Proceedings of the World Congress on Engineering and Computer Science 2007
WCECS 2007, October 24-26, 2007, San Francisco, USA

Scenario Generation And Analysis by Heuristic Algorithms

Jan Roupec, and Pavel Popela

Abstract— **The main purpose of this paper is to present of how to use heuristic approaches, especially genetic algorithms, to a scenario manipulation in two-stage stochastic programming. Algorithms developed for two-stage stochastic linear programs usually require a solution of large-scale linear and nonlinear programs because the deterministic reformulations of original programs are based on empirical or sampling discrete probability distributions, i.e. on so-called scenario sets. As these sets are often huge, the reformulated programs must be simplified, and therefore, the scenario set reduction techniques are required. Hence, randomly selected reduced scenario sets are often employed. Related confidence intervals for the optimal objective function values have been derived and are often presented as tight enough. However, there is also demand for goal-oriented scenario manipulations. Traditional deterministic techniques are limited by the size of scenario set. Therefore, this paper introduces a possibility how to modify scenario sets by using heuristic algorithms. As an example, the search of absolute lower and upper bounds by using genetic algorithm is presented and further enhancements are discussed. The proposed technique is implemented in C++ and GAMS and then tested on real-data examples.**

Index Terms— **Stochastic programming, scenarios, worst case analysis, heuristic and genetic algorithms.**

I. Basic concepts of stochastic programming

We denote a mathematical program as $? \in \arg\min\{f(\mathbf{x}) \mid \mathbf{x} \in C\}$. Then, we naturally obtain an underlying program $? \in \arg\min\{f(\mathbf{x},\xi) \mid \mathbf{x} \in C(\xi)\}$, as we replaced several original constant parameters by random elements. There is ξ, a random vector defined on the probability space (Ξ,Σ,P), and $f : \mathrm{IR}^n \times \Xi \to \mathrm{IR}$, is a measurable function for each decision $\mathbf{x} \in \mathrm{IR}^n$ that must belong to the feasible set *C*. To be able to solve optimization problem correctly, the deterministic reformulation must be further specified. Usually, we cannot wait for observation ξ^s, and we must decide here-and-now. In this case, we have to utilize suitable HN-reformulation and we have chosen the most typical one:

$$? \in \arg\min_{\mathbf{x}}\{E_\xi f(\mathbf{x},\xi) \mid \mathbf{x} \in C(\xi)\ \text{a.s.}\}, \tag{1}$$

where *E* denotes expectation functional and abbreviation a.s. means almost surely. However, there are also different approaches to random parameter modeling, see, e.g., [6] for details.

It is difficult to solve the stochastic program (1) in the case when the random vector has the continuous probability distribution. Then, the approximation techniques based on discretization are used, see [6]. So, we focus on a finite support case. For the discrete random vector ξ, instead of solution difficulty related to multidimensional integration to compute *E*, we have to deal with the computational complexity caused by the HN-reformulation size. Particularly, *E* is computed explicitly as we may denote $p_s = P(\xi = \xi^s)$ and write the expectation as a sum. Therefore, the SB-reformulation is a large nonlinear program:

$$? \in \arg\min_{\mathbf{x}}\left\{\sum_{\xi^s\in\Xi} p_s f(\mathbf{x},\xi^s) \,\middle|\, \mathbf{x} \in \bigcap_{\xi_s\in\Xi} C(\xi^s)\right\}. \tag{2}$$

The main problem is of how to choose suitable realizations ξ^s called scenarios when only incomplete information about the probability distribution is available. The scenario-related techniques are discussed in the paper [3], an interesting approach is proposed in [5].

II. Test examples

A scenario-based (SB), two-stage stochastic linear program, modeling a principal part of a melt control process in a suitable furnace (cupola, induced, or electric-arc) is chosen as an example for further computations because it allows to use real world data and can be easily modified, see [13] for details.

$$\begin{gathered} ? \in \arg\min_{x}\left\{\mathbf{c}^{\mathrm{T}}\mathbf{x} + \sum_{s=1}^{S} p_s Q(\mathbf{x};\xi^s) \;\middle|\; \mathbf{x} \geq 0\right\} \\ Q(\mathbf{x};\xi^s) = \min_{\mathbf{y}^s}\left\{\mathbf{q}^{\mathrm{T}}\mathbf{y}^s \,\middle|\, \mathbf{T}_1^s\mathbf{A}_1\mathbf{x} + \mathbf{A}_2\mathbf{y}^s \geq \mathbf{l}_2,\right. \\ \left.\mathbf{T}_1^s\mathbf{A}_1\mathbf{x} + \mathbf{A}_2\mathbf{y}^s \geq \mathbf{u}_2,\, \mathbf{x}, \mathbf{y}^s \geq 0,\, s = 1,\ldots,S\right\} \end{gathered} \tag{3}$$

where $\mathbf{x}$ is a tonnage of n_1 charge materials and $\mathbf{y}^s$

Manuscript received July 22 2007. This work has been supported by the Czech Ministry of Education in the frame of MSM 0021630529 *Intelligent Systems in Automation* and by the GA CR project: 103/05/0292 and the CQR research center of the Czech Republic.

J. Roupec is with the Institute of Automation and Computer Science, Faculty of Mechanical Engineering, Brno University of Technology, Technicka 2, 616 69 Brno, Czech Republic (phone: +420 54114 3346; e-mail: roupec@fme.vutbr.cz).

P. Popela is with the Institute of Mathematics, Faculty of Mechanical Engineering, Brno University of Technology, Technicka 2, 616 69 Brno, Czech Republic (e-mail: popela@fme.vutbr.cz).

represents a tonnage of n_2 alloying materials. Symbols $\mathbf{l}_2$ and $\mathbf{u}_2$ describe a minimum and maximum tonnage of m considered elements after alloying. Then, $\mathbf{A}_1$ is a matrix containing the known proportions a_{ij} of ith element in jth charge material and $\mathbf{A}_2$ is a matrix containing information about alloying materials. $\Xi = \{\xi^1, \ldots, \xi^S\}$ is a finite support of distribution ξ that is composed of scenarios ξ^s (or shortly s), and $p_s = P(\xi = \xi_s) = \frac{1}{S}$, $s = 1, \ldots, S$. The random utilization of elements in the charge melt is therefore defined by $\mathbf{T}_1^s$. The alloying utilization is described by the unit matrix $\mathbf{I}$ that stays in front of $\mathbf{A}_2$. The overall expected cost of repeated similar melts is minimized, therefore, $\mathbf{c}$ and $\mathbf{q}$ are vectors representing costs per ton for input materials. The discussed melt control model development is described in [10]-[14]. The developed techniques are related to material engineering approaches in [7] and [8].

Utilization matrix $\mathbf{T}_1^s$ has a diagonal form and we denote $\tau_1^s = t_{11}^s, \tau_2^s = t_{22}^s, t_3 = t_{33}$, and $t_4 = t_{44}$. Firstly, we assumed only two random diagonal elements to simplify the results analysis. As the next step, we have assumed all diagonal elements random. The largest problem solved with real data dealt with 16 random elements of the diagonal of utilization matrix. We assign remaining utilization $t_3 = 0.7$ and $t_4 = 1$. We denote $\boldsymbol{\tau} = (\tau_1, \tau_2)^{\mathrm{T}}$ and we assume that $\tau_1 \approx U(0.734;0.866)$ and $\tau_2 \approx U(0.902;0.998)$ are independent random variables with continuous discrete probability distribution (for next computations: $\tau_3 \approx U(0.672;0.733)$ and $\tau_4 \approx U(0.972;1.000)$). Then, $\boldsymbol{\tau}^s$ are realizations of $\boldsymbol{\tau}$. Input data in Table I is significantly modified in comparison with [17] and [19] as we generalized the previous simple recourse case, see Table I.

III. Sampling techniques

We want to compute the optimal HN-solution $\mathbf{x}_{\min}^{\mathrm{HN}}$ and related objective function value $z_{\min}^{\mathrm{HN}}$. One possible way is to approximate it and reduce a program size with random sampling. We denote a random sample from ξ as $\xi_{[.]} = (\xi_{[1]}, \ldots, \xi_{[\nu]})^{\mathrm{T}}$. There are $\xi_{[s]}$ random variables identically distributed as ξ and they are stochastically independent. The realization of this random sample is usually denoted as $\xi_{[.]}^s = (\xi_{[1]}^s, \ldots, \xi_{[\nu]}^s)^{\mathrm{T}}$. We often simplify our notation as follows: $\xi_{[.]}^s = (\xi^{s1}, \ldots, \xi^{s\nu})^{\mathrm{T}}$. For computational purposes, we may easily replace $E_\xi\{f(\mathbf{x},\xi)\}$ (and hence $E_\xi\{Q(\mathbf{x},\xi)\}$ for two-stage programs) by the realization of a sample mean $\frac{1}{\nu}\sum_{s=1}^{\nu} f(\mathbf{x},\xi^s)$ (and $\frac{1}{\nu}\sum_{s=1}^{\nu} Q(\mathbf{x},\xi^s)$ for two-stage programs):

$$z_{\min}^{\nu} = \frac{1}{\nu}\sum_{s=1}^{\nu} f(\mathbf{x}_{\min}^{\nu};\xi^s) = \min_{\mathbf{x}}\left\{\frac{1}{\nu}\sum_{s=1}^{\nu} f(\mathbf{x},\xi^s) \,\middle|\, \mathbf{x} \in C\right\} \tag{4}$$

Randomly generated observations of ξ may then serve to compute the estimate of the objective (and recourse) function. Therefore, scenarios are selected by random procedure. Then, the scenarios ξ^s are used to build a scenario tree, and this reduced program is solved instead of the original one. However, its blindfold and exaggerated use can lead to misleading results. So, in addition, Monte Carlo techniques may be necessary to obtain an estimate of how good is such a simplification. Then, repeated computations inform us about the result stability and sensitivity, see the piece-wise linear line in the middle of Fig. 1. The reduction of the objective function estimate variance is also illustrated.

Morton, Mak, and Wood prove in [9] the following inequalities:

$$E_\xi\{\varsigma_{\min}^{\nu}\} \le E_\xi\{\varsigma_{\min}^{\nu+1}\} \le z_{\min}^{\mathrm{HN}} \le z = E_\xi\{f(\mathbf{x};\xi)\}, \tag{5}$$

where $\mathbf{x}$ is any feasible solution from C. They assume that a random sample from ξ denoted as $\xi_{[.]} = (\xi_{[1]}, \ldots, \xi_{[\nu_u]})^{\mathrm{T}}$ is available and the ς denotes a random optimal objective function value depending on the random sample. They have ν_l random samples, each having size ν, therefore $\forall i = 14, \ldots, \nu_l : \xi_{[i.]} = (\xi_{[i1]}, \ldots, \xi_{[i\nu]})^{\mathrm{T}}$. They use inequalities (5) and the central limit theorem to derive the following bounds:

$$\begin{aligned} P\Bigg(& \frac{1}{\nu_l}\sum_{i=1}^{\nu_l} \min_{\mathbf{x}(\xi_{[i.]})}\left\{\frac{1}{\nu}\sum_{s=1}^{\nu} f(\mathbf{x}(\xi_{[i.]});\xi_{[is]}) \,\middle|\, \mathbf{x}(\xi_{[i.]}) \in C \text{ a.s.}\right\} - \frac{t_{1-\alpha/2}\, s_l(\nu_l)}{\sqrt{\nu_l}} \le \\ & \le E_\xi\{\varsigma_{\min}^{\nu}\} \le z_{\min}^{\mathrm{HN}} \le E_\xi\{f(\mathbf{x};\xi)\} \le \\ & \le \frac{1}{\nu_u}\sum_{s=1}^{\nu} f(\mathbf{x};\xi_{[s+]}) + \frac{t_{1-\alpha/2}\, s_l(\nu_l)}{\sqrt{\nu_l}}\Bigg) \approx 1-\alpha. \end{aligned} \tag{6}$$

The symbol $t_{1-\alpha/2}$ denotes the $1-\alpha/2$ quantile of $N(0;1)$ distribution. Symbols $s_l(\nu_l)$ and $s_u(\nu_u)$ denote usual estimates of standard deviations $\sqrt{\operatorname{var}\varsigma_{\min}^{\nu}}$ and $\sqrt{\operatorname{var} f(\mathbf{x};\xi)}$. Hence, we may set α, then substitute observations $\xi_{[.]}^s$ and $\xi_{i[.]}^s$ in the formula (6), and we obtain reliable bounds.

IV. Extreme scenario sets

We may see that for our melt control example, a resulted sequence of optimum objective values for different samples is on Figure 1 (the line in the middle). The aforementioned bounds in this case are also very promising. However, still one question remains. Are they so good because of small influence of randomness or only 'dangerous' scenarios are not participating in our samples?

So, in this case, we may try to realize the worst case analysis based on so called extreme scenario sets, generally defined as

		Elements						Solution	
St.	Alloys	C	Mn	Si	Cr	Bounds	Prices	EV	
t	j	$i:\mathbf{A}_1^{\mathrm{T}},\mathbf{A}_2^{\mathrm{T}}$				$\mathbf{b}$	$\mathbf{c}$, $\mathbf{q}$	$\mathbf{x}_{\min}^{\mathrm{EV}}$	$\mathbf{x}_{\min}^{\mathrm{LO-UP}}$
1	Iron	5	0.947	3.124	0	∞	60.0	0.73713	0.67858
1	Spinput	0	4.737	21.429	10	∞	129.0	0.03000	0.03000
1	FeSi-1	0	0	64.286	0	∞	130.0	0.00000	0.00000
1	FeSi-2	0	0	60.000	0	∞	122.0	0.01103	0.01437
1	Alloy-1	0	63.158	25.714	0	∞	200.0	0.00712	0.00664
1	Alloy-2	0	9.474	42.857	20	∞	260.0	0.00000	0.00000
1	Alloy-3	0	34.737	35.714	8	∞	238.0	0.00000	0.00000
1	SiC	18.75	0	42.857	0	0.01	160.0	0.00000	0.00000
1	Steel-1	0.5	0.947	0	0	0.1000	42.0	0.10000	0.10000
1	Steel-2	0.125	0.316	0	0	0.1000	40.0	0.01472	0.07041
1	Steel-3	0.125	0.316	0	0	0.1000	39.0	0.10000	0.10000
2	A-C	100	0	0	0	∞	800	0.007	0.00457
2	A-Mn	2	70	10	4	∞	1500	0.005	0.00130
2	A-Si	1	0	60	0	∞	1900	0.001	0.0005
2	A-Cr	1	6	1	40	∞	4000	0.001	0.0007
	$\mathbf{l}_2$	3	1.35	2.7	0.3				
	Mix	3.75	1.421	3.857	0.3				
	$T_1(E\xi)$	0.8	0.95	0.7	1				
	melt result	3	1.35	2.7	0.3		$z_{\min}$	67.556	
	$\mathbf{u}_2$	3.5	1.65	3.0	0.45		$E\{z_{\min}\}$	83.176	69.5702

Table I

follows (see [12] and [14]):

$$\min_{S\subset\Xi;|S|=n} \min\left\{\frac{1}{S}\sum_{\xi^s\in S} f\left(\mathbf{x};\xi^s\right)\middle|\,\mathbf{x}\in\bigcap_s C\left(\xi^s\right)\right\} \tag{7}$$

$$\max_{S\subset\Xi;|S|=n} \min\left\{\frac{1}{S}\sum_{\xi^s\in S} f\left(\mathbf{x};\xi^s\right)\middle|\,\mathbf{x}\in\bigcap_s C\left(\xi^s\right)\right\} \tag{8}$$

Because the objective function convexity with respect to ξ is not guaranteed in general case, this problem might be quite difficult to solve. Because of the problem size, it is also impossible to consider all scenarios as Rosa and Takriti in [15] and try to exclude certain scenarios computing the whole optimization program that sets their probabilities to zero.

V. Algorithms

The basic idea of a genetic algorithm (GA) is quite simple. GA works not only with one solution in time but with the whole population of solutions. The population contains many (ordinary several hundreds) individuals – bit strings representing solutions. The mechanism of GA involves only elementary operations like strings copying, partially bit swapping or bit value changing. GA starts with a population of strings and thereafter generates successive populations using the following three basic operations: reproduction, crossover, and mutation. Reproduction is the process by which individual strings are copied according to an objective function value (fitness). Copying of strings according to their fitness value means that strings with a higher value have a higher probability of contributing one or more offspring to next generation. This is an artificial version of natural selection. Mutation is an occasional (with a small probability) random alteration of the string position value. Mutation is needed since, in spite of reproduction and crossover effectively searching and recombining the existing representations, they occasionally become overzealous and lose some potentially useful genetic material. The mutation operator prevents such an irrecoverable loss. The recombination mechanism allows mixing of parental information while passing it to their descendants, and mutation introduces innovation into the population. When you submit your final version, after your paper has been accepted, prepare it in two-column format, including figures and tables.

In spite of simple principles, the design of GA for successful practical using is surprisingly complicated. GA has many parameters that depend on the problem to be solved. In the first, it is the size of population. Larger populations usually decrease the number of iterations needed, but dramatically increase the computing time for each of iteration. The factors increasing

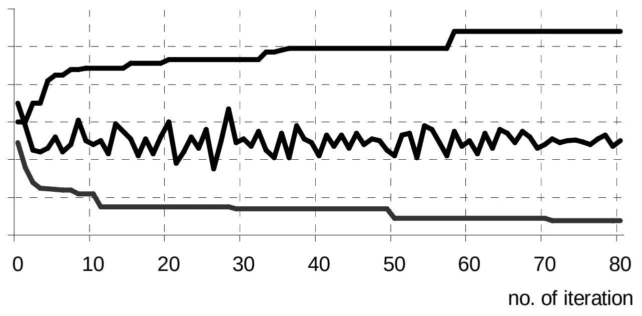

Fig. 1 - Development of various objective function values computed by GA

demands on the size of population are the complexity of the problem being solved and the length of the individuals. Every individual contains one or more chromosomes containing value of potential solution. Chromosomes consist of genes. The gene in our version of GA is a structure representing one bit of solution value. It is usually advantageous to use some redundancy in genes and so the physical length of our genes is greater than one bit. This type of redundancy was introduced by Ryan [20].

To prevent degeneration and the deadlock in local extreme the limited lifetime of individual can be used. Limited lifetime is realized by the “death” operator [21], which represents something like continual restart of GA. This operator enables decreasing of population size as well as increasing the speed of convergence. It is necessary to store the best solution obtained separately – the corresponding individual need not to be always present in the population because of the limited lifetime.

Many GAs are implemented on a population consisting of haploid individuals (each individual contains one chromosome). However, in nature, many living organisms have more than one chromosome and there are mechanisms used to determine dominant genes. Sexual recombination generates an endless variety of genotype combination that increases the evolutionary potential of the population. Because it increases the variation among the offspring produced by an individual, it improves the change that some of them will be successful in varying and often-unpredictable environments they will encounter. Using diploid or “multiploid” individuals can often decrease demands on the population size. However the use of multiploid GA with sexual reproduction brings some complications, the advantage of multiploidity can be often substitute by the “death” operator and redundant genes coding.

New individuals are created by operation called crossover. In the simplest case crossover means swapping of two parts of two chromosomes split in randomly selected point (so called one point crossover). In GA we use the uniform crossover on the bit level is used.

The strategy of selection individuals for crossover is very important. It strongly determines the behavior of GA. For grammatical evolution the ranking selection with elite brings satisfactory results.

Genetic algorithms commonly use heuristic and stochastic approaches. From the theoretical viewpoint, the convergence of heuristic algorithms is not guaranteed for the most of application cases. That is why the definition of the stopping rule of the GA brings a new problem. It can be shown [22], that while using a proper version of GA the typical number of iterations can be determine.

GAs we use have the following scheme:

1. *Generation of the initial population*: At the beginning the whole population is generated randomly, the members are sorted by the fitness (in descendent order).
2. *Mutation:* The mutation is applied to each gene with the same probability, all GAs we used use $p_{mut} = 0.05$. The mutation of the gene means the inversion of one randomly selected bit in the gene.
3. *Death*: Classical GA uses two main operations – crossover and mutation (the other operation should be migration). In GA described in this paper, we use the third operation – death. Every individual has the additional information – age. A simple counter that is incremented in each of GA iteration represents the age. If the age of any member reaches the preset lifetime limit LT, this member "dies" and is immediately replaced by a new randomly generated member. The age is not mutated nor crossed over. The age of new individuals (incl. individuals created by crossover) is set to zero. Lifetime limit in our application is set to 5.
4. *Sorting by the fitness.*
5. *Crossover*: Uniform crossover is used for all genes (each bit of the offspring gene is selected separately from corresponding bits of both parents genes).
6. When a stopping rule is not satisfied, go to step 2.

In crossover, we do not replace all members of the population. The crossover generates the number of individuals corresponding to the quarter of the population only. Created individuals are sorted into the corresponding places in the population according to their fitness in such a way that the size of the population remains the same. Newly created individuals of low fitness do not have to be involved in the population.

So, we assume that components of discrete random vector ξ are independent random variables with finite supports. We suggest a usual algorithm for general case of finite marginal supports Ξ_i. Evolutionary and genetic algorithms were already used in stochastic programming by Berland in [1] and a group of authors [18]. The changing set of scenarios S (7) is considered as a population. For each member of this population, the objective function value is computed by running an external program in GAMS [2] that allows various modifications of the model (3). Therefore, for given S and the optimal solution $\mathbf{x}_{\min}$ (similarly for $\mathbf{x}_{\max}$ and (8)) of (7), we calculate $\left\{f\left(\mathbf{x}_{\min};\xi^s\right)\right.$. Then the population is ordered by fitness values, and usual operations as reproduction, crossover, and mutation are realized.

Main search algorithm. At the end of this section, we describe the basic loop of main algorithm that searches extreme scenario sets:

1. During initialization, a main program controlling the whole computational process is started. It reads optimization engine name (GAMS [2] in our example), main source filename, included filename with scenarios, filename for melt control results, filename for GA input, number of scenarios, dimension, and distribution of τ.
2. The main program calls the optimization engine, and it solves the SB-program. Input data for GA is generated; it includes fitness values related to scenarios.
3. Then, the GA starts and generates a new set of scenarios.
4. When a stopping rule is not satisfied, continue by step 2.

VI. Results and Further Research

We have applied the aforementioned main algorithm together with GAMS source for (3) and discussed GA. As a result we obtained a sequence of randomly selected scenario sets together with sequences of searched pairs of extreme sets. Figure 1 presents the development of related objective function values (see [7] and [8]) for the case of diagonal with 4 random elements and implemented genetic algorithm. It is clearly visible, how the modified GA quickly searches for the both bounds. The aforementioned confidence intervals are significantly tighter even in the case of increased confidence level, hence may be interpreted as too optimistic for the situation when the worst case analysis is the goal. The algorithm proved the significant improvement in comparison with [19].

Further research will be focused on postoptimality analysis realized with respect to the original set of scenarios. Mainly, the cases where the incomplete recourse [6] appears to be very important, will be studied. We assume that the deeper understanding of the whole algorithm behaviour will lead to the used GA improvement.

The used test problems are derived from the underlying linear programs, therefore, the proposed technique will be applied to nonlinear programming engineering applications, e.g., in reliable structural design.

References

[1] N. J. Berland, *Stochastic Optimization and Parallel Processing*. PhD thesis, University of Bergen, 1993.

[2] A. Brooke, D. Kendrick, and A. Meeraus. *Release 2.25 GAMS A User's Guide*. The Scientific Press. Boyd & Fraser Publishing Company, 2nd edition, 1992.

[3] J. Dupačová, G. Consigli, and S. W. Wallace, “Scenarios for multistage stochastic programs,” *Annals of Operations Research*, 100: 25–53, 2000.

[4] W. H. Evers, “A new model for stochastic linear programming,” *Management Science*, 13:680–693, 1967.

[5] K. Hoyland and S. W. Wallace, *Generating scenario trees for multi stage problems*. Technical Report 4/97, Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology, Trondheim, 1996.

[6] P. Kall and S. W. Wallace, *Stochastic Programming*. John Wiley and Sons, Chichester, 1994.

[7] Z. Karpíšek. “Heterogenity characteristics of cast iron alloying elements,” *Folia Fac. Sci. Nat. Univ. Masarykianae, Mathematica*, 7:31–36, 1998.

[8] F. Kavička, K. Stránský, J. Stětina, V. Dobrovská, J. Dobrovská, and B. Velička. “Contribution to optimization of continuous casting of steel,” *Acta Metallurgica Slovaca*, 5:367–370, 1999.

[9] W. K. Mak, D. P. Morton, and R. K. Wood, “Monte Carlo bounding techniques for deterministic solution quality in stochastic programs,”. *Operations Research Letters*, 24:47–56, 1999.

[10] P. Popela, “A multistage blending problem with an expert estimate of parameters” (in Czech), in *Proceedings of 3μ Conference on Modern Mathematical Methods*, Ostrava, Czech Republic, 1994, pp 101–105.

[11] P. Popela and R. Setnička, *Analysis of steel production* (in Czech). Technical report, ŽĎAS, Žďár nad Sázavou, Czech Republic, 1995.

[12] P. Popela, “Advanced scenario tuning in stochastic programming,” in *Proceedings of Czech-Slovak-Japan Workshop*, Bratislava, Slovakia, 1998, pp. 307–310.

[13] P. Popela, “Application of stochastic programming in foundry,” *Folia Fac. Sci. Nat. Univ. Masarykianae Brunensis, Mathematica*, 7:117–139, 1998.

[14] P. Popela, *An Object-Oriented Approach to Multistage Stochastic Programming: Models and Algorithms*, PhD thesis, Charles University, Prague, 1998.

[15] C. H. Rosa and S. Takriti, *A minimax formulation for stochastic programs using scenario aggregation*, Technical report, SABRE Dec. Technol. and IBM T. J. Watson Center, July 17 1997.

[16] Y. Yoshitomi, T. Takeba, S. Tomita, and H. Ikenoue, “Genetic algorithm approach for solving stochastic programming problems,” in Proceedings of *International Conference on Stochastic Programming*, Vancouver, Canada, 1998.

[17] J. Zeman, *Optimization models in metallurgy* (in Czech), Master thesis, Dept. of Math., FME Brno University of Technology, Brno, Czech Republic, 1998.

[18] J. Sklenar, “Simulation of Queuing Systems in QUESIM,” in *Proceedings of the 2005 European Simulation and Modelling Conference ESM2005*, Riga, Latvia, 2005, pp. 35-37.

[19] P. Popela and J. Roupec, “GA-Based Scenario Set Modification in Two-Stage Melt Control Problems,” in *Proceedings of the 5th International Conference MENDEL’99*, Brno, Czech Republic, 1999, pp.112–117.

[20] C. Ryan, “Shades. Polygenic Inheritance Scheme,” in *Proceedings of the 3th International Conference on Soft Computing MENDEL ’97*, Brno, Czech Republic, 1997. pp. 140 – 147.

[21] J. Roupec, *Design of Genetic Algorithm for Optimization of Fuzzy Controllers Parameters*. (In Czech) PhD Thesis. Brno University of Technology, Brno, Czech Republic, 2001.

[22] J. Roupec, P. Popela, and P. Ošmera, “The Additional Stopping Rule for Heuristic Algorithms,” in *Proceedings of the 3th International Conference on Soft Computing Mendel ’97*, Brno, Czech Republic, 1997, pp. 135–139.