DISCUSSION PAPER

Leibniz Institute of Agricultural Development in Central and Eastern Europe

The Spatial Agent-Based Competition Model (SpAbCoM)

Marten Graubner

Discussion Paper No. 135
2011

Theodor-Lieser-Straße 2, 06120 Halle (Saale), Germany
Phone: +49-345-2928 110
Fax: +49-345-2928 199
E-mail: iamo@iamo.de
Internet: http://www.iamo.de

Marten Graubner is postdoctoral research associate in the Department of "Structural Development of Farms and Rural Areas" at the Leibniz Institute of Agricultural Development in Central and Eastern Europe (IAMO). His research focuses on spatial pricing and competition in agricultural markets and its investigation within a framework of computational economics. This paper is based on Chapter 6 of the author's dissertation.

Address:	Leibniz Institute of Agricultural Development in Central and Eastern Europe (IAMO) Theodor-Lieser-Straße 2 06120 Halle (Saale) Germany
Phone:	++49-345-2928-320
Fax:	++48-12-6624-399
E-mail:	graubner@iamo.de
Internet:	http://www.iamo.de

ISSN 1438-2172

ABSTRACT

The paper presents a detailed documentation of the underlying concepts and methods of the Spatial Agent-based Competition Model (SpAbCoM). For instance, SpAbCoM is used to study firms' choices of spatial pricing policy (GRAUBNER et al., 2011a) or pricing and location under a framework of multi-firm spatial competition and two-dimensional markets (GRAUBNER et al., 2011b). While the simulation model is briefly introduced by means of relevant examples within the corresponding papers, the present paper serves two objectives. First, it presents a detailed discussion of the computational concepts that are used, particularly with respect to genetic algorithms (GAs). Second, it documents SpAbCoM and provides an overview of the structure of the simulation model and its dynamics.

JEL: Y90

Keywords: Agent-based modelling, genetic algorithms, spatial pricing, location model.

ZUSAMMENFASSUNG

DAS RÄUMLICHE AGENTEN-BASIERTE WETTBEWERBSMODELL
SPABCOM

Das vorliegende Papier dokumentiert die zugrundeliegenden Konzepte und Methoden des Räumlichen Agenten-basierten Wettbewerbsmodells (Spatial Agent-based Competition Model) SpAbCoM. Anwendungsbeispiele dieses Simulationsmodells untersuchen die Entscheidung bezüglich der räumlichen Preisstrategie von Unternehmen (GRAUBNER et al., 2011a) oder Preissetzung und Standortwahl im Rahmen eines räumlichen Wettbewerbsmodells, welches mehr als einen Wettbewerber und zweidimensionalen Marktgebiete berücksichtigt. Während das Simulationsmodell in den jeweiligen Arbeiten kurz anhand eines Beispiels eingeführt wird, dient das vorliegende Papier zwei Zielen. Zum Einen sollen die verwendeten computergestützten Konzepte, hier speziell Genetische Algorithmen (GA), detailliert vorgestellt werden. Zum Anderen besteht die Absicht dieser Dokumentation darin, einen Überblick über die Struktur von SpAbCoM und die während einer Simulation ablaufenden Prozesse zu gegeben.

JEL: Y90

Schlüsselwörter: Agent-basierte Modellierung, Genetische Algorithmen, räumliche Preissetzung, Standortmodell.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

To analyze spatial competition, we must consider a number of requirements regarding the methodology. These can be summarized in the issues of *complexity* and the need for *flexibility*. First, there are spatially differentiated actors. Transport costs affect the cost structure of a firm such that each location may feature an individual price, supply, and consequently profitability. Particularly, under a two-dimensional framework, the solution space may be large and discontinuous, and functional relations may not be tractable with common calculus-based methods. These features introduce a high degree of complexity to the investigation of spatial competition. Second, the review of spatial economic literature shows a broad variety of models that differ by one or more important assumptions. In this respect, we might consider the firm's price strategy or competitive behaviour as well as whether transport costs are quadratic or linear in distance. A further important characteristic is the elasticity of the supply or demand functions. As heterogeneous as the assumptions of the models are, their results may also differ and sometimes even be contradictory. The uni-causal nature of most analytical models is a major reason for this variety. Of course, the approach is necessary due to the complexity issues named above. However, it may be desirable to have a more flexible tool that can assess the impact of particular conditions as well as their interdependencies on the model's outcome.

Agent-based modelling (ABM) is one option for investigating complex systems under flexible conditions. Not only can we account for heterogeneity within the model to explicitly consider the spatial dimension, but we can also incorporate powerful solution concepts as genetic algorithms (GA). GAs are particularly important for the Spatial Agent-based Competition Model (SpAbCoM) presented in this paper because they enable the identification of solutions where other methods may fail. The motivation to use an ABM approach is extensively discussed in literature (cf. AXELROD, 1997; TESFATSION, 2006) and shall not be repeated here. Instead, I present the implementation of the agent-based structure into a simulation model of spatial competition. While ABM enables the introduction of spatially differentiated actors into the model, the method also provides the possibility to consider other sources of heterogeneity. In this respect, the subsequent presented simulation model does not exploit the potential of ABM in many aspects. Some selected features, in which extensions are not only interesting but also easily importable, are presented in footnotes within this paper.[1] The major objective of this paper, however, is to describe the functioning of SpAbCoM. Particular attention is given to GAs as optimization technique with respect to the spatial competition strategy of agents.

In the next section, we reconsider the general decision problem that should be solved by SpAbCoM, before we introduce the computational tools that are used. The conceptual framework is based on a spatial input market and described in Section 3. The identification of optimal strategies under spatial competition is the major objective of the simulation; therefore, we discuss the GA optimization in more detail. In Section 4, I characterize GAs and their standard structure and present the adjustment to strategic interactions. In Section 5, I introduce the simulation software that is used, before describing the structure and finally the dynamics of SpAbCoM in Sections 6 and 7, respectively. The latter represents the stepwise description of the simulation.

[1] In this regard, it is worth mentioning that an extension of the simulation model in such aspects might be easily accomplished, the interpretation or evaluation of the results, however, is most likely a rather demanding task.

2 OBJECTIVE AND METHODS

First, we may reconsider the objective of the simulation. Under a framework of spatially distributed producers and processors, we assume market power on the processor's side. Accordingly, we use the simulation as a technical mean to identify the optimal strategy $\bar{\gamma} \in \Gamma$ of the maximization problem:

(1) $\Pi(\bar{\gamma}) = \max\{\Pi(\gamma, \boldsymbol{\gamma}') | \gamma \in \Gamma\}$.

In this spatial competition framework, we denote γ as a strategy consisting of four decision variables with $\gamma = (m, \alpha, x, y)$ and $\boldsymbol{\gamma}'$ is a vector of the other players' strategies. The last two variables of a strategy are the location of a firm in space represented by x-y coordinates. The other two variables refer to a firm's linear price strategy. Thereby, a local price $m(r)$ at distance r from the processor's location is defined as $m(r) = m - \alpha t r$. While t is the constant transport rate, m is the price at the processor's location and $\alpha \in [0,1]$ relates to the degree of spatial price discrimination β, with $\beta = 1 - \alpha$. If $\beta = 0$, there is no spatial price discrimination (cf. GRAUBNER et al., 2011a).

To implement the spatial competition model, we employ the Recursive Porous Agent Simulation Toolkit (Repast) (COLLIER et al., 2003). Repast was originally developed by SALLACH (2004), NORTH et al. (2006), and others as a free open source toolkit. The software provides the basics for agent-based programming as a graphical user interface, sample models, and a comprehensive class library. Particularly, Repast J from the Repast 3 package is used. This component enables the model development in Java. As C++ or Smalltalk, Java is an object oriented programming (OOP) language.

In basic terms, OOP follows the principle: "*from the general to the specific*". That is, one seeks to simplify complex structures and relations by breaking them down into simple parts. In OOP, objects are data structures of attributes and operations (methods), mostly in terms of a class. Classes are prototypes, whereas instances of them can be generated by assigning values to the characteristics of the classes. To illustrate the concept, we can think of cars. A car has, among other things, four wheels, a colour, one engine, and a steering wheel, and it should be drivable (accelerate and brake). While each car has these attributes (engine, wheels) and methods (accelerate and brake), one can subdivide cars. For instance, we can differentiate gas and diesel cars. Both are subclasses of cars featuring the same attributes and methods but additionally the type of fuel. Moreover, for each class it is possible to think of the different shaping of the common characteristics. One car is red; the other is blue. A red car is an instance of the class "cars". A blue car powered with diesel is an instance of the subclass "diesel cars".

In this respect, OOP is appealing for agent-based modelling. TESFATSION (2006) defines agents as an encapsulated piece of software that includes data together with behavioural methods which act on these data. In this sense, agents are objects as defined above. Agent-based models are characterized by a set of autonomous heterogeneous agents and their interactions.

The agents' heterogeneity can be distinguished at two levels. There may be inter-or intra-class heterogeneity. The latter simply refers to differences in the values of the characteristics, as in the case of red and blue cars. Hence, these are differences among instances of the same class. Inter-class heterogeneity goes beyond the value differences. Instead, the type of attributes or methods is different, or there may be additional attributes (e.g., diesel or gas). With respect to our spatial competition problem, inter-class heterogeneity refers to the differences between producers and processors. Intra-class heterogeneity, e.g., denotes the spatial differentiation of producers due to their location in the market region.

3 Conceptual Model

The core model to implement the simulation is an abstraction of a buyer-seller interaction, which corresponds to the discussion in Graubner et al. (2011a,b). Because the focus is on agricultural markets and their spatial nature, there is a high number of spatially distributed producers, whereas processing is concentrated at few locations. Accordingly, buyers have local market power. Actors of that group can influence the supply over the price, whereas producers are price takers. Nevertheless, the role of the producers is crucial with respect to market interactions and the allocation of supply. Based on the highest price at a producer's location, i.e., the local price, the supplier not only decides the quantity that should be produced but also where it will be delivered.

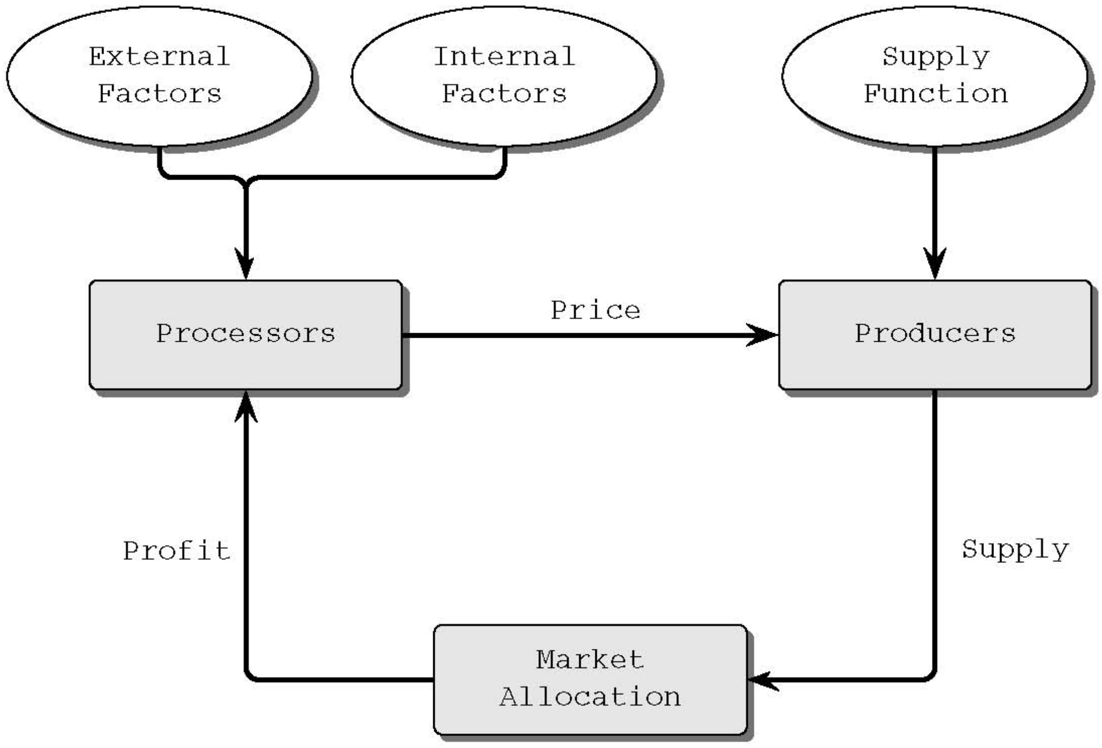
The local price of a producer hinges on the price setting of the processors. Basically, there are internal and external factors that determine the price at a particular location of a particular processor. The external conditions are, e.g., the transport rate or the price on the product market. The first refers to the cost of transportation required for one unit of the input over one unit of distance. The latter is the marginal revenue product, i.e., the price received by the processor for one unit of the processed input. Internal factors, on the other hand, can be directly influenced by the processor and correspond to its decision variables. There are four decision variables in the most general implementation used within SpAbCoM. Two determine the location of the processor within the market and the two others represent its price strategy. The identification of the optimal internal conditions is the core of the spatial competition problem and also the major focus of the simulation model.

Figure 1 summarizes the conceptual model of the simulation. Based on the combination of external and internal factors, the processing firm offers a price at particular locations and, thus, to particular producers. For instance, a supplier will not receive a price offer if the corresponding transport costs are too high. The general rule, however, is that the local profit of a processor must be non-negative.

The input quantity of the producer is determined based on the local price and its translation through the producer's supply function. Furthermore, the production is allocated to processors at the market level. Once the price, supply and its allocation is known, the processor's profit is determined and the price decision can be updated. The selection and update (or evaluation) of a price strategy is implemented by a genetic algorithm, which will be discussed in Section 4. We separate the selection and evaluation because under competition there is a difference between the set of producers who receive a price offer from a particular processor and the set of producers who actually deliver to that processor (see also Section 7.2). While the first depends on the profitability to serve a location by the processor, the latter hinges on the processor's price and whether there are better (price) options for the producer.

Finally, it is worth mentioning that we investigate a static competition model. The simulation, however, features a dynamic process. These dynamics are internal and required for optimizing the processor's decision by a GA. We may think of this process as an internal model of a decision maker in the framework of a classical oligopoly model such as Cournot or Bertrand competition. Basically, these models are static, yet we can calculate reaction functions determining the optimal strategy subject to the other firms' decisions. In most of the analyzed cases by Graubner et al. (2011a) and Graubner et al. (2011b), these reaction functions are multivariate and discontinuous. While analytically not tractable, the simulation can work along such reaction function to identify equilibria (if any exists). As reaction functions are a technical and illustrative device in strategic interaction of a static game (TIROLE, 2003), so is the GA optimization an abstraction of the internal decision rule of agents. Thus, we must be clear that even though there are internal dynamics of the simulation, the core problem is static throughout this paper.

Figure 1: Buyer-seller interaction with market power on the buyer's side.



4 IMPLEMENTING THE PROCESSOR'S DECISION MAKING BY GENETIC ALGORITHMS

In an input market framework of spatial competition, processing firms have local market power. The consequence is that strategic interactions must be incorporated within the simulation model. Hence, to determine its optimal decision, a processor must not only consider the effects on the decision of the producers but also on the other processing firms in the neighbourhood (neighbourhood interactions). While the decision of the producers can be implemented as a deterministic condition (see Section 6.2), this is not feasible in the case of the processors due to a highly complex solution space. Instead, this task, the determination of the optimal spatial strategy, is modelled by means of genetic algorithms. The objective of this section is to introduce the concept of GA, before we discuss the actual implementation within SpAbCoM.

4.1 Genetic Algorithms: An Introduction

Genetic algorithms are heuristic search methods for optimization or for identification of equilibria (REEVES AND ROWE, 2003). The major fields of application are complex problems with large, potentially poorly understood decision or strategy spaces. GAs are based on the mechanics of natural selection and natural genetics (GOLDBERG, 1989). A basic principle of the search algorithm is the "*survival of the fittest*" (DAWID, 1999). That is, strategies which are regarded as superior based on an evaluation criterion are more likely to be selected as the solution to a problem.

The origin of the development of genetic algorithms is linked to the pioneering work of John Holland and colleagues (HOLLAND, 1975). However, the idea of adapting biological principles such as selection and mutation as solution techniques can also be found in RECHENBERG (1973) and SCHWEFEL (1977) or FOGEL (1963) and FOGEL et al. (1966).

To classify GAs, we can distinguish different groups of optimization methods. First, there are direct and numerical methods. The latter can be subdivided into enumerative and random search methods. In contrast to direct, calculus-based optimization methods, numerical methods do not rely on derivatives or the existence of closed form solutions. Hence, these methods are useful if functions are not twice continuously differentiable or if closed form solutions do not exist or are hardly obtainable. The disadvantage of such methods, however, is that a solution is always an approximation with the corresponding stochastic error. While enumerative methods evaluate the objective function at each point of the solution space, random search only works on a sample of it. The GA belongs to the group of random search methods. These are (intuitively) based on probability ideas as the law of large numbers and the central limit theorem (JUDD, 1998). Accordingly, the structural error regarding the random number that represents the outcome of the method can be influenced by the sample size. However, a GA differs from other random search methods in two important ways (GOLDBERG, 1989): the GA works on coded parameters and not with the parameters themselves and it evaluates a population of points rather than a single point at one time. These features make the GA robust and efficient in handling "*problems of far greater complexity and size than most other methods*" (JUDD, 1998, p.285).

The capability of GAs or similar concepts, summarized as evolutionary computation (EC), and their relation to evolutionary principles has caused increasing interest in these methods in recent years. EC is applied to a variety of problems in a broad range of disciplines including natural science, engineering, mathematics, and economics (FOSTER, 2001). Important contributions, particularly of GAs in economics, include the simulation of the repeated prisoners dilemma by AXELROD (1987) or the application on the cobweb model by ARIFOVIC (1994). The same model serves DAWID AND KOPEL (1998) in investigating different implementations of GAs. PRICE (1997) applied a GA on standard models of industrial organization as monopoly, Cournot, and Bertrand competition. VALLÉE AND BAŞAR (1999) as well as ALEMDAR AND SIRAKAYA (2003) investigate leader-follower competition, whereas BALMANN and HAPPE (2001) explore agricultural land markets with respect to oligopolistic behaviour. YANG (2006) incorporates GAs into a state dependent dynamic portfolio optimization system to improve the expected return estimation and portfolio efficiency, while HARUVY et al. (2006) explore proposed reforms of the judge/law clerk market in the United States.

Another important application of GAs is the simulation of learning as part of human behaviour (BIRCHENHALL, 1995; DAWID, 1999; RIECHMANN, 1999; 2001; VRIEND, 2000; ARIFOVIC AND LEDYARD, 2004; BRENNER, 2006). However, unlike these or most of the previous examples, we do not consider time within the simulation, which is an essential part of learning. Nevertheless, as we will see in the following sections, to find the optimal solution of a static problem, the GA consists of a dynamic procedure. In this regard, we can interpret the GA as iterative search for Nash equilibria, which can be represented as a Markov process (REEVES AND ROWE, 2003, pp. 112). A familiar example of iterated algorithms is the Cournot model, where the repeated, alternate evaluation of the players' reaction functions (given the strategy of the competitor) finally leads to the Nash-equilibrium outcome.

The interpretation of learning in terms of an iterated process as search for on optimal solution of a (static) problem (BRENNER, 2006) provides an important link between GAs and evolutionary game theory. RIECHMANN (2001) shows that GA learning is a specific form of an evolutionary game. The simple canonical GA as presented in Section 4.2 converges towards a Nash equilibrium in terms of an evolutionary stable state.

Although we consider a static model, the method is not applied to a pure optimization problem. In fact, most of the applications in social science go beyond optimization because strategic interactions are involved. In this respect, SON AND BALDICK (2004) demonstrate that the canonical GA can misidentify Nash equilibria by following a local optimization path. In this case, the GA cannot differentiate between local optima and "real" Nash equilibria. Following PRICE (1997), SON AND BALDICK (2004) show that a co-evolutionary approach overcomes this problem. We introduce the structure and functioning of a standard GA for a traditional optimization problem, before we discuss the implementation of the co-evolutionary GA to determine the optimal strategy of processing firms within the spatial simulation and with strategic interactions involved.

4.2 GAs for Optimization

Comprehensive introductions to the method are, e.g., GOLDBERG (1989) or MITCHELL (1996). The more recent book of REEVES AND ROWE (2003) also entails a detailed discussion regarding different theoretical perspectives of GAs. For the presentation of the method based on industrial organization models, the paper of PRICE (1997) is recommended.

Although there is no rigorous definition of a GA, at least three attributes are considered as standard: a population, genetic operators, and a fitness function. A collection of candidate solutions to a particular problem is called the population. The fitness function is the representation of the underlying (exogenous) objective function and is used to evaluate the quality of a solution. Finally, genetic operators as selection, crossover, and mutation can be applied to the population to identify good solutions as well as incorporate and consider new (probably superior) solutions.

We may want to apply a GA to a simple optimization problem over a search space Σ:

(2) $h : \Sigma \rightarrow \Re$

to maximize the objective function h with respect to a vector of decision variables $\sigma \in \Sigma$. However, the GA does not work on the decision variables themselves. Instead, the variables are coded according to a certain alphabet A through a coding function $\kappa : A^l \rightarrow \Sigma$. The size of the coded search space is determined by l. A problem related information of σ is called the phenotype, while the coded representation $a \in A$ is the genotype of a solution. Because a solution may consist of a number of decision variables, we distinguish genes and chromosomes. Genes are the coded representation of one decision variable; the chaining of those forms the chromosome. For instance, the chromosome can be regarded as the coded strategy of a game. One possibility of genotype-phenotype mapping is to use binary coding. Thereby, the candidate solutions are represented by binary numbers as follows:

phenotype (σ)	κ	genotype (a)
5	$0^3 + 2^2 + 0^1 + 2^0$	0 1 0 1
11	$2^3 + 0^2 + 2^1 + 2^0$	1 0 1 1

In the example, the strategy consists of one decision variable which is coded as a bitstring of length $l = 4$. Of course, it is clear that the search space must not exceed $\Sigma = \{0,1,\ldots,15\}$. Hence, the genotype-phenotype mapping depends on the problem at hand.

For a larger search space, we may use $l > 4$ or even other coding schemes. However, regardless of the particular coding, the representation of a collection of candidate solutions builds the initial population and is the first step to initialize a GA. Commonly, the initial population consists of a predefined number of random candidate solutions. The size of the population depends on the problem in general and the size of the search space in particular. Basically, a large population size increases the probability of good solutions within the initial population but may decrease the speed of computation.

The second step of the GA is to evaluate the objective function (2) at all points σ represented by the chromosomes of the population. It is convenient to use monotonic transformation of the objective function to ensure a positive measure of the solution's quality. For instance, non-negative values are required for the selection operator as described below. The transformation is called the fitness function and the quality of the solution is the fitness of the chromosome. We can determine a non-negative fitness value with:

(3) $f : A^l \rightarrow \Re^+$

Accordingly, if $h(\sigma)$ yields a negative value for the real world problem and $\kappa(a) = \sigma$, $f(a)$ is the positive fitness value of the chromosome based on the monotonic transformation (3).

Once the fitness of all chromosomes is determined, the first genetic operator can be applied. Although there are different options to implement selection, the common feature is selection proportional to the fitness value of a chromosome. The higher the fitness of a solution, the higher the probability to be selected. A common approach for fitness proportional selection is roulette wheel selection (RWS). Thereby, the total fitness of a population is represented by a roulette wheel. Each chromosome has a slot on the wheel that is sized proportional to its fitness value. If we spin the wheel, the probability of selection is proportional to the area of a sector on the wheel. For instance, the sector is twice as large as a second sector on the wheel if the fitness of the corresponding solution is twice the fitness of the second solution. However, RWS is stochastic and there is no guarantee that the best solution is selected. Another approach is to select a particular number of the fittest chromosomes of the population. This is usually called best chromosome selection (BCS). The population of chromosomes is ordered according to the fitness value of the solutions, e.g., in ascending order. The selection of a predefined number of fittest chromosomes ensures that the best solution of the population is always selected. Additionally, the selection pressure can be influenced by the number of chromosomes (the share of the population) which are considered for selection.

In either case, selection creates a new population consisting of chromosomes from the initial population but in a different composition. Due to its nature, selection decreases the variability of the population. The initial population consists of random candidate solutions, so it is unlikely that a good or even close to optimal solution is contained. In this respect, the concepts of mutation and crossover are important. Mutation is a random manipulation of a solution, while crossover recombines the information of two parent chromosomes which yields two offspring. To illustrate both, we use the previous example. Mutation randomly alters the genetic information of a chromosome. For instance, one bit at position three of the string (the genotype) is flipped, which also causes a change in the phenotype.

	σ	a	a'	σ'
mutation:	5	$0\,1\,\underline{0}\,1$	$0\,1\,\underline{1}\,1$	7
crossover:	13	$\underline{1\,1\,0\,1}$	$\underline{1\,1}\,1\,1$	15
	11	$1\,0\,1\,1$	$1\,0\,\underline{0\,1}$	9

Crossover refers to an exchange of information between two parent chromosomes to create two offspring. Thereby, the genetic subsequences before and after a randomly chosen locus are traded. The example above shows that the trade of the first and last two bits of the chromosomes changes the phenotype from 13 to 15 and from 11 to 9, respectively.

While each of the presented parts of a GA fulfils important tasks, the power of the method rests in the interaction of the single components. By following MITCHELL (1996, p.10), we summarise a simple GA by the following steps:

1. Generate the initial population of *x* random candidate solution.
2. Calculate the fitness of all chromosomes of the current population.
3. Repeat the following steps *x* times to create a new population:
   - i Select two chromosomes from the current population. The probability of selection is an increasing function of the chromosome's fitness.
   - ii Execute crossover with given probability, the crossover rate, at the random locus (crossover point) of the two chromosomes. The crossover point is determined according a uniform probability distribution. The result of the crossover are two offspring. If no crossover takes place, the offspring are exact copies of the parents.
   - iii Mutate the offspring at each locus with a given probability, the mutation rate, and place the chromosomes in the new population.
4. Replace the current population with the new population.
5. Go to step 2.

The succession of fitness evaluation and application of genetic operators (selection, mutation, and crossover) is called a generation. The set of all generations is a run. Within a run, the number of generations is limited by a termination criterion. Common approaches are, e.g., a predefined number of generations or a certain threshold with respect to the population's diversity. While the first criterion ensures that the optimization always stops at the same iteration, the latter terminates the run if, e.g., some share of the population consists of the same chromosomes or the variance of the chromosomes' phenotypes is low and constant over a particular number of generations.

During the run, i.e., the succession of generations, selection tends to cause good solutions to survive. In this respect, we can refer to selection as the decision to keep a particular chromosome in the GA's population and thus in the process of optimization. However, the variability of the population decreases. In this respect, selection is the operator that drives the population's convergence toward the best solution of the population.

The importance of mutation and crossover rests in two tasks. First, they enable the algorithm to increase the precision of the method (over generations) by approaching (local) optima. New solutions, created by mutation or crossover, are favoured over the fittest chromosome of the population if they are closer to an optima. Consequently, such new solutions are established in the population through selection. Second, the more important feature is that mutation and crossover avoid the lock-in of the optimization in local optima. Because both operators create new solutions, the GA is able to cover the search space effectively during a run. Based on the chosen coding scheme, these operators can introduce solutions which are significantly different from previous solutions. In this way, the search space is extended during the optimization process, which also decreases the dependence regarding the initial population, i.e., the initial conditions of the optimization.

4.3 Co-evolutionary GAs

The previous structure of a GA is only applicable if the fitness of a chromosome exclusively depends on its bit values. This is, e.g., not the case in most economic systems in general or in a normal form, multi-player game in particular. Instead, the payoff of a player's strategy depends on the strategies of the other players. This is particularly true under spatial competition due to neighbourhood interactions. Therefore, we use a so called co-evolutionary (or parallel) GA model. Each player exhibits an individual population of strategies, i.e., in the simplest case of two players there are two populations as well.

The basic features of the GA from the previous section, such as coding and the genetic operators, are not affected. The major difference is the fitness evaluation. In Section 4.2, we simply translated the genetic information of a chromosome into the phenotype and used this information to evaluate the (one-dimensional) objective function. While the chromosome in Section 4.2 could also be a multi-dimensional vector of decision variables, we must deal with a *n*-dimensional objective function in a co-evolutionary framework. Thus, we can rewrite (2) to:

(4) $h: \Sigma \rightarrow \Re^n.$

A common approach to consider (strategic) interactions within the players' fitness evaluation is to implement a tournament (PRICE, 1997; BALMANN AND HAPPE, 2001). Random strategies are chosen from each player's population and the vectors of these strategies are evaluated. The outcome is an individual fitness value for each player, which is assigned to the corresponding strategy of a player. The process is repeated until a sufficient number of observations are available to assign values to all strategies of all players. The number of iterations or rounds within the tournament depends on the search space as well as on the number of players and is increasing in both. However, the outcome of the tournament is an average fitness value for each of the players' strategies. Accordingly, we can alter the previously described implementation of the GA only by including the tournament in step 2 of the GA's schedule:

1. Start with a number of initial populations equating the number of players.
2. Repeat the following steps over a sufficient number of iterations:
   - i Randomly select a single strategy of each player's population.
   - ii Play the strategies against each other and determine the individual fitness value subject to the other players' strategies.
   - iii Assign the fitness value to the corresponding strategy.
   - iv Go to 2i.
3. Determine the average fitness of each strategy based on the values realized in 2iii and apply the genetic operators: selection, crossover and mutation (similar to step 3 in the previous GA example on page 14) to create the new population.
4. Replace the current population with the new population.
5. Go to step 2.

The above scheme is the core of the GA implementation within SpAbCoM as described in Section 6.2. Two particular examples, which illustrate how a co-evolutionary (two-population) GA works are also presented in GRAUBNER et al. (2011a) and GRAUBNER et al. (2011b).

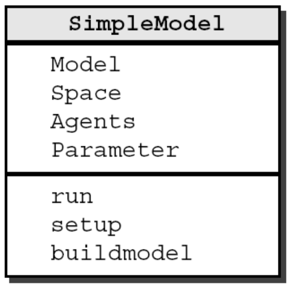
5 THE REPAST MODEL

Based on the conceptual considerations we implement the simulation with the ABM software Repast. Particularly, Repast's "*Simple Model*" serves as the point of departure. This model is a single class, and simplified it consists of: the simulation schedule, the space, and the agents therein. To illustrate the structure of an application based on OOP, one can use a standardized tool as the unified modelling language (UML) (BOOCH et al., 2005). For instance, UML class diagrams highlight the (class) structure of a software program, how parts of it are related, and the components of the different classes. Figure 2 shows a UML representation of Repast's `SimpleModel` class. The first (upper) part of any UML class is the class name (bold). The second part represents the attributes (starting with capital letter), while the third part contains operations or methods (lowercase letter). Accordingly, the implementation shown in Figure 2 features four attributes and three methods.

The attribute `Model` provides the graphical user interface of the Repast model. `Space` characterizes the artificial region and `Agents` is a list of objects that can be located in that region. Predefined parameters as the number of agents, the size of the region, or others are stored in `Parameter`.

The methods of the class are `setup`, `buildmodel`, and `run`. The first method initializes the Repast model with the defined parameters and opens the Repast interface. The `buildmodel` method is required to locate agents into the space and visualize the artificial region. While the previous methods are commonly used only once at the beginning of a simulation, `run` is responsible for the dynamics of the simulation. The agents interact within this method, i.e., actions are executed by agents who may change their and other agents' attributes. A simple example is the location of agents within the region and the distance between them. Location and distance to other agents may be the agents' attributes and there is a method move specified in `run`. If agents change their location while `run`, this clearly affects their location coordinates. Hence, their attributes are changed. Additionally, this may affect the distance among agents and thus influences the attributes of other agents.

Figure 2: An abstraction of the "Simple Model" class in Repast



The simple example already shows how Repast's `SimpleModel` is extended. Basically, the attributes and methods in this model are empty. To run a simulation, we must fill them. Therefore, we implement characteristics and behaviour rules derived from theoretical considerations regarding spatial competition.

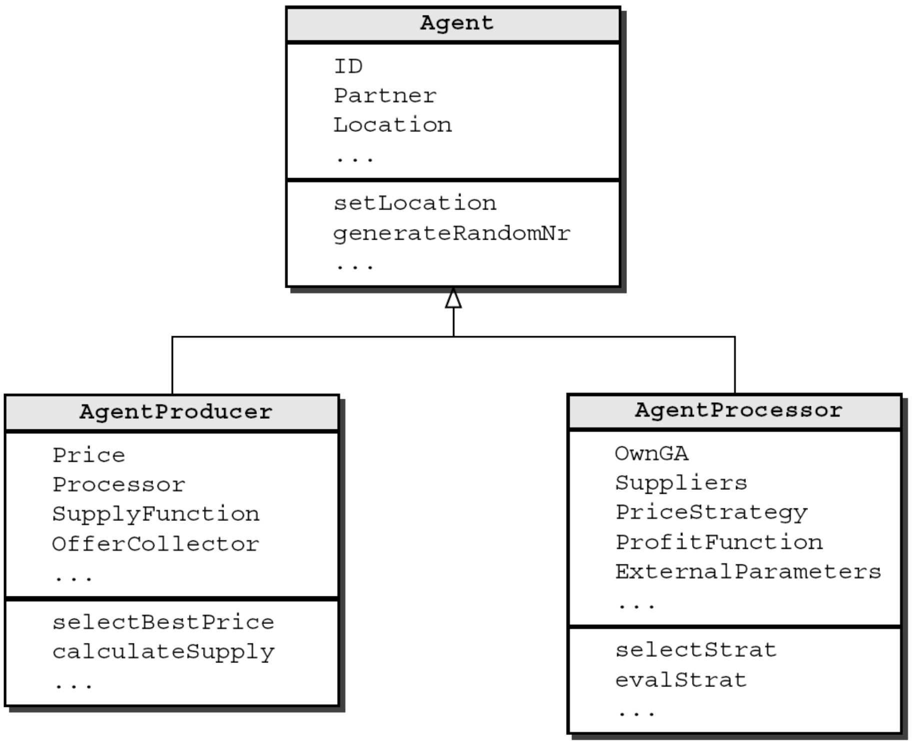
6 THE STATIC STRUCTURE OF SPABCOM

The aim of this section is to present the structure of the simulation model. We introduce the most important parts as the agents' hierarchy followed by the behaviour rules on which the agents' decisions are based. In the third part, additional components of the simulation are presented to give a simplified overview of the whole structure. We use UML class diagrams as a technical device to visualize the structure of the simulation based on the OOP approach.

6.1 Agents

OOP was introduced at the beginning of this paper as a central concept regarding the implementation of the spatial competition model. The classification of the agents is a good example to illustrate this. Not surprisingly agents are the core component of an agent-based model. According to our conceptual framework we can distinguish two types of agents: producers and processors. Since both types feature common characteristics and there may be a large number of, e.g., producers, OOP provides an efficient way of modelling. Figure 3 shows the simplified agents' class hierarchy of the simulation model. For clarity reasons, the picture shows only selected attributes and methods of the corresponding classes. In addition to the representation of classes and their components, UML provides the possibility to illustrate and specify the relationship among classes. For instance, *associations* are depicted by lines. The type of an association is specified, e.g., by different arrowheads. In Figure 3, the empty arrow from the two lower classes to the upper class denotes *inheritance*. That is, `AgentProducer` and `AgentProcessor` are subclasses of `Agent`. Particularly, the two derived classes feature the same characteristics as the base class plus specific attributes or methods.

Figure 3: Class structure of the agents



Common features of all agents are a unique identification key (`ID`), the location in space (as *x*-*y* coordinates), and a list of other agents (`Partner`). Methods derived from the `Agent`-class control, e.g., the location (set or change) within the region or a generator for (pseudo) random numbers. The field `Partner`, e.g., allows assignment of suppliers to a particular processor or a purchaser to a particular producer. Most of the features of an object of type `Agent` are model intern rather than specific for the spatial competition problem. For instance, the agent is required to be placed into the artificial region or the clear identification of an object within the simulation must be enabled. Only instances of the two derived classes `AgentProducer` and `AgentProcessor` are actually generated during a run of SpAbCoM.

Both types of agents have specific characteristics. Besides the attributes and methods from `Agent`, an instance of the class `AgentProducer` particularly features four attributes and two methods. The received price (`Price`) represents the highest price at the producer's location. Because producers are price takers and there is potentially more than one processor that sets a price at this location, the supplier needs to determine the optimal (highest) available price. This is implemented by `selectBestPrice`, which also can set or change `Price` and `Processor`, depending on the supplier's decision. The latter attribute identifies the destination of the producer's supply. The quantity to produce, in turn, is determined by `Price` and the producer's individual supply function (`SupplyFunction`).[2] In fact, the supply of the producers is an important piece of information for the processor's decision making, particularly regarding the evaluation of their strategies.

Processors face strategic interactions caused by local market power, so the identification of optimal strategies is a nontrivial problem. Therefore, an object of `AgentProcessor` exhibits a powerful optimization tool in the form of a genetic algorithm (`OwnGA`). The concept of the underlying GA is discussed in detail in Section 4.3 and its implementation is presented in the next section. However, the major task of the field `OwnGA` is to provide the methods to identify optimal strategies in terms of the spatial competition game. One feature of the GA is that there is one best strategy in each generation and this current (optimal) strategy is represented by `Price-Strategy`.

In addition to the attribute Partner of the base class, there is the class specific field Suppliers. Both are used to determine and distinguish the desirable and actual market allocation from the processor's point of view. The differentiation between the two lists of producers rests in the fact that the set of addressees of a price offer (Partner) and the set of actual suppliers (`Suppliers`) do not have be equal.[3]

`ProfitFunction` is used to evaluate a price strategy. It represents the fitness function of the GA and is explained in more detail in the corresponding subsection of Section 6.2. A further attribute of the processors consists of external conditions as the price on the output market (for the processed good) or the transport rate, which are summarized in the field `ExternalParameters`.[4]

6.2 Behaviour Rules

While the agent's class hierarchy is one important part of the simulation's structure, the agents' behaviour rules are another crucial component of SpAbCoM. In our framework, a behaviour rule is the implementation of the agents' objective functions. On the one hand, selecting the optimal (highest available) price is the behaviour rule of the producers. On the other hand, the behaviour rule of the processors is to maximize profit by optimizing the spatial competition strategy as written in (1). A GA is integrated into the simulation to fulfil this complex task.

[2] At this stage, ABM also provides the opportunity to differentiate producers with respect to their production functions. Although there might be interesting applications for this framework, we assume a uniform supply function over the set of producers.

[3] This differentiation of actual and potential suppliers is also depicted in Figure 6 and discussed in Section 7.2.

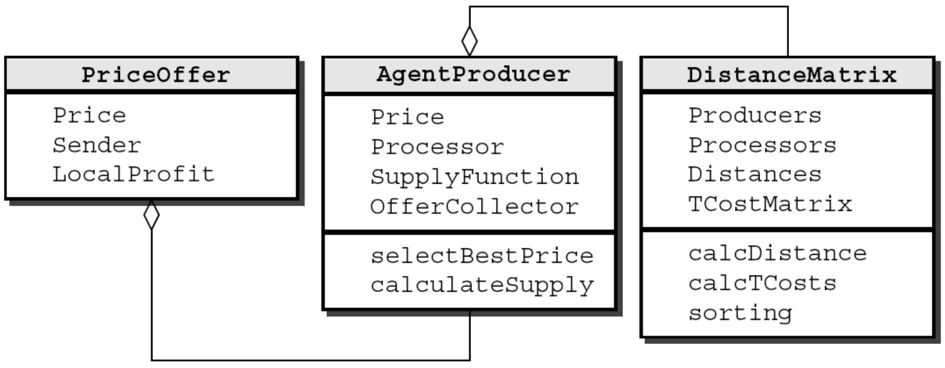
[4] In the case of the processor, it would be feasible to extend the simulation with respect to different marketing possibilities or transport technologies. In this way, further research could differentiate, e.g., processing firms with respect to the received output prices or their cost structure. However, throughout this work we assume that these factors are identical for all processors.

Although the price maximization rule of the producers is rather simple, there are cases in which we must consider the spatial nature of the problem. In particular, the decision of the producers may not only consist of the selection of the highest price from a list. Instead, we may want to incorporate the possibility of arbitrage. In this respect, an agent of the type `AgentProducer` may find it profitable to sell its product at a point different from the own location. Arbitrage is feasible if the price at that point less transport costs, is higher than the price at the location of the producer. The components, required for their decision, in terms of the class structure, are presented in the following section before we discuss the implementation of the processor's decision making via a GA.

Price maximization of the `AgentProducer`'s class

Figure 4 illustrates the relation of the three crucial classes that are used to determine the optimal price for the producers. Associations with an empty diamond as from `PriceOffer` to `AgentProducer` and from `AgentProducer` to `DistanceMatrix` denote aggregations. For instance, an object of the type `DistanceMatrix` contains a set of objects of the class `AgentProducer`.

Figure 4: Classes associated to `AgentProducer` to determine the optimal price



The producer collects the price offers from processors in a list, `OfferCollector`, to determine the supply by its supply function. The general form of this function is $q = w^{\varepsilon}$ with q being the supply, w the received price, and ε the supply elasticity. Basically, a price offer contains three pieces of information: the price at the location of the producer (`Price`), who set that price (`Processor`), and what is the processor's local profit per unit supply at this location (`LocalProfit`). Based on the price information, a producer can sort `OfferCollector` to select the highest offer by `selectBestPrice`.[5]

The `DistanceMatrix` class is required to incorporate the possibility of arbitrage by suppliers. Note in this framework arbitrage denotes a positive price difference between two locations that is larger than the transport costs between these locations. In this case, it is profitable for the producer to sell the input at the location with the higher price. For each instance of the class `AgentProducer`, `DistanceMatrix` provides data and methods to determine whether or not this is true. In particular, there are two lists, `Producers` and `Processors`, that comprise the agents of the region. `Distances` and `TCostMatrix` are fields in form of matrices consisting of the distances between all producers as well as the

[5] At this point it is easily feasible to incorporate some degree of bounded rationality. For instance, suppliers may be not able to distinguish marginally different prices. In this case, we could assume that $w_1 = w_2$ if $|w_1 - w_2| \leq \omega$ and ω is a certain threshold so that the producer is indifferent between the corresponding processors. Even though we do not consider bounded rationality in any part of this work, the possibility to do so is a further advantage of agent-based modelling.

producers and processors of the region and the corresponding transport costs. The methods of `DistanceMatrix` allow setting and changing of the values of the matrices or, as in the case of sorting, comparing these values. In this way, each producer can access the distance and transport costs to all other producers to compare their own price with the prices at other locations, considering the transport costs. We can summarize the decision rule or objective function of each producer by:

(5) $q = \max\{w, w' - TC\}.$

Accordingly, a producer selects the maximum price at its location or any other location subject to the transport costs *TC* to that location.

Strategy optimization of the* `AgentProcessor`*'s class

This section presents the particular implementation of the GA within the structure of the simulation. The Repast distribution that was used, contains the Java Genetic Algorithm Package (JGAP) which provides the basics of a simple GA for optimization (MEFFERT et al.). In the following section, the adaptation of the GA to strategic interactions under spatial competition is presented. Similar to the producer's decision making process, processors, i.e., objects of `AgentProcessor`, use instances of different classes to determine the optimal strategy. Important classes, their features, and relations are illustrated in Figure 5.

The structure is derived from the concept of a co-evolutionary GA as discussed in Section 4.3. Accordingly, each processor exhibits an individual GA. In Figure 5, a line with the diamond at its end, an aggregation, illustrates this relation. For instance, the aggregation from `GenomP` to `AgentProcessor` denotes that the first is a part of the latter. Furthermore, `ProfitFunct` represents the fitness function of the processor, and is also part of `AgentProcessor`. The two methods of this class allow decoding of the genetic information of a chromosome through the genotype-phenotype mapping (`decode`) and evaluation of the fitness of the corresponding strategy (`evaluate`).

The major parts of the individual GA are summarized by an instance of the `GenomP`, which class exhibits a collection of chromosomes (`Population`) and defines crucial parameters of the GA optimization as the population size (`Popsize`) and the number of generations (`Generations`). Other parameters are represented by the object `OwnConfiguration`, which is stored in the field Configuration of `GenomP`. Here, the type of selection (`NaturalSelector`), the collection of genetic operators (`GeneticOperators`), and an algorithm to generate (pseudo) random numbers (`RandomGenerator`) are set.[6] Even though it is feasible to define the different parameters of the GA within the individual configuration of each agent, we use global values. That is, these parameters as the mutation or crossover rate are uniform over the set of producers.[7] Table 1 summarizes the parameter setting of a processor's GA.

[6] The latter is necessary because a number of processes require stochastic initialization as the selection of the initial population or the determination of the chromosome's loci for crossover or mutation.

[7] Nevertheless, the individual GA of each processor may represent one of the most interesting options to extend the simulation. For instance, consider the German raw milk market. Here, we find a mixed market structure where processors are organized either as cooperatives or investor-owned firms. This may have an impact on the behaviour rule. Due to the individualized decision making of the agents within the simulation, it would be feasible to incorporate such heterogeneity.

Table 1: Specification of the GA parameters

Parameter	Value
Population size	25
Number of generations	2,500
Genes/chromosome	4
Type of selection	BCS*
Mutation rate	0.02-0.10
Crossover rate	0.05-0.20

* BCS = best chromosome selector

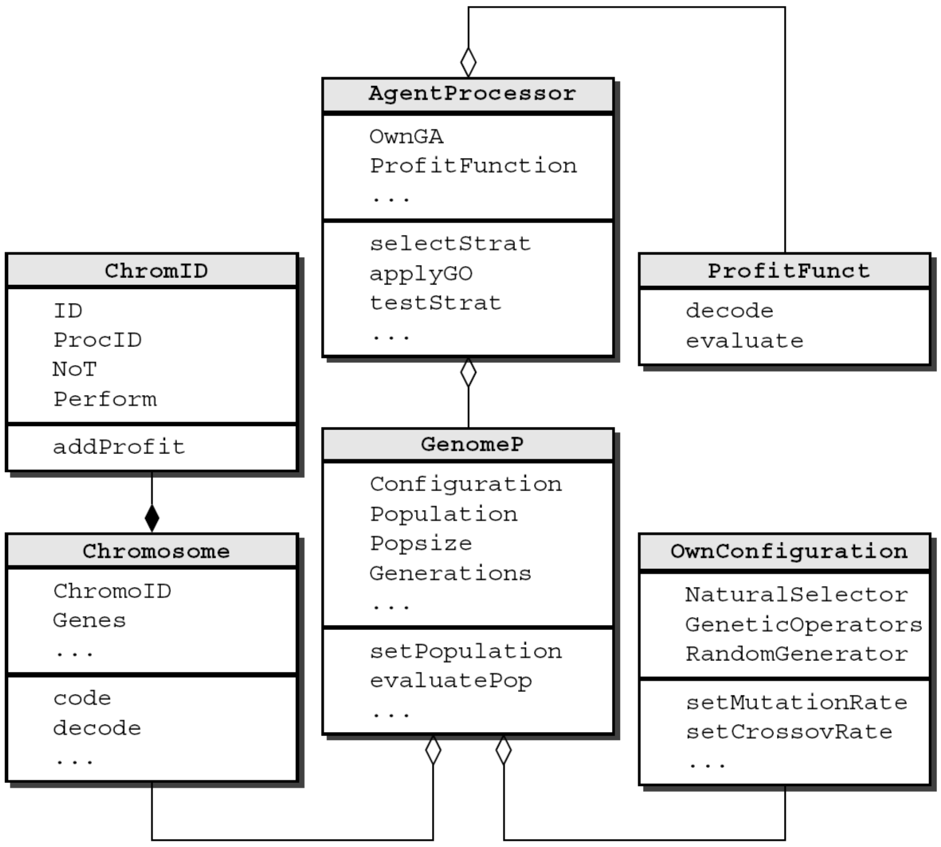
The values of the parameters are earned by test simulations and from common values in literature. The selection scheme of best chromosome selection (BCS) is chosen based on three considerations. The first is the observation that BCS yields a higher adjustment to the optimal value of spatial optimization problems during test simulations compared to RWS. The second point considers the extension of the search space. RWS does not guarantee the selection of the best strategy due to its stochastic nature. Otherwise, it is feasible to use BCS to select a certain share of the original population and add random strategies to yield the original size of the population. This extends the search space at a higher rate than only mutation or crossover during simulation. At the same time, the best strategies remain in the population. The third point considers the fitness differential of strategies and its impact on selection.

Because we base our analysis mostly on the fittest chromosome and not on the average fitness of the population, the disadvantage of BCS, neglecting the fitness differential among the best strategies, is not effective. Basically, in test simulations we observe a higher volatility of the fittest solution under RWS, while the variety of the population is higher under BCS. If we assume that a higher variability in the population decreases the probability of look-in of the GA, this may be an advantage of the BCS.[8]

The central component of the `GenomP` class is the population of candidate solutions. Such a processor's individual strategy pool, Population, is a list of randomly initialized chromosomes. Solutions of the spatial competition game can consist of four decision variables (regarding the location and price policy). In this case, a strategy is represented by chromosomes of four genes. The JGAP library provides a model chromosome class which we use for the implementation of a processor's strategy (`Chromosome`). The genes of a chromosome code the four decision variables: mill price (m), price discrimination (in terms of α), and location coordinates (x and y). The two methods of the `Chromosome` class are used for genotype-phenotype mapping. Additionally, each chromosome exhibits a `ChromID` object. In Figure 5, the filled diamond between `Chromosome` and `ChromID` denotes a stronger form of aggregation, namely a composition. This specifies that the whole, the chromosome, and its part (`ChromID`) cannot be separated in terms of their lifetimes, i.e., the part must be initiated at the moment the chromosome is created.

[8] In general, the choice of the selection operator is always arbitrary to some degree. Advantages or disadvantages can be compensated for by specific adjustments to the underlying problems. For example, under RWS it is feasible to always keep the fittest chromosome in the population. However, based on the programming of the selection operators within the JGAP package, BCS yields superior results for the spatial competition model.

Figure 5: The class structure of the processors decision making



The `ChromID` object does not only identify the particular strategy (with the field `ID`) but also the corresponding processor (`ProcID`) holding the strategy in its population. In addition to the identification, an instance of `ChromID` performs important tasks in the framework of the co-evolutionary GA concept. As presented in Section 4.3, the core of the parallel GA is a tournament. A `ChromID` object collects the information during the test rounds. The method `addProfit` assigns the outcome of a test, the current fitness value, to a list (`Perform`) whose number of entries equals the number of tests (`NoT`). Based on this list, the average fitness value of the strategy is determined by the method `evaluate` of `ProfitFunct`. Because there is a certain probability to test a worse strategy against an even worse strategy of the competitor, we use the median instead of the mean. In this way, we avoid overestimating the influence of outsiders.

The interplay of the single parts of the processors' decision making are controlled by the methods of the `AgentProcessor` class. In Section 6.1, we briefly introduced the two methods `selectStrat` and `evalStrat`. The latter is not depicted in Figure 5 because we can decompose the method in `applyGO` and `testStrat`. If we reconsider the schedule of a co-evolutionary GA (as described on page 15) we can classify the task of the methods. Based on a selected strategy by `selectStrat`, `testStrat` is responsible for executing the tournament within the GA. During this and potentially other test rounds fitness values are assigned to the strategy and yield an average fitness. Finally, `applyGO` is responsible for applying the genetic operators: selection, crossover, and mutation.

The structure as illustrated in Figure 5 shows the components required for the processor's decision making. The outcome of this process is a price offer that in turn is available to suppliers. However, neither the process nor the outcome is pictured. Basically, the interaction of both, the producers' and processors' decisions, yields the market allocation and finally the optimal strategy of the processors. Before we present this process we summarize the static structure of SpAbCoM by uniting the introduced parts and add some additional important components.

6.3 Summarizing Overview

We can summarize the structure of the model with a simplified UML class diagram. For clarity reasons, Figure 6 shows only the class names and the most important relations between the objects. We recognize the agents' hierarchy (although in slightly different order) in the centre of the picture as well as the structure of the decision making of both the processors on the right side and the producers on the lower left side, respectively. Additional classes are required to manage the run of a simulation. For instance, the object `LocateAgents` is responsible for the initial location of the agents at the start of a simulation. Throughout this work we use a uniform distribution of suppliers, i.e., exactly one producer occupies a cell of the grid representing the artificial region. Processors, however, might relocate during the simulation. The location of the processors at the beginning of the simulation is either predefined or randomly initialized.[9]

With respect to additional relations between classes we recognize the labelled aggregations between `AgentProducer` and `AgentProcessor`. The two relations take into account the decisive differentiation of potential and actual suppliers. The difference within the simulation model is implemented by two lists containing the actual and potential suppliers of each processor.

The major component of the simulation model not yet discussed is the `Market` class. One instance of this class is created for each simulation. `MainSim`, which only contains the `main` method as required to start any computer program, is responsible for that. All of the parameters that are necessary for the simulation as well as the sequence of actions are defined in `Market`. In fact, `Market` is the major object to control the execution of the simulation. Important variables that are set and can be changed between simulations are listed in Table 2.

The table highlights the parameters directly related to a spatial competition problem. Because the individual variables are introduced and defined in the single applications (cf. GRAUBNER et al., 2011a,b) we do not explain them again here. Except for the price of the finished good (the marginal revenue product of the processors), different values of the variables (within the listed domains or states) are used to conduct simulations. For instance, in GRAUBNER et al. (2011a) a line market of size (x, y) = (300, 1) serves to identify the optimal spatial price strategy in a duopsonistic framework. While the number of processors as well as the number of suppliers was constant (2 and 300, respectively), we conducted simulations over the relevant range of transport costs, namely a transport rate of $0 \leq t \leq 2$.

Once these parameters are defined and initialized by the `Market` object, the actual simulation process starts. The dynamics of this procedure are described in the following section.

[9] In fact, in the case of a game including the location decision of firms, the initial location is just a illustrative device at the start of each simulation. The location variables are immediately overwritten once the processor selects a strategy.

Figure 6: The static class structure of SpAbCoM

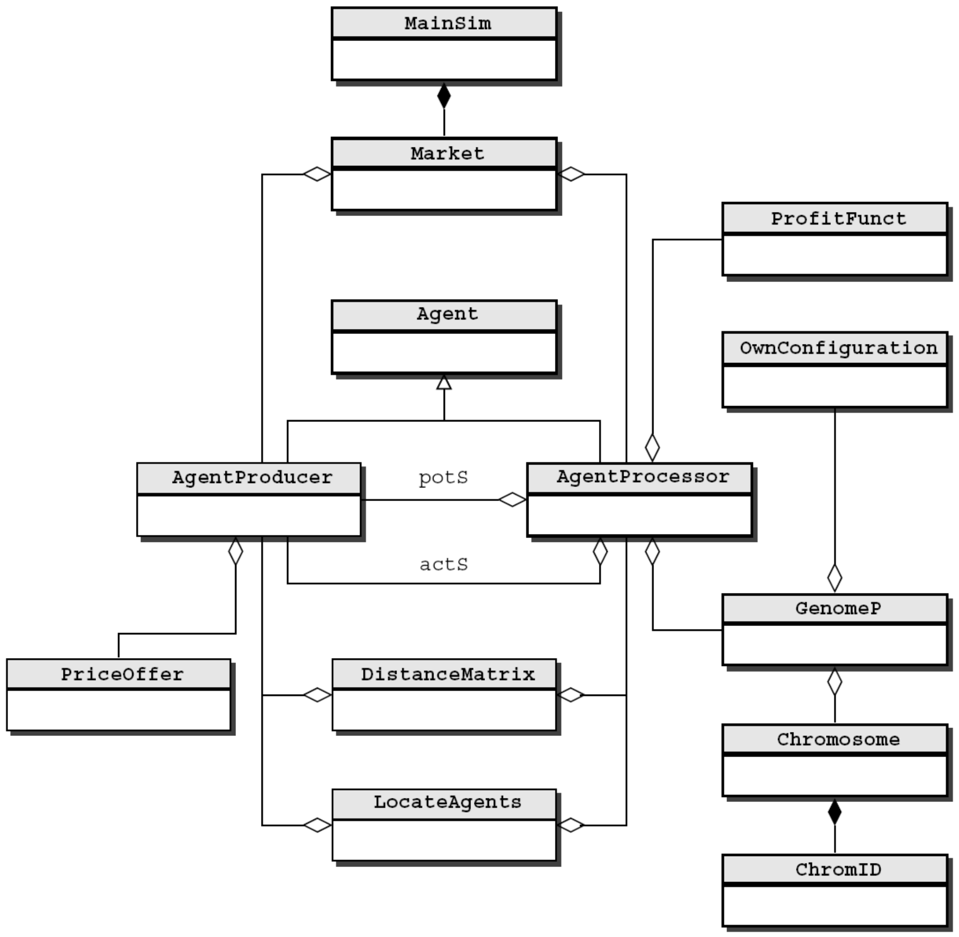


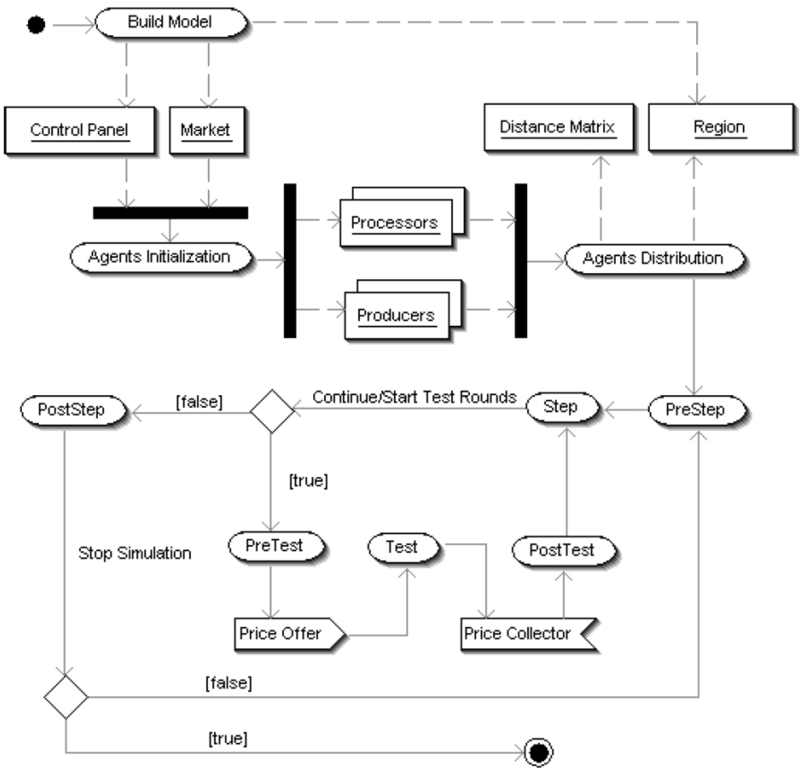
Table 2: Selected parameters of the simulation model

Parameter	Value
Size of the region	(x, y)= ({300, 20}, {1, 20})
Number of suppliers	{300, 400}
Number of suppliers	{2, … , 6}
Transport rate	[0, 5]
Supply elasticity	[0, 1]
Arbitrage of suppliers	{true, false}
Relocation of processors	{true, false}
Two-dimensional space	{true, false}
Border effects	{true, false}

7 Dynamics of the Simulation

The sequences of actions within the simulation can be decomposed in an initialization and running phase. The first is responsible for creating the simulation's environment: the space, agents, and some control objects. The running phase consists of three steps which can be further subdivided. Figure 7 shows a simplified scheme of the simulation in terms of a UML activity diagram. A single run of the simulation starts at the black point in the upper left corner of the picture. The upper part of it represents the initialization of the simulation. The running phase of a simulation starts with the `PreStep` activity. The difference between the two levels is that each simulation has only one initialization but multiple running phases. The number of loops, composed of `PreStep`, `Step`, and `PostStep`, corresponds to the number of generations of the GA.

Figure 7: Activity diagram of the simulation's schedule

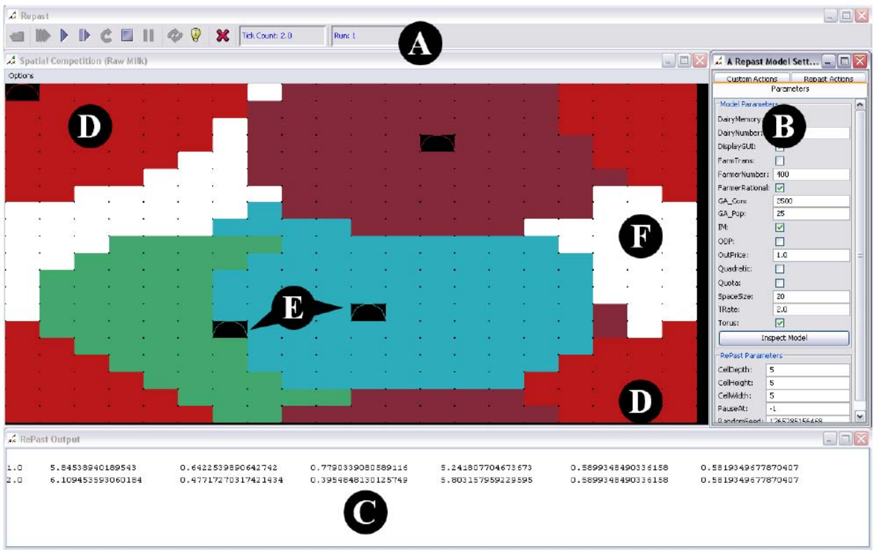


7.1 Initialization of the Simulation

We describe the dynamics of the simulation in more detail, starting from the upper left corner of Figure 7. First, three objects are initialized by the `main` method of `MainSim`: the control panel of the simulation, the market, and the artificial region. The first is provided by Repast and allows control of the simulation via a graphical interface. In this respect, a good comparison is, e.g., the Windows Media Player. As such, the control panel of Repast provides the option to load a file, start and run the program, or stop or pause it. Additionally, it is feasible to directly set the parameters from Table 2 through the interface. For an illustration, Figure 8 shows a screenshot of Repast's graphical interface while SpAbCoM is active. An important option of the control panel is to load an input file. In this way, values of one or several parameters can be defined, and a number of simulations can be performed over these values. In particular, the input file specifies the start and stop value of a particular parameter. For example, we can specify that the number of processors should vary between two and four, and the number of repetitions should be 50. Hence, there are 50 runs of the simulations for two, three, and four processors and 150 simulations in total.

As mentioned in the previous section, most of the environmental variables of the simulation are defined in the `Market` class. Either the values of the parameters depicted in Table 2 are directly defined within an instance of this class or they are set via the Repast interface. Once the parameters are set, the agents can be initialized. The number of processors is predefined by the corresponding parameter, whereas the number of producers depends on the market size. Two integers (the x and y coordinates) measure the extent of the market, and the multiplication of both yields the number of producers.

Figure 8: Screenshot of the Repast interface during an active simulation of SpAbCoM



```
DairyMemory:
DairyNumber:
DisplayGUI:
FarmTrans:       ☐
FarmerNumber:    400
FarmerRational:  ✓
GA_Gen:          2500
GA_Pop:          25
IM:              ✓
ODP:             ☐
OutPrice:        1.0
Quadratic:       ☐
Quota:           ☐
SpaceSize:       20
TRate:           2.0
Torus:           ✓

CellDepth:       5
CellHeight:      5
CellWidth:       5
PauseAt:         -1
RandomSeed:      1265285156468
```

```
1.0   5.84538940189543    0.6422539890642742   0.7790339080589116   5.241807704673673   0.5899348490336158   0.5819349677870407
2.0   6.109453593060184   0.4771727031742l434  0.3954848130125749   5.803157959229595   0.5899348490336158   0.5819349677870407
```

Note: A = control panel, B = parameter settings, C = live output, D = market region of the processor with `ID` = 0, E = other processors, F = unserved locations. The artificial region is constructed as torus and the location of the first processor (`ID` = 0) is fixed at $L_0 = (x_0, y_{\max}) = (0, 20)$.

Values are assigned to the individual features of processors and producers by initializing the agents. For instance, each producer gets a unique location, or each processor's GA is set. As one result, the processor's individual chromosome population is created.

At this stage of the initialization phase, there are two lists of agents, producers and processors. The next step is to locate the agents into the artificial region. Therefore, the already introduced `LocateAgent` class is deployed. The region is a grid of cells. The corresponding object is provided by Repast and initialized at the start of the simulation. After the agents are located, the distances and transport costs between them are calculated. The object `DistanceMatrix` provides the corresponding functions. Predefined parameters, such as the transport rate or whether the region features border effects or not, are considered. To model space without border effects, we use a torus, i.e., cells at the edges are direct neighbours of cells which are at the opposite edge. This is the case in Figure 8. To calculate the distance between two locations L_A and L_B of agents A and B, we use:

$$(6) \qquad \overline{L_A L_B} = \begin{cases} \Delta d_{A,B} & \text{if borders exist} \\ \min\left\{ \Delta d_{A,B}, \sqrt{\dfrac{\left(x - |x_A - x_B|\right)^2 + \left(y - |y_A - y_B|\right)^2}{x^2 + y^2}} \right\} & \text{otherwise,} \end{cases}$$

$$\text{where } \Delta d_{A,B} = \sqrt{\frac{(x_A - x_B)^2 + (y_A - y_B)^2}{x^2 + y^2}}.$$

A distance vector is assigned to each cell and can be assessed by the processor after a potential relocation. Once the distance matrix is built, the initialization phase is finished, and the core simulation of the spatial competition starts.

7.2 Running Phase of the Simulation

The running phase of the simulation can be subdivided into three steps: `PreStep`, `Step`, and `PostStep`. A passage through these steps corresponds to a single generation of a GA. The first step sets parameters, which were potentially changed during previous generations, back to the default values. The activities of `Step`, in turn, can be subdivided into `PreTest`, `Test`, and `PostTest`. The differentiation is caused by the co-evolutionary concept of the GA.

As noted in Section 4.3, the important feature is a tournament of strategies among agents. `Step` represents this tournament of co-evolutionary GAs. As `PreStep`, `PreTest` sets parameters potentially changed during previous test rounds back to the default values. `Test` and `PostTest` correspond to the strategy selection and evaluation, respectively. The basic concept of the tournament is to test random strategies of all the processors against each other. Each processor selects the first chromosome of its strategy list in `Test`. This chromosome represents the active strategy, and the processor's decision variables are set according to its values. The active strategy is replaced from the first to the last position of the list such that the second place chromosome is selected in the next round. Suppose there are *n* strategies available to each firm; in order to test all *n* strategies a minimum number of times, say five, we impose a total of 5*n* test rounds. To avoid testing strategies (five times) in the same combination, we randomize the order of chromosomes of a player's strategy list after *n* test rounds.

Besides the randomized pairing of the strategies, the major activity of `Test` is the determination of the set of potential suppliers, i.e., suppliers at locations where the processor earns positive local profits subject to the selected strategy. Due to the uniform distribution of the suppliers within the region, we can represent the set of potential suppliers over the market radius $\hat{R}$ as derived in GRAUBNER et al. (2011a):

$$(7) \quad \hat{R} = \min\left\{ \left.\frac{m}{t\alpha}\right|_{\alpha>0}, \left.\frac{1-m}{t(1-\alpha)}\right|_{\alpha<1} \right\}.$$

This market radius depends on the individual strategy of the processor: the mill price *m*, the degree of transport cost transfer *α*, and the distance to the suppliers dependent on the processor's location (*x*, *y*) as well as on the transport costs *t*. Accordingly, if the distance $\overline{L_A L_B}$ between processor A and producer B, as calculated by (6), is smaller than $\hat{R}$, B is an element of the set of potential suppliers.[10] Based on the selected strategy, the processor sets a local price at the locations of the potential suppliers. Hence, these producers are addressees of the processor's price offer. Whether an offer is answered with the producer's supply hinges on the decision making of the suppliers, particularly whether the offered price represents the highest price at the producer's location. The producers' decision takes place in `PostTest`.

Because it is most likely that more than one processor sets a price at a particular location, each producer exhibits an offer collection (see Figure 4). The producer first sorts its list of price offers in ascending order to determine the highest price at its location. If we allow the suppliers to take advantage of arbitrage, as described in Section 6.2, each producer compares

[10] For a more technical definition of potential suppliers see also GRAUBNER et al. (2011b).

this price with the price of other locations. If there is a positive difference from another local price considering the transport costs to this location, the supplier identifies the location with the highest difference and the corresponding price as optimal choice. Otherwise, the highest price at the own location is selected. The producer determines the local supply based on the local price and its supply function.

From Section 6.2 we know that the object `PriceOffer` has a field `LocalProfit`. If the producer selects the price from a price offer at its location, the value of `LocalProfit` is determined by this price and the local supply. However, the value of `LocalProfit` is zero if it is profitable for the producer to deliver the input to another location due to sufficiently positive price differences. In this case, it is feasible to show that the destination of the delivery is always a market border of a processor's potential market area. According to (7), this location is characterized by zero profits for the processing firm. Although the location of a price offer and the location of an accepting producer do not necessarily have to coincide, we can identify the relation between a producer and processor over `PriceOffer` if the former delivers the input to the latter without being a potential supplier.

Based on the decision of the suppliers, the processor can evaluate the performance of the chromosome, which is determined by the sum of the local profits realized by this strategy. Unlike potential suppliers, only actual suppliers contribute positive local profits. In this respect, the set of actual suppliers is an outcome of the spatial interaction among the processors driven by their price strategies. Because a producer can deliver only to one processor, the set of actual suppliers represents the market allocation.

The calculation of the profit is the last step of each test round. A new round starts with Step if the number of desired test rounds is not reached. Otherwise, there is a predefined number of profit values assigned to each processor's strategy. The median of the values represents the average performance of a strategy. Once this is determined, the GA optimization continues with the `PreTest`. The activities of this step are the application of the genetic operators as described in Section 4.3. After the selection of successful strategies but before mutation and crossover, the best strategy of each player, i.e., the chromosome with the highest average profit during the test rounds, is used to determine the market allocation. The outcome is reported through an output file. The last step of the loop is to check whether the termination rule of the GA is fulfilled, i.e., if the desired number of generations is reached. If not, the next generation of the co-evolutionary GA simulation starts with `PreStep`, or the simulation is terminated.

8 SUMMARY AND POSSIBLE EXTENSIONS

The aim of this paper is to provide a detailed description of the Spatial Agent-based Competition Model (SpAbCoM). SpAbCoM was developed to analyse spatial competition because even restrictive assumptions in theoretical models or often analytically intractable. The advantage of the presented computational framework is its flexibility to introduce relaxed conditions regarding pricing, location, and market structure. For instance, SpAbCoM is used to study firms' choices of spatial pricing policy by GRAUBNER et al. (2011a), who showed that price discrimination but no FOB pricing emerges as equilibrium strategy in the spatially differentiated duopsony. GRAUBNER et al. (2011b) extended the theoretical framework and analysed pricing and location within a two-dimensional, oligopsony model. The authors highlight that the relation of price discrimination and (spatial) differentiation among firms hinges on specific model assumptions as the number of competitors or the nature of supply or demand functions. Both studies helped to deepen our understanding of driving forces and mechanisms behind pricing and competition in spatial markets.

However, the presented simulation model does not exploit the potential of agent-based modelling in many aspects. We may extend SpAbCoM, e.g., regarding differentiated producers or processors. Producers, i.e., suppliers of the agricultural input, may feature individual production functions that differ from production functions of other producers in the market, e.g., in terms of the produced quantity or quality of the agricultural good. Conversely, processors could differ with respect to marketing possibilities or transport technologies and thus there is a heterogeneous output price or cost structure across processing firms.

It would also be feasible to incorporate some degree of bounded rationality. For instance, if suppliers are not able to distinguish marginally different local prices or processing firms are unable to determine the exact location of the marginal supplier and thus set a too large/small market area. However, the potentially most interesting option to extend the simulation rests in the co-evolutionary Genetic Algorithm. Because each processing firm exhibits an individual GA, we could differentiate the decision rule of these agents. This may be interesting in the case of mixed markets as the German raw milk market, where processors are organized either as cooperatives or investor-owned firms. This may have an impact on the (profit-maximizing) strategy used by these firms and most likely will influence the market allocation.

Further possible applications for SpAbCoM are consumer markets. We can readily relate spatial competition to a more general framework of horizontal product differentiation. While price discrimination may play a crucial role in such settings, again the analysis is hampered by a number of analytical difficulties, particularly in the case oligopoly and multi-characteristic space. The input market framework showed that SpAbCoM is able to deal with these complex market situations and thus it seems promising for the investigation of a number of research questions regarding consumer markets.

In general, all these extensions seek to introduce more realistic assumptions into a spatial competition model to relate the model to real world observations (on particular markets). Therefore, a consequent research agenda could consist of the incorporation of statistical data and/or empirical tests of hypotheses derived from the simulation model. In both cases, the presented simulation framework SpAbCoM can serve as point of departure.

REFERENCES

ALEMDAR, N., SIRAKAYA, S. (2003): On-line Computation of Stackelberg Equilibria with synchronous Parallel Genetic Algorithms. *Journal of Economic Dynamics and Control 27(8)*, pp. 1503-1515.

ARIFOVIC, J. (1994): Genetic Algorithm Learning and the Cobweb Model. *Journal of Economic Dynamics and Control 18(1)*, pp. 3-28.

ARIFOVIC, J., LEDYARD, J. (2004): Scaling up Learning Models in Public Good Games. *Journal of Public Economic Theory 6(2)*, pp. 203-238.

AXELROD, R. (1987): The Evolution of Strategies in the Iterated Prisoner's Dilemma. In DAVIS, L. (ed): *Genetic Algorithms and Simulated Annealing*. Pitman, London.

AXELROD, R. (1997): The Complexity of Cooperation. Princeton University Press, Princeton (NJ).

BALMANN, A., HAPPE, K. (2001): Applying Parallel Genetic Algorithms to Economic Problems: The case of Agricultural Land Markets. In JOHNSTON, R., SHRIVER, A. (eds.): *Microbehavior and Macroresults*. Proceedings of IIFET 2000. International Institute of Fisheries Economics and Trade, Oregon State University, Corvallis (OR).

BIRCHENHALL, C.R. (1995): Genetic Algorithms, Classifier Systems and Genetic Programming and their Use in the Models of Adaptive Behaviour and Learning. *Economic Journal 105(430)*, pp. 788-795.

BOOCH, G., RUMBAUGH, J., JACOBSON, I. (2005): The Unified Modeling Language User Guide. Addison-Wesley, Amsterdam.

BRENNER, T. (2006): Agent Learning Representation: Advice on Modelling Economic Learning. In TESFATSION, L., JUDD, K. L. (eds.): *Handbook of Computational Economics*, *Vol. 2*. Elsevier Science, Amsterdam.

COLLIER, N., HOWE, T. NORTH, M. (2003): The Transition to Repast 2.0. In Proceedings of the First Annual North American Association for Computational Social and Organizational Science Conference.

DAWID, H. (1999): Adaptive Learning by Genetic Algorithms, Springer, Berlin.

DAWID, H., KOPEL, M. (1998): On Economic Applications of the Genetic Algorithm: A Model of the Cobweb Type. *Journal of Evolutionary Economics 8(3)*, pp. 297-315.

FOGEL, D. B. (1963): Biotechnology: Concepts and Applications. Prentice Hall, Englewood Cliffs (NJ).

FOGEL, D. B., OWENS, A. J., WALSH, M. J. (1966): Artificial Intelligence through Simulated Evolution. John Wiley, New York (NY).

FOSTER, J.A. (2001): Evolutionary Computation. *Nature: Reviews Genetics 2(6)*, pp. 428-436.

GOLDBERG, D. E. (1989): Genetic Algorithm in Search, Optimization and Machine Learning. Addison-Wesley, Reading (MA).

GRAUBNER, M., BALMANN, A., SEXTON, R. J. (2011a): Spatial Price Discrimination in Agricultural Product Procurement Markets: A Computational Economics Approach. *American Journal of Agricultural Economics (forthcoming)*.

GRAUBNER, M., BALMANN, A., SEXTON, R.J. (2011b): Price Discrimination and the Location of Firms under a General Spatial Input Market Framework. Working Paper.

HARUVY, E., ROTH, A. E., ÜNVER, M. U. (2006). The Dynamics of Law Clerk Matching: An Experimental and Computational Investigation of Proposals for Reform of the Market. *Journal of Economic Dynamics and Control 30(3)*, pp. 457-486.

HOLLAND, J. H. (1975): Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (MI).

JUDD, K. L. (1998): Numerical Methods in Economics. MIT Press, Cambridge (MA).

MITCHELL, M. (1996): An Introduction to Genetic Algorithm. MIT Press, Cambridge (MA).

MEFFERT, KLAUS et al.: JGAP – Java Genetic Algorithms and Genetic Programming Package. URL: http://jgap.sf.net.

NORTH, M., COLLIER, N. VOS, J. R. (2006): Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. *ACM Transactions on Modeling and Computer Simulations 16(1)*, pp. 1-25.

PRICE, T. C. (1997): Using Co-Evolutionary Programming to Simulate Strategic Behaviour in Markets. *Journal of Evolutionary Economics 7(3)*, pp. 219-254.

RECHENBERG, I. (1973): Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Holzboog Verlag, Stuttgart.

REEVES, C.R., ROWE, J.E. (2003): Genetic Algorithms – Principles and Perspectives. Kluwer Academic Publisher, Norwell (MA).

RIECHMANN, T. (1999): Learning and Behavioral Stability – An Economic Interpretation of Genetic Algorithms. *Journal of Evolutionary Economics 9(2)*, pp. 225-242.

RIECHMANN, T. (2001): Genetic Algorithm Learning and Evolutionary Games. *Journal of Economic Dynamics and Control 25(6/7)*, pp. 1019-1037.

SALLACH, D. (2004): Repast for Oz/Mozart. M. North, Argonne (IL).

SCHWEFEL, H. P. (1977): Numerische Optimierung von Computermodellen mittels der Evolutionsstrategie. Birkhäuser Verlag, Basel.

SON, Y. S., BALDICK, R. (2004): Hybrid Coevolutionary Programming for Nash Equilibrium Search in Games with Local Optima. *IEEE Transactions on Evolutionary Computation 8(4)*, pp. 305-315.

TESFATSION, L. (2006): Agent-based Computational Economics: A Constructive Approach to Economic Theory. In TESFATSION, L., JUDD, K. L. (eds.): *Handbook of Computational Economics*, *Vol. 2*. Elsevier Science, Amsterdam.

TIROLE, J. (2003): The Theory of Industrial Organization. MIT Press, Cambridge (MA).

VALLÉE, T., BAŞAR, T. (1999): Off-line Computation of Stackelberg Solutions with the Genetic Algorithm. *Computational Economics 13(3)*, pp. 201-209.

VRIEND, N. J. (2000): An Illustration of the Essential Difference Between Individual and Social Learning, and its Consequences for Computational Analyses. *Journal of Economic Dynamics and Control 24(1)*, pp. 1-19.

YANG, X. (2006): Improving Portfolio Efficiency: A Genetic Algorithm Approach. *Computational Economics 28(1)*, pp. 1-14.