Generating Computational Models for Serious Gaming

Wim Westera(✉)

Open University of the Netherlands,
Valkenburgerweg 177, 6419 AT Heerlen, The Netherlands
wim.westera@ou.nl

Abstract. Many serious games include computational models that simulate dynamic systems. These models promote enhanced interaction and responsiveness. Under the social web paradigm more and more usable game authoring tools become available that enable prosumers to create their own games, but the inclusion of dynamic simulations remains a specialist's job involving knowledge of mathematics, numerical modeling and programming. This paper describes a methodology for specifying and running a specific subset of computational models without the need of bothering with mathematical equations. The methodology comprises a knowledge elicitation procedure for identifying and specifying the required model components, whereupon the mathematical model is automatically generated. The approach is based on the fact that many games focus on optimisation problems that are covered by a general class of linear programming models. The paper thus sketches the principles of a creativity tool that removes barriers for harvesting the creative potential of teachers and students.

1 Introduction

For over 30 years games and simulations have been used in training and education. Their application is motivated by the engaging interactions and authentic experiences they offer (e.g. [1, 2]). In 1970 Abt [3] introduced the term 'serious games' to indicate games for job training, such as the training of army personnel or insurance salesmen. Serious games now span everything from learning to advancing social causes, and from promoting better health to marketing and cultural engagement [4, 5]. So far, the adoption of games for learning in formal education has been quite limited. Among adoption barriers are the limited availability of games and technologies, costs of games, limited time and resources for implementing games, the intrinsic complexity of games and their design, the unfamiliarity of teachers with games, the supposed conservative culture of education, limited empirical evidence for the effectiveness of games, and difficulties of integrating games into the curriculum [6–12].

Still, in recent years game-based learning has gained popularity among educators and learners, as the costs of multimedia and graphics went down. Various affordable game authoring tools have become available ranging from simple puzzle creation tools to full 3D programmable engines. Game engines have become popular tools in programming courses [13–15]. Some game engines are particularly tuned to education and

© Springer International Publishing Switzerland 2014
A. De Gloria (Ed.): GALA 2013, LNCS 8605, pp. 53–63, 2014.
DOI: 10.1007/978-3-319-12157-4_5

learning, e.g. EMERGO [6], Emperor [16], e-Adventure [17], Starlogo and Scratch (MIT). Hu [18] lists a number of special requirements for educational games. Nadolski et al. [19] identified over 500 game engines and showed that many engines allow for educational scenarios: the key of serious games is in game design rather than game technology. Playing a game means engaging in a process and learning by doing while influencing the process in a favourable way.

Many games include computational models that simulate dynamic systems, which procure enhanced interaction and responsiveness. Well-known examples are in management and business games, where computational models cover the dynamics of supplies, customer flows, production processes, and sales revenues. Other examples are in social simulations, ecology or system evolution games, surveillance games, traffic and logistics games and many other types of games. Games that comprise such dynamic models provide immediate and relevant feedback, which enables players to learn from their successes and mistakes.

A common tendency of today's social web is that users become active contributors of content, while they are supported by online authoring tools and services that have become publicly available. Teachers and students increasingly produce and use their own videos, web pages and interactive stories. Although more and more free game authoring tools have become available online, self-authored (serious) games are scarce and hardly exceed the level of simple multiple choice quizzes or puzzle games. Moreover, whatever brilliant ideas teachers or students may have for creating a serious game, the development of an appropriate computational model for enhanced dynamics and interactions seems to remain a specialist's job that inevitably requires knowledge of mathematics, numerical modeling and programming.

This paper addresses this problem by providing a methodology for specifying a computational model without going into mathematical equations. The methodology comprises a knowledge elicitation procedure for identifying and specifying the required model components, whereupon the mathematical model is automatically generated. The approach is based on the fact that many games focus on optimisation problems that are covered by a general class of linear programming models. The notion of linear programming is not so much about programming but refers to a mathematical problem solving method, which is applicable to a wide range of optimisation problems in different domains and contexts, including operations research, and non-zero sum games [20–22]. Quite some simulation modeling software based on linear programming is available on the market, e.g. Vstep, FlexSim, and Siemens Plant Simulation, but in all cases these tend to specialise in particular domains such as vehicle simulation, logistics, manufactory planning, crop simulation, and process automation, which are hard to be used by non-specialists. Also, they are closed solutions that don't allow for interfacing with external software. The elicitation methodology proposed in this paper brings the model parameters, model coefficients and model logic to the surface in a pragmatic way, without requiring computational modeling skills. The approach benefits from the generic nature of linear programming and the wide spectrum of optimisation problems that it covers. In addition, various algorithms such as the simplex method [23] are available for solving linear programming problems. The optimum solutions that these algorithms provide can be used as a pedagogical benchmark for providing guidance and feedback to learners involved in optimisation tasks. The paper is a setup as follows.

First we will summarise the formal basics of linear programming. Second we will show how the linear programming framework can be linked with simulation modeling. Finally we will describe the elicitation approach and explain its implications for serious game design.

2 Linear Programming

Many processes in business, economy and nature can be described as linear programming problems. Linear programming refers to a mathematical methodology for minimising or maximising linear functions (e.g. minimising costs, maximising profits) subject to linear constraints (e.g. limited resources, limited time). In their elementary form such models reflect conversion processes or mappings that link a set of input variables to a set of output variables (cf. Fig. 1).

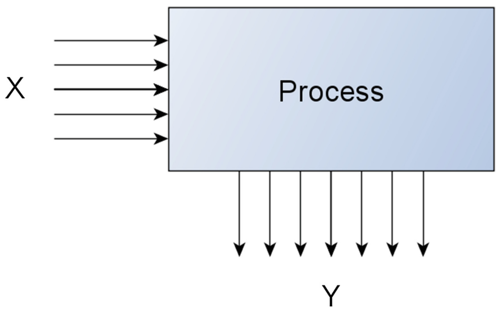


Fig. 1. A black-box system with input vector X and output vector Y

Linear programming problems may have very different manifestations, but all rely on the same mathematical methodology, which offers optimal solutions. Below we will list some problem examples (e.g. [21]):

- Diet problems: Compose a menu of food products (inputs) at minimum costs, which preserves the minimal daily doses of nutrients, e.g. proteins, vitamins, minerals, calories and so on (outputs).
- Shipping problems: Optimising the transportation of products from a series of warehouses (inputs) to a series of markets (outputs), while fulfilling market demands.
- Activity planning problems: Distributing a factory's resources (inputs) e.g. machines, money, energy, or staff to activities (outputs) that produce highest profits.
- Staff assignment problems: Allocating staff (inputs) to production activities (outputs), taking into account the people's different effectiveness on different tasks.

These examples cover a wide range of optimisation problems that can be varied by changing the number of inputs and outputs, choosing different types of processes, or linking multiple problems together.

2.1 Mathematical Description

The standard minimum problem has the following formal structure: Find the n-vector X (X_1, X_2, …, X_n) that minimises the objective function O given by

$$O = \sum_{j=1}^{n} c_j \cdot X_j \tag{1}$$

Minimisation is required under m functional constraints given by

$$\sum_{j=1}^{n} A_{ij} \cdot X_j \geq p_i \tag{2}$$

Also, n nonnegativity constraints apply:

$$X_j \geq 0 \tag{3}$$

Here c_j is an n-dimensional vector of utility coefficients, p_i are m constraint coefficients and A_{ij} is an m × n matrix linking variables to constraints. The equations represent the standard minimum problem in canonical form [24]. It is technically equivalent with the standard maximum problem, which aims at maximising the objective function. For both the standard minimum problem and standard maximum problem solution algorithms are available. The Simplex tableau method is known for its efficiency, although occasionally cycling degeneration may occur [21, 24]. Calculated solutions could be used as a benchmark for evaluating user-created solutions. Herein lies its educational potential: learners working on a task to optimise a simulated process could receive informative feedback how well they do as compared with the calculated optimum. Below we will further detail the relevant process variables and coefficients that have to be specified and explain how practical problems can be translated into standard form.

2.2 Connecting Input and Output Variables

In accordance with Fig. 1 we assume a process or mechanism that connects an input vector X = (X_1, X_2, …, X_m) with an output vector Y = (Y_1, Y_2, …, Y_n). These variables reflect the amounts of each input and output, e.g. (1) the amounts of food products and nutrients, (2) the quantities of product shipped from a harbour or shipped to a market, (3) the amount of resources allocated to different activities, or (4) the amount of time that people are allocated to tasks, and so on. The distinction between inputs and outputs is not essential: allocating people to tasks is technically the same as allocating tasks to people.

The characteristics of the process are covered by an m × n matrix a_{ij}, which interconnects the two vectors X and Y. For explaining the nature of these interconnections we need to distinguish between two separate problem classes, each of which require a different approach. The classes differ by the type of interventions users are allowed to make, while dealing with the optimisation problem: (1) Adjusting the vector

X (or Y), or (2) Adjusting the matrix a_{ij}. Below we will subsequently elaborate the model descriptions of the two problem classes.

2.3 Model Class 1: Adjusting the Vector X (or Y)

In this problem class the user has to adjust the input variable X to arrive at an optimal output Y. Examples are:

- The activity planning problem: The user decides upon the input resources X_i that produce the best output Y;
- The diet problem: The user decides upon the amounts of food products X_i that offer required nutrients Y_j.

Amount Attribution. The process reflects a mapping of X onto Y, that is, it converts inputs X into outputs Y, which is defined by the elements of matrix a_{ij}. It assumes that each input variable X_i is related to each output variable Y_j by an amount a_{ij}, which describes the attribution of input variable X_i to the output variable Y_j. This is expressed as follows:

$$Y_j = \sum_{i=1}^{m} a_{ij} \cdot X_i \tag{4}$$

In the diet case a_{ij} would describe the amount of nutrient Y_j contained in one unit of food product X_i.

Assigning Value. The amount of a variable may differ from its value. Indeed kilograms or cubic meters are different from Euros or Dollars. Since many optimisation problems are based on value rather than amounts, we have to incorporate value rates. Mostly (but not necessarily) these will be monetary values (money). We introduce the input value rate VX_i, which is the (monetary) value of one unit of input X_i. In the activity planning problem VX_i would be the value (or costs) of one unit of the factory's resource X_i. In the diet problem VX_j would be the value (or costs) of one unit of food product X_i. Alternatively, one might want to define and use the values VY_j.

Objective Function. The total value VX_{total} is the summed value of inputs given by:

$$VX_{total} = \sum_{i=1}^{m} X_i \cdot VX_i \tag{5}$$

This total value is likely to be the objective function to be minimised or maximised in the problem solution, e.g. the total costs of a factory's resources. Alternatively, output values VY_{total} may be calculated likewise, e.g. the profits of products' sales. Note that in some cases, users may be required to optimise total amounts rather than total values. If so, the value rates VX or VY are set to unity.

Functional Constraints and Nonnegativity Constraints. Constraints refer to lower or upper boundaries that apply to the input or output variables. We distinguish the following cases of functional constraints:

1. Inputs are subjected to an upper limit:

$$X_i \le b_i \, for \; i = 1, \ldots, m \tag{6}$$

2. Inputs are subjected to a lower limit:

$$X_i \ge c_i \, for \; i = 1, \ldots, m \tag{7}$$

3. Outputs are subjected to an upper limit:

$$Y_j \le d_j \, for \; j = 1, \ldots, n \tag{8}$$

4. Outputs are subjected to a lower limit:

$$Y_j \ge e_j \, for \; j = 1, \ldots, n \tag{9}$$

These constraints of inputs and outputs can be understood in terms of supplies and demands: e.g. limited supplies available (constraint 1), reducing supplies (constraint 2), avoiding overproduction (constraint 3), and meeting output demands (constraint 4). In addition we have the following nonnegativity constraints:

5. All inputs are nonnegative:

$$X_i \ge 0 \, for \; i = 1, \ldots, m \tag{10}$$

6. All outputs (amounts) are nonnegative:

$$Y_j \ge 0 \, for \; j = 1, \ldots, n \tag{11}$$

Note that in all cases the input and output vectors represent amounts of entities, which cannot be negative.

Conversion to Standard Form. The vector problem of minimisation can now be summarised as follows. Determine an input vector X that has to be adjusted to optimise the objective function given by Eq. (5), e.g. the costs of the food menu under the functional constraint of Eq. (9), e.g. minimum daily doses of nutrients, and the non-negativity constraint of Eqs. (10) and (11). This is in accordance with the standard minimum form. Similar considerations hold for maximisation problems.

2.4 Model Class 2: Adjusting the Matrix a_{ij}

In this class of problems the user adjusts the coefficients a_{ij} rather than X_i or Y_j for producing an optimal solution. Examples:

- The shipping problem:
  Distributing product quantities from various warehouses to different markets reflects decisions about the attributions a_{ij}, viz. the amount of product to be shipped from a location X_i to a destination Y_j;
- The staff assignment problem:
  Allocating staff X_i to tasks Y_j, while taking into account the productivity differences between people at different tasks.

Amount Attribution. The matrix problems reflect the allocation of each input element X_i onto the output vector Y: amounts of input entities X_i are distributed over the outputs Y_j. This means that Eq. (4) is no longer valid for describing the attribution of input vector X to output variable Y_j. Instead this attribution is given by:

$$X_i = \sum_{i=1}^{m} a_{ij} \tag{12}$$

and

$$Y_j = \sum_{j=1}^{n} a_{ij} \tag{13}$$

Assigning Value. Each attribution a_{ij} decided upon by the user goes with a value that is determined by a value rate matrix $Va_{ij,}$ indicating the (monetary) value per unit of a_{ij}. In the shipping problem Va_{ij} would be the value (or costs) per unit product shipped from warehouse i to market j. In the job allocation case Va_{ij} would be the value per unit of time that person X_i is allocated to job Y_j.

Objective Function. The total value Va_{total} of all decisions is then given by Eq. (14).

$$Va_{total} = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} \cdot Va_{ij} \tag{14}$$

This is likely to be the objective function to be minimised or maximised, for instance total value of all shipping, or total value of job allocations. Note that in some cases, the problem may require optimisation of total amounts rather than monetary value. If so, the value rates Va_{ij} are all set equal to unity.

Constraints. Constraints refer to lower or upper boundaries that apply to input or output variables. Options are given by Eqs. (6)–(11).

Conversion to Standard Form. The problem description of this model class can be demonstrated to correspond with the standard form: it reflects minimisation (or maximisation) of an objective function, cf. Eq. (14), subject to functional and nonnegative constraints.

3 Model Elicitation

For producing a computational model without bothering about mathematical complexities we have developed a procedure that supports game authors at expressing their ideas and at the same time extracts the nature, the variables and the coefficients needed for the model description. Table 1 lists the high level model decisions that have to be taken, including the intervention variable, the objective dimension, the optimisation criterion and the constraints.

Table 1. Successive decisions to be taken

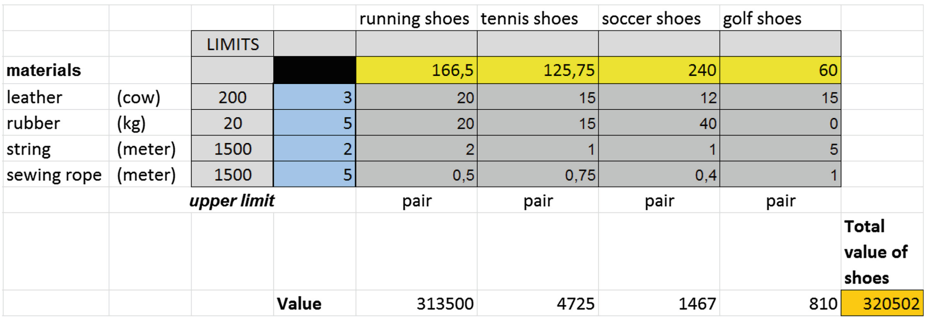
Decisions	Options	Number of options
Intervention variable	X, Y, a_{ij}	3
Objective dimension	X, Y	2
Optimisation criterion	minimise, maximise	2
Constraints on Y	lower, upper, both, none	4
Constraints on X	lower, upper, both, none	4
Total number of options		192

It follows that the total number of options is 192 (neglecting the dimensionalities m and n of X and Y, respectively), which means that the mathematics described above allow for the specification of 192 different model types. Based on this set of decisions we have developed a structured elicitation procedure, which comprises a sequence of standardised questions, e.g.:

- "What type of process do you want to define?" (discriminating between matrix and vector model)
- "How would you qualify the output of your process?" (making explicit the sort of outcome)
- "What different types of outputs do you consider?" (extracting outputs Y_j and associated labels)
- "What units would you use to express the respective output types?" (required standards for calculations)
- "What is the value of one unit of Y_j?" (converting amounts to monetary values)
- Etcetera.

A prototype of the elicitation procedure was implemented as an Excel form. A formative test procedure included interviews with five volunteers. Model elicitation took typically 20 to 30 min. After each interview weaknesses in the elicitation procedure were identified and discussed, whereupon the form was improved. After completion of an elicitation session, the Excel prototype generated the model, which allowed for testing and making adjustments (cf. Fig. 2).

Figure 2 shows the Excel-representation of a fictitious shoe factory using 4 inputs (materials: leather, rubber, string, sewing rope) constrained to upper limits (e.g. limited supplies) and 4 outputs (running shoes, tennis shoes, soccer shoes, golf shoes) aiming for maximizing total output value by deciding about the output volumes.



				running shoes	tennis shoes	soccer shoes	golf shoes	
		LIMITS						
materials				166,5	125,75	240	60	
leather	(cow)	200	3	20	15	12	15	
rubber	(kg)	20	5	20	15	40	0	
string	(meter)	1500	2	2	1	1	5	
sewing rope	(meter)	1500	5	0,5	0,75	0,4	1	
		upper limit		pair	pair	pair	pair	
								Total value of shoes
			Value	313500	4725	1467	810	320502

Fig. 2. Example screenshot of the Excel simulation prototype.

4 Discussion and Outlook

The implementation of the elicitation procedure and the associated mathematics in the Excel prototype was quite straightforward. The model was tested against a wide range of input conditions and proved to produce the required outputs, which demonstrated the feasibility of the approach. Test persons lacking any mathematical background were pleased to see how their verbalised ideas were immediately brought to life as a working simulation model. Also, the prototype made use of Excel's equation solver, which provided approximate solutions of the optimisation problem. This yields a benchmark for player performance that can be used for providing feedback to gamers.

Two main findings led to a readjustment of the elicitation procedure. First, although test persons demonstrated to be able to specify process models, it turned out to be very hard for them to imagine any process just from scratch. It was very difficult for them to make substantiated choices, for example between a matrix model and a vector model, even though quite common terms were used to guide the decision (e.g. "allocation", "production", "composing"). Specifying constraints also appeared quite difficult. Adding instructional materials appeared helpful, but not in all cases. Secondly, from the tests we found that the elicitation procedure sometimes produced degenerated models, that is, the models either don't allow for solutions, have trivial solutions, or don't reflect an optimisation problem. For instance, in case of upper limit output constraints any output minimisation problem will have a trivial solution of zero output. Analysis has shown that 154 out of the 192 model options that follow from the decision tree (cf. Table 1) are degenerate models. This leaves 36 valid model types, which can all be covered by a basic set of 12 model templates. For avoiding degenerate models we aim to use this basic set of templates as a starting point for guiding the elicitation process. We will attach concrete examples to each of the templates in order to promote a better understanding and informed decision making by game authors.

The approach explained in this paper allows for the easy extraction of model variables, coefficients and relationships, provided that the optimisation process fits in the class of linear programming problems. This opens up possibilities for developing simplified computational model builders that can be used for creating models in games and simulations. A next step would be the development of an authoring tool that implements the elicitation dialogue and model composition as a user-friendly, computer-guided service, preferably using sprites and other graphical objects for visualising the process. It should also contain a Simplex equation solver for making available a performance feedback reference. So far the problem set explained in this paper reflects single-shot problems that don't take into account progression over time. However, there are two ways to transform the approach into a time-dynamic challenge. First, in a game or simulation environment players may be asked to continue their optimisation task and repeatedly enter new inputs to find a better solution. Adding time constraints and associated scores may help enhancing the dynamic experience. Second, the models can be easily adapted to allow for a progressive accumulation of variables, e.g. sales, supplies, costs over time, which would reflect a history that contributes to the narrative of the player's performance in a game. Such accumulation may be well understood as a discrete time Markov chain, which would preserve the linear programming model as single shot, while only reformulating the constraints and objective function in the course of time. In addition, problem cases need not be restricted to a single linear problem core but could be composed of multiple cascaded or interlinked processes, each covering a single problem issue: outputs of one process acting as inputs of follow-up models in the process chain. In all cases standard linear programming models remain the heart of the description. Eventually, there are no principal barriers for using an elicitation dialogue for more complex linear, probabilistic or even nonlinear (e.g. exponential, logarithmic or power law) models.

References

1. Aldrich, C.: The Complete Guide to Simulations and Serious Games: How the Most Valuable Content Will Be Created in the Age Beyond Gutenberg to Google. Pfeiffer, San Francisco (2009)
2. David, M.M., Watson, A.: Participating in what? Using situated cognition theory to illuminate differences in classroom practices. In: Watson, A., Winbourne, P. (eds.) New Directions for Situated Cognition in Mathematics Education. Springer, New York (2010)
3. Abt, C.: Serious Games. Viking Press, New York (1970)
4. Michael, D., Chen, S.: Serious Games: Games that Educate, Train and Inform. Thomson Course Technology, Boston (2006)
5. Klopfler, E., Osterweil, S., Salen, K.: Moving Learning Games Forward; Obstacles, Opportunities and Openness. MIT - The Education Arcade, Boston (2009). http://education.mit.edu/papers/MovingLearningGamesForward_EdArcade.pdf
6. Westera, W., Nadolski, R., Hummel, H., Wopereis, I.: Serious games for higher education: a framework for reducing design complexity. J. Comput.-Assist. Learn. **24**(5), 420–432 (2008)
7. Kapp, K.M.: The Gamification of Learning and Instruction: Game-Based Methods and Strategies for Training and Education. Pfeiffer, New York (2012)
8. Arnab, S., Berta, R., Earp, J., de Freitas, S., Popescu, M., Romero, M., Stanescu, I., Usart, M.: Framing the adoption of serious games in formal education. Electron. J. e-Learn. **10**(2), 159–171 (2012). www.ejel.com
9. Westera, W.: The eventful genesis of educational media. Educ. Inf. Technol. **17**(3), 345–360 (2012)

10. Proctor, M.D., Marks,Y.: A survey of exemplar teachers' perceptions, use, and access of computer-based games and technology for classroom instruction. Comput. Educ. **62**, 171–180 (2012). http://dx.doi.org/10.1016/j.compedu.2012.10.022
11. Lean, J., Moizer, J., Towler, M., Abbey, C.: Simulations and games use and barriers in higher education. Act. Learn. High. Educ. **7**(3), 227–242 (2006). doi:10.1177/1469787406069056
12. Kebrichi, M.: Factors affecting teachers' adoption of educational computer games: a case study. Br. J. Educ. Technol. **41**(2), 256–270 (2010). doi:10.1111/j.1467-8535.2008.00921
13. Jeon, J., Kim, K., Jung, S.: A study on the game programming education based on educational game engine at school. J. Educ. Learn. **1**(2), 282–287 (2012). doi:10.5539/jel.v1n2p282
14. Berigel, M.: Learning programming through game design: a case study. Int. Online J. Commun. Mark. Technol. **1**(1), 32–44 (2012). http://www.iojcmt.net/ojs/index.php/IOJCMT/article/view/4
15. Kumar, B.: Gamification in education - learn computer programming with fun. Int. J. Comput. Distrib. Syst. **2**(1), 46–53 (2012). http://www.cirworld.com/index.php/IJCDS/article/view/IJCDS218
16. Kiili, K., Ojansuu, K.: Emperor: game engine for educational management games. In: Kommers, P., Richards, G. (eds.) Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2005, pp. 1775–1782. AACE, Chesapeake (2005)
17. Torrente, J., Vallejo-Pinto, J.A., Moreno-Ger, P., Fernández-Manjón, B.: Introducing accessibility features in an educational game authoring tool: the <e-adventure> experience. In: Proceedings of the 11th IEEE International Conference on Advanced Learning Technologies (ICALT 2011). IEEE, Athens (2011)
18. Hu, W.: A common software architecture for educational games. In: Zhang, X., Zhong, S., Pan, Z., Wong, K., Yun, R. (eds.) Edutainment 2010. LNCS, vol. 6249, pp. 405–416. Springer, Heidelberg (2010). http://dx.doi.org/10.1007/978-3-642-14533-9_42
19. Nadolski, R.J., Hummel, H.G.K., Slootmaker, A., Van der Vegt, W.: Architectures for developing multiuser, immersive learning scenarios. Simul. Gaming **43**(6), 825–852 (2012)
20. Adler, I.: On the equivalency of linear programming problems and zero-sum games. Int. J. Game Theory 17(April) (2012). doi:10.1007/s00182-012-0328-8
21. Ferguson, T.S.: Linear Programming, A Concise Introduction. United States Naval Academy, Annapolis (2008). http://engine4.org/l/linear-programming---united-states-naval-academy-w2532-pdf.pdf
22. Anderson, D., Sweeney, D., Williams, T.: Quantitative Methods for Business. West Publishing Company, St. Paul (1995)
23. Dantzig, G.B.: Linear Programming and Extensions. Princeton University Press, Princeton (1963)
24. Murty, K.G.: Linear Programming. Wiley, New York (1983)