i(m)$. This assumption means linearity of r_i as a function of h_i. If we wish to make the relation more "adaptive", then we have to consider nonlinear functions $r_i = r(h_i)$, too. We shall define a family of functions $r_i = r(h_i)$ by a fixed number of parameters $x = (x_n, n = 1, \ldots, N)$. Then we can write $r_i(m) = r(m, h_i, x)$. Here

$$\sum_{m \in D_i} r_i(m) = 1. \tag{34}$$

For example, we can implement the chosen algorithm including condition (34) in the following three steps:

- divide the unit interval into M_i parts $U(m) = r_i(m), m \in D_i$;
- generate a random number $\xi \in [0, 1]$ uniformly;
- choose the decision m, if $\xi \in U(m)$.

In this case the problem is to get the best expression of probabilities r_i as a function of heuristics h_i.

[6] We minimize the objective v.

8.1. Polynomial randomization. An ordinary polynomial may be good if we prefer the probabilities r_i to be expressed as some monotonous functions of the heuristics h_i. Here we use zero-one condition (32),

$$r_i^0 = r^0(h_i, x) = \sum_{n=0}^{N-1} a_{in} x_n h_i^n(m), \tag{35}$$

where

$$\sum_{n=0}^{N-1} x_n = 1, \; x_n \geqslant 0, n = 0, \ldots, N-1 \tag{36}$$

and from condition (34)

$$a_{in} = \frac{1}{\sum_{m=1}^{M_i} h_i^n(m)}.$$

If $r_i^0(m) < \varepsilon$ for some m, then we correct the probabilities (Mockus A. *et al.*, 1994).

The number n in expression (35) may be regarded as the "degree of greed". The number $n = 0$ means no greed, because all feasible decisions are equally desirable. If the number of greed n is large, then we prefer the decisions with the best heuristics. Optimizing x we define degrees of greed such that provide the most efficient randomized decision procedure.

8.2. Non-smooth randomization

8.2.1. Delta randomization. If we wish to consider a "mixture" of pure and randomized heuristics, then we may apply the "delta" randomization. Here the randomized heuristics:

$$r(m) = a_0 x_0 + a_1 x_1 h(m) + a_2 x_2 \Delta_i(h(m)), \tag{37}$$

where

$$\Delta(h(m)) = \begin{cases} 1, & \text{if } h(m) = \max_{k \in \{1,\ldots,M\}} h(k), \\ 0, & \text{otherwise,} \end{cases} \tag{38}$$

and

$$a_0 = \frac{1}{M}, \quad a_i = \frac{1}{\sum_{m=1}^{M} h(m)}, \quad a_2 = 1. \tag{39}$$

8.2.2. Simulated annealing. The most popular method of global optimization, namely, the simulated annealing, may be considered in the framework of the BAH, too. We use the following heuristics, specific for simulated annealing,

$$h_i(m) = -\big(v(m) - v(\bar{m})\big). \tag{40}$$

Here

$v(m)$ is the objective after the permutation i,

$v(\bar{m})$ is the objective before the permutation i,

$D_i = \{m, \bar{m}\}$, what means that $|M_i| = 2$.

The randomized permutation heuristics:

$$r_i(m) = \begin{cases} e^{\frac{h(m)}{x/\ln(1+i)}}, & \text{if } h(m) < 0, \\ 1, & \text{otherwise,} \end{cases} \tag{41}$$

i is the iteration number;

x is the "initial temperature".

The main difference from the usual simulated annealing is that we optimize the parameter x for some fixed number of iterations. We disregard the asymptotics because the asymptotic properties are beyond the framework of the Bayesian heuristics.

9. Flow-shop problem

9.1. Definition. The flow-shop problem is important in various applications and convenient for investigation of the BHA (Mockus A. *et al.*, 1994; Kenneth and Baker, 1974).

Denote by $\tau_{j,s}$ the duration of operation (j, s), where $j \in J$ denotes a job and $s \in S$ denotes a machine. We denote by J and S the set of jobs and machines, respectively. The assumption $\tau_{j,s} = 0$ means that the operation (j, s) is irrelevant.

Suppose the sequence of machines s to be fixed for each job j. One machine can do only one job at a time. Several machines cannot do the same job at the same moment.

The decision $d_i(j) \in D_i$ means to start a job $j \in J_i$ at the stage i. We define a set of feasible decisions D_i as the set J_i of jobs available at the stage i conforming to the flow-shop rules.

The objective function is the make-span v. Denote by $T_j(d)$ the time when we complete the job j (including the gaps between operations) using the decision sequence d. Then the make-span for d is $v(d) = \max_{j \in J} T_j(d)$.

9.2. Heuristics. We can see that the number of feasible decisions is very large. We can reduce this number considering only a small subset of schedules, the so-called permutation schedules.

The permutation schedule is a schedule with the same job order on all machines, – a schedule that is completely characterized by a single permutation of job indices $1, 2, \ldots, n$. We assume the first operation to be on the first machine, the second-one on the second, and so on. The expert opinion is that using permutation schedules we can approach the optimal decision well enough (Kenneth and Baker, 1974).

Denote

$$\tau_j = \sum_{s=1}^{|S|} \tau_{j,s},$$

where $|S|$ stands for the number of machines.

We call τ_j the length of the job j.

Define the greedy heuristics by expression (32)

$$h_i(j) = \frac{\tau_j - A_i}{A^i - A_i}. \tag{42}$$

Here

$$A_i = \min_{j \in J_i} \tau_j, \qquad A^i = \max_{j \in J_i} \tau_j. \tag{43}$$

Thus, the greedy heuristics prefer longer jobs at each stage i.

$$t_j = \frac{e_j}{\min_{1 \leqslant s \leqslant |S|-1} (\tau_{j,s} + \tau_{j,s+1})},$$

and

$$e_j = \begin{cases} +1, & \text{if } \tau_{j,1} < \tau_{j,|S|}, \\ -1, & \text{otherwise.} \end{cases}$$

In both cases we define the randomized decision function r_i by expression (35). We optimize it by solving the stochastic optimization problem (Mockus A. *et al.*, 1994).

9.3. Results.

Table 1 shows the results of the Bayesian method after 100 iterations using heuristic (42) and different randomization procedures. The number of jobs J, of machines S, and of operations O each of them being equal to 10. We define the

Table 1. The results of Bayesian methods using heuristics (42)

$R = 100$, $K = 1$, $J = 10$, $S = 10$, and $O = 10$					
Randomization	f_B	d_B	x_0	x_1	x_2
Delta	6.183	0.133	0.283	0.451	0.266
Taylor 3	6.173	0.083	0.304	0.276	0.420
CPLEX	12.234	0.00	–	–	–

lengths and the sequences of operations generating random numbers uniformly from 0 to 99. We estimate the expectations and the standard deviations for each type of randomization using the results of 40 optimizations of the same randomly generated problem.

In Table 1 the symbol f_B denotes the mean, and d_B denotes the standard deviation of span. "Delta" denotes randomization (37), "Taylor 3" denotes randomization (35) with the number of terms $N = 3$, and "CPLEX" denotes the results of the well known general discrete optimization software after 2000 iterations (one CPLEX iteration is comparable with one Bayesian observation). The bad results of CPLEX show that the standard MILP technique is not efficient solving a specific problem of discrete optimization. It is not yet clear how much one may improve the results using specifically tailored branch-and-bounds techniques. We have some doubts about that.

We see that the Bayesian adaptation of heuristics significantly improved the results. The results of Table 1 in general correspond to the theoretical predictions.

Usually we compare the Bayesian and other methods neglecting the "learning" potential of the Bayesian methods. Fig. 1 estimates the teaching effect by showing the density of optimal values of the parameters x_1 and x_2. Those values correspond to 100 different randomly generated flow-shop problems. We see that the maximal density of optimal values of x_1 and x_2 exceeds the average density several times. It means that the optimal parameters of the Bayesian methods are rather robust and thus could be applied to other related problems.

For a different approach to the flow-shop problem, see (Biegler *et al.*, 1988).

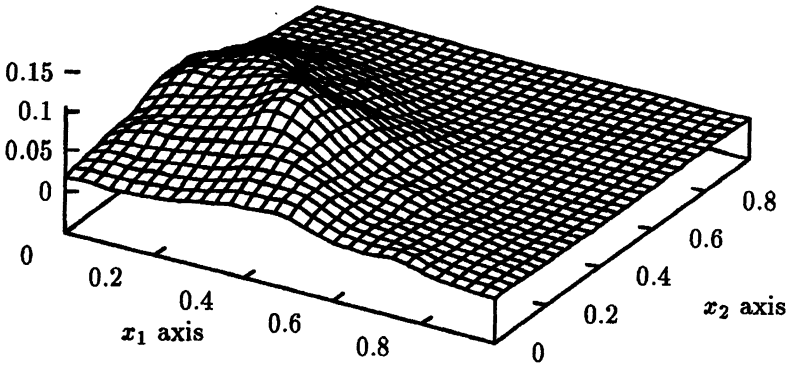

Fig. 1 The density of optimal parameters in solving different randomly generated flow-shop problems.

10. Parameter grouping

10.1. Definition. Parameter grouping is important in the cluster analysis and empirical data processing (Andenberg, 1973). Besides it is a good example to show how to apply the BHA improving the traditional techniques of simulated annealing.

A good partition means:

- strong interaction inside the groups;
- weak interaction outside the groups.

We describe the partition by a discrete vector $d = (d(1), \ldots, d(n))$. Here $d(i) = j$ means that the parameter s_i belongs to the group g_j. We maximize the sum of parameter-group interactions (Dzemyda and Senkienė, 1992) . The permutation m means adding or subtracting a unit to $d(i)$. We change only one component at a time (Mockus J., 1993). We stop the "line" search after k steps.

We optimize the initial "temperature" x (see (41)). We apply the one-dimensional Bayesian algorithm defined by the Wiener a priori measure μ.

10.2. Results. The number of randomly generated problems is 20.

The number of iterations is 20. The initial partition is:

$g_1 = \{s_1, \ldots, s_5\}$, $g_2 = \{s_6, \ldots, s_{10}\}$, $g_3 = \{s_{11}, \ldots, s_{15}\}$, $g_4 = \{s_{16}, \ldots, s_{20}\}$.

The optimal value is $x^* = 0.08$; see Fig. 2.

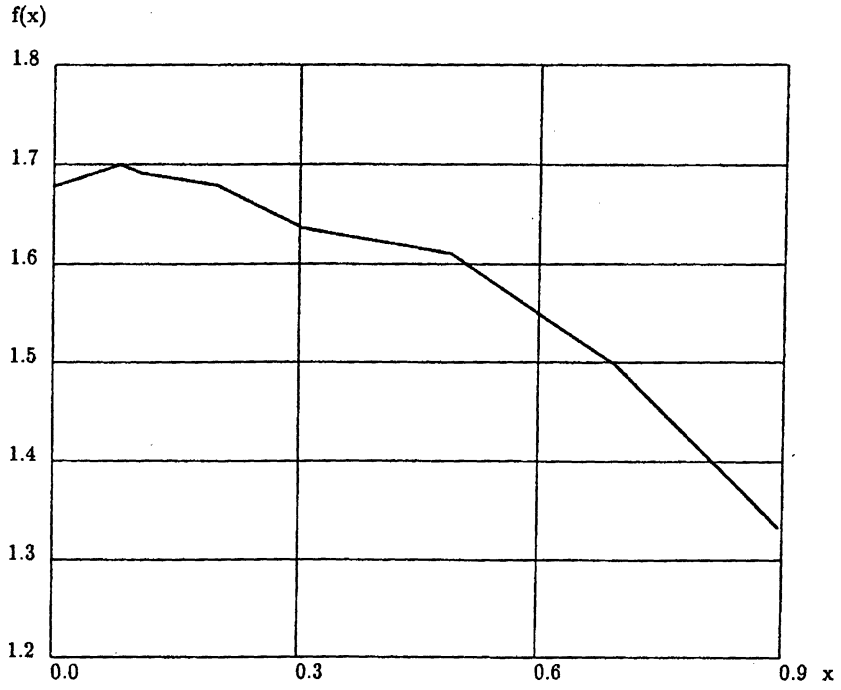

Fig. 2. The relation of simulated annealing results with the initial temperature x.

11. Scheduling of batch operations

11.1. General description. Scheduling problems occur in a wide range of industries, including the chemical processing industries. Batch production has long been the accepted procedure for the manufacture of many types of chemicals, particularly those which are produced in small quantities and for which the production processes or the demand pattern are likely to change. These products are frequently fine chemicals of high commercial value and it is important that sufficient manufacturing flexibility should be available to avoid the loss of potential sale by a failure to meet the customer's requirements in due time. As a result there has been an increasing interest in the development of procedures for scheduling batch process operations (Musier and Evans, 1989).

We consider here a simplified version of the scheduling problem (Moc-

kus L. and Reklaitis, 1994). We use the state-task-network (STN) representation of the batch process. Fig. 3 and Fig. 4 show simple examples. We consider three types of states: feed, intermediary, and product. We regard four types of events, correspondingly: start, finish, receipt, and delivery. We repeat the description of the batch scheduling problem (Mockus A. *et al.*, 1994). Then we describe the new solution algorithm and the new results (Mockus L. and Reklaitis).

11.2. Notation. Initial data are the processing time, the receipt and delivery of material, the unit price and the unit cost of material, the running cost of storing, the utility cost, the input ratio, and the output ratio.

τ_{is} is the processing time for the output of task i to the state s for each feasible pair (i, s);

x^R_{sk} is the receipt of material in the feed state s at the time t_k, usually $x^R_{sk} = a_{sk}$;

x^D_{sk} is the delivery of material in the product state s at the time t_k, usually $x^D_{sk} = d_{sk}$;

p_{sk} is the unit price of material in the state s at the time x_k;

c_{sk} is the unit cost of material in the state s at the time x_k;

c^S_{sk} is the running cost of storing a unit of material in the state s over a unit interval [7] starting at the time x_k;

c^U_{uk} is the unit cost of utility u in the state s over a unit interval [8] starting at the time t_k;

ρ^S_{is} is the input ratio of the task i from the state s;

The main variables are starting times $t = (t^S_{ij})$ and amounts of materials $x = (x_{ij})$.

t^S_{ij} is the starting time of the task i in the unit j;

x_{ij} is the amount of material involved in the task i in the unit j.

Auxiliary variables are output times, event times, event indicators, the amount of material stored, and the number of operating units.

$t^F_{is} = t^S_{ij} + \tau_{is}$ is the output time of the task i to the state s;

t_k is the time of event k [9];

$y^S_{ijk} = 1$, if the unit j starts the task i at the time t_k; $y^S_{ijt} = 0$, otherwise;

---

[7] We assume that the running cost does not change between events.

[8] We assume that the unit cost does not change between the events.

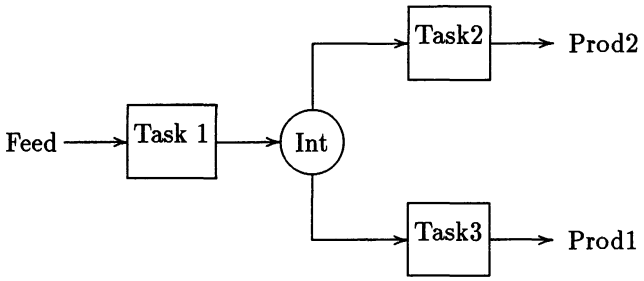

Fig. 3. State task network for example BATCH1.

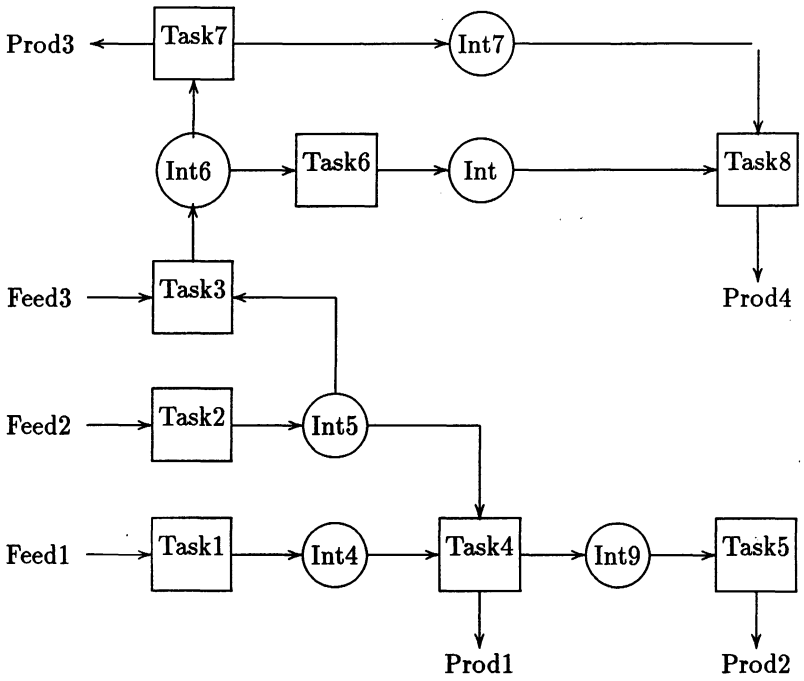

Fig. 4. State task network for example BATCH4.

$y^F_{isk} = 1$, if the task i outputs to the state s at the time t_k; $y^F_{isk} = 0$, otherwise;

B_{sk} is the amount of material stored in the state s at the time t_k (see (48));

N_{jk} is the number of units j operating at the time t_k (see (45)).

We order the events k by the inequality

$$t_{k+1} > t_k, \ k = 0, \ldots, K. \tag{44}$$

11.3. Allocation constraints. At any time a unit may perform one task at most. If the item starts performing a given task, then it cannot start any other task until the current one is finished, i.e., the operation is nonpreemptive. These requirements can be expressed in the following terms

$$N_{jk+1} = N_{jk} + \sum_i y^S_{ijk} - \sum_i y^F_{ijk}, \tag{45}$$

$$\sum_j N_{jk} \leqslant 1. \tag{46}$$

11.4. Material balance. The net increase in the amount of material stored in the state s at the time t_k is given by the difference of the amount produced in this state and that used.

$$B_{sk+1} = B_{sk} - d_{sk} + a_{sk} - \sum_i \rho^S_{is} \sum_j y^S_{ijk} x_{ij} + \sum_i \rho^F_{is} \sum_j y^F_{ijk} x_{ij}, \tag{47}$$

$$\begin{aligned} y^S_{ijk} &= 1, \quad \text{iff} \quad t_k = t^S_{ij}, \\ y^F_{ijk} &= 1, \quad \text{iff} \quad t_k = t^F_{is}. \end{aligned} \tag{48}$$

11.5. Capacity constraints. The amount of material stored in the state s must not exceed the maximum storage capacity B_s at any time

$$0 \leqslant B_{sk} \leqslant B_s. \tag{49}$$

[9] The time when we start (finish) some task, or when we receive (deliver) material. It means that $t_k = t^S_{ij}$, *or* $t_k = t^F_{is}$, *or* $t_k = t^R_s$, *or* $t_k = t^D_s$ correspondingly.

The amount of material when starting the task i in the unit j at the time t_k is limited by the minimum and the maximum capacity of that unit:

$$y^S_{ijk}B^{min}_{ij} \leqslant y^S_{ijk}x_{ij} \leqslant y^S_{ijk}B^{max}_{ij}. \tag{50}$$

11.6. Objective function. The objective $P(x,t)$ is the profit, given both delivery and receipt:

$$P(x,t) = \Big(\sum_s \sum_{k\in D} p_{sk}d_{sk} - \sum_{k\in R} c^R_{sk}a_{sk} - \sum_k \sum_s c^S_{sk}B_{sk}(t_{k+1}-t_k) - \sum_i \sum_k \sum_u U_{uik}c^U_{uk} \sum_j (t_{k+1}-t_k)\Big) \tag{51}$$

We denote by U_{uik} the amount of utility u needed for the task i in the interval starting at time t_k. We express it as follows

$$U_{uik} = \alpha_{ui} + \beta_{ui} \sum_j y^S_{ijk}x_{ij}.$$

We denote by D and R the set of delivery and receipt events, respectively.

We should satisfy conditions (44), (48), (49), (50), and (45).

We optimize only the last two components of expression (51) describing the storage and the utility cost. The first two components are constants, since both delivery and receipt are given. We may use the same algorithm to optimize the delivery and the receipt, too.

11.7. Penalty function. The penalty function $L_n(x,t)$ is a convenient way to reduce the feasible region to the rectangular form. We define the penalty function by three components:

$$L_(x,t) = -P(x,t) + P^b_n(x,t) + P^c_n(x,t). \tag{52}$$

Here $P^b_n(x,t)$ is a penalty for the violation of Boolean constraints in the iteration n, and $P^c_n(x,t)$ is a penalty for violation of continuous constraints in the iteration n. Further we call those functions as a Boolean and a continuous residual, respectively.

From (45) the Boolean residual:

$$P^b_n(x,t) = \begin{cases} r^b_n \sum_j N_{jk} - 1, & \text{if } \sum_j N_{jk} \geqslant 1, \\ 0, & \text{otherwise,} \end{cases} \tag{53}$$

$$r_{n+1}^b > r_n^b. \tag{54}$$

From (48), (49), and (50) the continuous residuals:

$$P_n^{c_0}(x,t) = \begin{cases} r_n^c(-B_{sk}), & \text{if } B_{sk} < 0, \\ 0, & \text{otherwise,} \end{cases} \tag{55}$$

$$P_n^{c_1}(x,t) = \begin{cases} r_n^c(B_{sk} - B_s), & \text{if } B_{sk} \geqslant B_s, \\ 0, & \text{otherwise,} \end{cases} \tag{56}$$

$$P_n^{c_2}(x,t) = \begin{cases} r_n^c(x_{ij}y_{ijk}^S - B_{ij}^{\max}y_{ijk}^S), & \text{if } x_{ij}y_{ijk}^S \geqslant B_{ij}^{\max}y_{ijk}^S, \\ 0, & \text{otherwise,} \end{cases} \tag{57}$$

$$P_n^{c_3}(x,t) = \begin{cases} r_n^c(B_{ij}^{\min}y_{ijk}^S - x_{ij}y_{ijk}^S), & \text{if } B_{ij}^{\min}y_{ijk}^S \geqslant x_{ij}y_{ijk}^S, \\ 0, & \text{otherwise,} \end{cases} \tag{58}$$

$$P_n^c(xt) = \sum_{k=0}^{k=3} P_n^{c_k}(x,t), \tag{59}$$

$$r_{n+1}^c > r_n^c. \tag{60}$$

The right choice of penalty parameters r_n^b and r_n^c is an important problem using penalty functions. If we set too large values of those parameters, then the optimization problem may degenerate into the search for the "nearest" feasible decision. If we set too small values for r_n, then we may violate the constraints. We may reach some compromise by decreasing penalty parameters after each iteration (see inequalities (54) and (60)).

The Boolean penalty parameter r_n^b must be much greater than the continuous one r_n^c. The reason is that we may allow some violation of continuous constraints, if the constrains are defined by economical considerations. If not, then we may include some "safety margin" while defining continuous constraints of strict engineering or legal nature. Unfortunately, we can't do such things in the case of Boolean constraints. We may ignore the strict nature of Boolean constraints at the initial stages of global optimization, when the optimum is still far away. We have to keep to those constraints exactly when approaching the global optimum.

11.8. Algorithms. A direct way of optimizing the penalty function $L(x,t)$ is by the usual continuous global optimization techniques. Unfortunately, those techniques are designed for the case when the penalty parameters r_n don't depend on the iteration number.

If the number of variables is not large we may apply the exact Branch-and-Bounds techniques. However, if the number of variables is great, then the Bayesian heuristics approach seems a good alternative.

A simple way of applying this approach is by fixing some initial decision (x^0, t^0) (not necessarily feasible), and then improving it by the optimized randomization techniques (Mockus A. *et al.*, 1994).

The permutation of the current decision (x^n, t^n) may be performed sequentially by adding some Gaussian $(0, \sigma)$ random variable to each component of x and t one by one. A more sophisticated permutation would be by adding a Gaussian random vector to several components at a time. In the latter case we need some expert knowledge for choosing the right combination of components.

In the simplest simulated annealing case (see Subsection 5.3) we have to optimize only one randomization parameter, namely, the initial temperature. However, we may also optimize the penalty parameters r_n^b and r_n^c. For example, the parameters $r_0^b,\ r_0^c$, where

$$r_n^b = r_0^b/n,\ r_n^c = r_0^c/n. \tag{61}$$

Here n is the iteration number.

Optimization of the permutation variance σ^2 could also be useful.

If we wish to optimize the penalty parameters we ought to finish the optimization of the penalty function by searching for the nearest feasible solution. It is a difficult additional problem, especially for the Boolean constraints. However it is a less difficult problem than the global optimization. It may be solved by various techniques. For example, by fixing a very large Boolean penalty parameter, or by repeating the random search as long as we hit the feasible Boolean values, or by changing the starting times by a specific heuristic rule.

There are several publications using simulated annealing in different shedulling problems (Tandom *et al.*, 1991; Floquet *et al.*, 1993; Das *et al.*, 1990; Patel *et al.*, 1991). No regular optimization of simulated annealing parameters was performed.

11.9. Results. We compared the BHA algorithm with branch and bound enumeration ($B\&B$), (see (Zentner *et al.*, 1994)), using examples BATCH1 and BATCH4 from (Sahinidis and Grossmann, 1991). Fig. 3 and Fig. 4 show the corresponding state-task networks. Table 2 and Table 3 show the data. We modify the examples so that the uniform time discretization interval length be much

smaller than the scheduling horizon. We may do that by some insignificant changes in the processing times of their original values.

Table 2. Data used for example BATCH1

Units, tasks			
Units	Size	Unit suitability	Processing times
Unit1	1500	Task1	0.9
Unit2	1000	Task2	1.1
Unit3	1000	Task3	0.8

States		
States	Capacity limits	Prices
Feed	Unlimited	5
Int	5000	
Prod1	Unlimited	10
Prod2	Unlimited	8

Demands						
	Time					
States	4	6	7	10	11	12
Prod1	200		300	400	100	
Prod2	50	150		200		100

Cost data		
$\alpha_{ui} = 200$	$\beta_{ui} = 0.6$	$c_s^M = 0.18$

We summarize the results (Mockus L. and Reklaitis (to appear)) in Table 4. There are three lines for each example. The lines relate to stopping $B\&B$ enumeration after 10000, 20000, and 30000 nodes, correspondingly. We did get close enough CPU times for the Bayesian algorithms, too, by modifying the stopping conditions accordingly. We were surprised to see that the "optimal" $B\&B$ profit remains the same in all the three lines. The "optimal" Bayesian profit increases with the number of observations, as expected.

We see that the proposed approach is undoubtedly more efficient as compared with the branch and bound enumeration in the cases when the uniform time discretization interval is small.

Table 3. Data used for example BATCH4

Units, tasks

Units	Size	Unit suitability	Processing times
Unit1	1000	Task1	0.8
Unit2	2500	Task3,7	1.2
Unit3	3500	Task4	0.9
Unit4	1500	Task2	1.1
Unit5	1000	Task6	0.8
Unit6	4000	Task5,8	1

States

States	Capacity limits	Prices
Feed1,2,3	Unlimited	0
Int4	1000	
Int5	1000	
Int6	1500	
Int7	2000	
Int8	0	
Int9	3000	
Prod1	Unlimited	18
Prod2	Unlimited	19
Prod3	Unlimited	20
Prod4	Unlimited	21

Demands

	Time					
States	3	4	5	6	7	8
Prod1	110	110	133.3	100	33.3	33.3
Prod2		233.1	260		360	360
Prod3			116.6	56.6		116.6
Prod4				333.3	333.3	685.8

Cost data

$\alpha_{ui} = 200$ $\beta_{ui} = 0.6$ $c_s^M = 0.18$

Table 4. Comparison of BHA and B&B

	B& B		BHAq	
	Profit	Time	Profit	Time
BatcH1	2744.5	584.9	3007	232.3
	same	1217.5	3079	925.7
	same	1901.2	3079	1396.5
BATCH4	60224	944.8	60306.2	882.1
	same	1919.7		
	same	2944.5	60318.8	20981.3

12. Conclusions. We may apply the traditional IBC definitions and notions directly while formulating the global optimization problem in general. If we wish to include less formal aspects, such as expert knowledge, then the Bayesian framework seems more and natural.

Bayesian Heuristics Approach (BHA) is convenient for parallel computing. The flow-shop and parameter grouping examples show the computational advantages of the BHA.

Applying the BHA technique to solving batch scheduling problems we used the well-known and rather common simulated annealing randomization techniques instead of some specific heuristics as a first step of investigation. And we have got encouraging results.

In theory the polynomial type randomization or a random mixture of different heuristics should be better since we get more parameters to tune up. The Bayesian randomized heuristics techniques are designed for the optimization of stochastic problems, too. Therefore those techniques may work well while solving problems with stochastic processing times, with random demands, etc. These and other related issues are the topics of our further research in this area.

REFERENCES

Anderberg, M.R. (1973). *Cluster Analysis for Applications.* Academic Press, New York.

Biegler, L.T., I.E.Grossmann and G.V.Reklaitis (1988). Applications of operations research methods in chemical engineering. In *Engineering Design: Better Results Through OR Methods,* North-Holland,, Amsterdam.

Cramer, H., and M.R.Leadbetter (1967). *Stationary and Related Stochastic Processes.* John Wiley, New Yok.

Das, M.D., H.Cummings, P.T. Le Van (1990). Scheduling of serial multiproduct batch processes via simulated annealing. *Computers and Chemical Engineering,* **14**, 1351–1362.

Dzemyda, G., and E.Senkienė (1990). Simulated annealing for parameter grouping. In *Information Theory, Statistical Decision Theory, Random Processes,* Prague. pp. 373–383.

Floquet, P., P.Pibouleau, S.Domenech (1993). Scheduling and simulated annealing application to a semiconductor circuit fabrication plant. *Computers and Chemical Engineering,* **14**, 1351–1362.

Helman, P., B.M.E. Moret and H.D.Shapiro (1993). An exact characterization of greedy structures. *SIAM Journal of Discrete Mathematics,* **6**, 274–283.

Kenneth, R.Baker (1974). *Introduction to Sequencing and Scheduling.* John Willey & Sons, New York.

Kuryla, H., and J.Mockus (1995). Solving differential equations by event driven techniques for parameter optimization. *Informatica,* **5**(3–4), 338–350.

Mockus, A., J.Mockus and L.Mockus (1994). Bayessian approach adapting stochastic and heuristic methods of global and discrete optimization. *Informatica,* **5**(1–2), 123–166.

Mockus, J. (1989). *Bayesian Approach to Global Optimization.* Kluwer Academic Publishers, Dordrecht–London–Boston.

Mockus, J., and L.Mockus (1991). Bayesian approach to global optimization and applications to multiobjective and constrained optimization. *Journal of Optimization Theory and Applications,* **70**(1), 155–171.

Mockus J. (1993). Application of global line search in optimization of networks. In Ding Zhu Du and P.M.Pardalos (Eds.), *Network Optimization Problems,* World Scientific Publishing Co.

Mockus, L., and G.V.Reklaitis (1994). Mathematical programming formulation for scheduling of batch operations using nonuniform time discretization. In *AIChE Annual Meeting,* **235d**, San-Francisco, California.

Mockus, L., and G.V.Reklaitis A new global optimization algorithm for batch process scheduling. *Journal of Global Optimization,* (to appear).

Musier, R.F.H., and L.B.Evans (1989). An approximate method for the production scheduling of industrial batch processes with parallel units. *Computers Chem Engng.,* **13**, 229–238.

Packel, E.W., and H.Wozniakowski (1987). Recent developments in information-based complexity. *Bulletin of the AMS,* **17**, 9–35.

Pardalos, P.M., K.A.Murthy and T.P.Harrison (1993). A computational comparison of local search heuristics for solving quadratic assignment problems. *Informatica,* **4**(1–2), 172–187.

Sahinidis, N.V., and I.E.Grossmann (1991). Reformulation of multiperiod milp models for planning and scheduling of chemical processes. *Computers and Chemical Engineering,* **15**, 255–272.

Patel A.N., R.S.H.Mah, I.A.Karimi (1991). Preliminary design of multiproduct noncontinuous plants using simulated annealing. *Computers and Chemical Engineering,* **15**, 451–469.

Traub, J.F., and H.Wozniakowski (1992). perspectives on information-based complexity theory. *Bulletin of the AMS,* **26**, 26–52.

Tandom, M., P.T.Cumming, M.D.La Van (1991). Flowshop sequencing with nonpermutation schedules. *Computers and Chemical Engineering,* **15**, 601–607.

Zentner, M.G., J.F.Pekny, D.L.Miller and G.V.Reklaitis (1994). Rcsp++: A scheduling system for the chemical process industry. In *Proc. PSE'94,* Kyongju, pp. 491–495.

Žilinskas A., and A.Zygliavsky (1992). *Method of Global Optimization.* Nauka, Moscow.

Received April, 1995

J. Mockus graduated from the Kaunas Technological University, Lithuania in 1952. He received a Doctor habilius degree from the Institute of Computers and Automation, Latvia, in 1967. He is head of the Optimal Decision Theory Department, Institute of Mathematics and Informatics, Vilnius, Lithuania, and a professor of Kaunas Technological University.

VIDUTINIS SUDĖTINGUMAS IR BAYESO HEURISTINIS POŽIŪRIS DISKRETINIAME OPTIMIZAVIME

J. Mockus

Taikomos informacinės sudėtingumo teorijos (IBC) koncepcijos globalinio bei diskretinio optimizavimo uždaviniams aprašyti. Tradiciniai IBC apibrėžimai naudojami aprašant gerai formalizuojamus uždavinių aspektus. Aprašant sunkiai formalizuojamus aspektus, tokius, kaip ekspertinės žinios bei heuristikos, naudojamas Bayeso požiūris (BA).

Tradicinis Bayeso požiūris pritaikomas naudojant heuristikas. Tai pavadinama heuristiniu Bayeso požiūriu (BHA). Aptariamos lygiagrečių skaičiavimų galimybės naudojant BHA.

Teorija iliustruojama trimis praktiniais uždaviniais: vienu tolydiniu ir dviems diskretiniais.