GILLARDON

Auszug Publikationen 2004

- Modern Heuristics for Finance Problems:
  A Survey of Selected Methods and Applications

Wir denken nach, um vorzudenken

Modern Heuristics for Finance Problems: A Survey of Selected Methods and Applications

Dr. Frank Schlottmann / Prof. Dr. Detlef Seese

Rachev, S. (Ed.): Handbook of Computational and Numerical Methods in Finance

Abstract

The high computational complexity of many problems in financial decision-making has prevented the development of time-efficient deterministic solution algorithms so far. At least for some of these problems, e.g., constrained portfolio selection or non-linear time series prediction problems, the results from complexity theory indicate that there is no way to avoid this problem. Due to the practical importance of these problems, we require algorithms for finding optimal or near-optimal solutions within reasonable computing time. Hence, heuristic approaches are an interesting alternative to classical approximation algorithms for such problems. Over the last years many interesting ideas for heuristic approaches were developed and tested for financial decision-making. We present an overview of the relevant methodology, and some applications that show interesting results for selected problems in finance.

1. Introduction

It is one of the goals of computational finance to develop methods and algorithms to support decision making. Unfortunately, many problems of practical or theoretical interest are too complex to be solvable exactly by a deterministic algorithm in reasonable computing time, e.g., using a method that applies a simple closed-form analytical expression. Such problems require approximation procedures which provide sufficiently good solutions while requiring less computational effort compared to an exact algorithm. Heuristic approaches are a class of algorithms which have been developed to fulfil these requirements in many problem contexts. Today, there are many different paradigms for heuristic approaches which have been tested in several fields of application. Particularly in the last ten years, a growing number of applications in the area of finance were investigated. Comparing the tremendous variability of the different heuristic approaches, it is often difficult to decide which heuristic method should be applied to a given financial problem setting. To support the right choice among different heuristic approaches we gather their basic characteristics with a special focus on financial applications. In the following text we mainly introduce the core methodology of different heuristics and present many references which cover theoretical aspects, e.g., convergence properties or parameter choice, and a selection of successful applications in finance. Before discussing the modern heuristic approaches, we will first point out a formal view of complexity and a classical local search algorithm in the following section.

2. Complexity of finance problems and local search algorithms

A widely accepted formal definition of complex problems is the computational intractability of problems which are hard to solve from an algorithmic perspective, i.e., we require huge computational resources to compute the exact solution of a problem having input size n (e.g., an exponential number 2^n of necessary calculations for n given input variables of the considered problem). Besides other complexity classifications, the theory of **NP**-*completeness* yields a well-defined formalisation of such complex problems, see e.g., Garey & Johnson [1] for a detailed coverage of this topic and a large collection of **NP**-*complete* problems. Until now, there is no known algorithm which requires only a polynomial number of computational steps depending on the input size n for an arbitrarily chosen problem that belongs to the class of **NP**-*complete* problems. See e.g., Papadimitriou [2] for the formal definitions of computational complexity and further implications.

Many problems in finance belong to the class of **NP**-*complete* problems, since they have a combinatorial structure which is equivalent (with respect to polynomial-time reductions) to well-known **NP**-*complete* problems, e.g., constrained portfolio selection and related questions of asset allocation are equivalent to **NP**-*complete* knapsack problems. Cf. Seese & Schlottmann [3] for such complexity results.

Therefore, we require approximation algorithms that yield sufficiently good solutions for complex finance problems and consume only polynomial computational resources measured by the size of the respective problem instance (e.g., number of independent variables). For some complex problem settings and under certain assumptions, particularly linearity or convexity of target functions in optimization problems, there are analytical approximation algorithms which provide a fast method of finding solutions having a guaranteed quality of lying within an ε-region around the globally best solution(s). If the considered problem instance allows the necessary restrictions for the application of such algorithms, these are the preferred choice, see Ausiello et al. [4] for such considerations. However, some applications in finance require non-linear, non-convex functions (e.g., valuation of exotic option contracts), and sometimes we know only the data (parameters) but not the functional dependency between them (bankruptcy prediction problems for instance), so there is nevertheless a need for methods that search for good solutions in difficult problem settings while spending only relatively small computational cost. This is the justification for heuristic approaches.

Almost all heuristics that are discussed in this survey belong to *local search algorithms*. This means that for a given current solution x_i to the problem which is to be solved, a promising solution candidate which can be derived by small modifications of x_i is to be chosen to continue the search for the globally best solutions. The set of solutions which are close to x_i will be called the local *neighbourhood* $N(x_i)$ of x_i in the following text. The exact definition of $N(x_i)$ is problem-specific, and an adequate choice for a given problem is often crucial for the success of a certain algorithm. Most heuristics require an initial solution from which they start their search for improvement. In many cases, a random initialisation of this initial solution yields the best outcome of the respective problem solving method on average, i.e., over all possible instances for a fixed problem class.

A standard local search algorithm is *Hill Climbing (HC)* which basically works as shown in Algorithm 1. The algorithm moves from the current solution x_i to a solution x_j from the neighbourhood of x_i if and only if x_j is better than x_i concerning the problem to be solved, otherwise the algorithm stops and returns x_i as the candidate for the globally best solution (in the following text, φ denotes a sample target function that is to be maximized by the respective algorithm).

Algorithm 1. *Hill Climbing*

```
Input: Initial solution $x_j$
$i := 1$ (iteration counter)
Repeat
    Set current solution $x_i := x_j$
    Generate neighbourhood $N(x_i)$
    Choose the best solution $x_j \in N(x_i)$
    If $\varphi(x_j) > \varphi(x_i)$ Then
        TerminateSearch := False
        $i := i + 1$
    Else
        TerminateSearch := True
Until TerminateSearch := True
Terminate
Output: Best solution found $x_i$
```

There are many examples of such HC procedures for solving finance problems in the standard literature, so we do not consider them here in detail. Usually, analytically tractable problems can be solved using Newton's method or other gradient-based approaches. On the other hand, if the problem to be solved contains non-linear, non-convex functions and/or integer constraints, such a local search procedure that decides about the next move solely based on local information in the neighbourhood of x_i, can get stuck in suboptimal solutions without discovering the globally best (optimal) solution throughout the search process. The methods described in the following sections try to avoid this problem by using certain strategies, e.g., for choosing the next solution x_j to be investigated or for deciding about the termination of the search process. Beside the more specific references appearing in the text, general overviews of the concepts presented below can e.g., be found in Reeves [5], Osman & Kelly [6], Aarts & Lenstra [7], Fogel & Michalewicz [8], Pham & Karaboga [9] and Nelles [10] which cover a variety of methods and mostly non-financial applications. A more finance-related, recent survey on a subset of the methods discussed below is Chen's book [11].

3. Simulated Annealing

The basic working principle of *Simulated Annealing (SA)* is an analogy to conducting thermodynamical annealing processes in a heath bath to obtain low-energy states for a solid (cf. Kirkpatrick et al. [12] and Cerny [13]). A simulation algorithm for such a thermodynamical process was introduced in 1953 by Metropolis et al. [14], and the so-called *Metropolis criterion* proposed in their work is also the central part of the SA heuristic that is shown in Algorithm 2 ($Z_i \in [0, 1]$ are assumed to be independently and identically distributed uniform random variates).

Algorithm 2. *Simulated Annealing*

```
Input: Initial solution $x_j$, parameter value $T_1 \in \mathbb{R}$
$i := 1$ (iteration counter)
$x_{best} := x_j$ (best solution found so far)
Repeat
    Set current solution $x_i := x_j$
    Randomly choose a solution $x_j \in N(x_i)$
    If $\varphi(x_j) > \varphi(x_{best})$ Then
        $x_{best} := x_j$
    If $\varphi(x_j) > \varphi(x_i)$ Then
        TerminateSearch := False
    Else
        If $Z_i < e^{-\frac{\varphi(x_i) - \varphi(x_j)}{T_i}}$ Then
            TerminateSearch := False
            $i := i + 1$
            Adapt $T_i$ according to predefined rule (cooling schedule)
        Else
            TerminateSearch := True
Until TerminateSearch = True
Terminate
Output: Best solution found $x_{best}$
```

The parameter T_i which represents the temperature in the heat bath conducts the annealing process. In analogy to the Metropolis criterion, the SA algorithm not only accepts a better solution x_j with probability 1, but also a deterioration of the current solution with probability $e^{-\frac{\varphi(x_i) - \varphi(x_j)}{T_i}}$. For an optimal low-energy state of the solid to be annealed, an appropriate cooling schedule is required for T_i, and this is also crucial if the SA heuristic is applied to finance problems. Aarts et al. [16] provide a thorough analysis of the convergence properties of the SA heuristic using Markov chains, some guidelines for choosing appropriate cooling schedules and many references to applications of the SA heuristic. Here, the main convergence results are shortly summarized by the following theorem (for more details see [16]):

Theorem 1. *Assuming an appropriate cooling schedule for T_i in the SA algorithm, x_i converges to a globally optimal solution with probability 1 for $i \rightarrow \infty$ (i is the number of iterations of the loop in Algorithm 2).*

Regrettably, this does neither hold for a finite number of iterations, nor for an arbitrary cooling schedule. Nevertheless, there are many reports of successful SA applications in the literature, particularly in other areas than finance. A sample finance application was e.g., built by Chang et al. [17] who applied different heuristics including SA to constrained and unconstrained portfolio selection problems. Their goal was to identify the mean-variance efficient frontier (cf. e.g., Markowitz [18]) for given sets of alternative investments. Besides the good results for smaller problems from the Hang Seng, DAX, FTSE or S & P stock market indices, the SA heuristic found a good approximation of a constrained efficient frontier for a draw of 10 from 225 alternative investments from the Nikkei stock market index in less than 10 minutes on a single workstation computer.

4. Threshold Accepting

The *Threshold Accepting (TA)* heuristic was introduced in 1990 by Dueck & Scheurer [19] as a simplification of Simulated Annealing. Instead of applying the Metropolis criterion to decide about the acceptance of a deterioration of the current solution (cf. Algorithm 2), TA uses a straightforward threshold parameter T_i for the maximum deterioration of the current solution that is accepted without terminating the algorithm. Like the temperature cooling schedule in SA, an adaptation rule for T_i is necessary to ensure proper convergence, and the choice of this adaptation rule is a crucial point when applying TA to a problem. Moreover, it has to be emphasized that the success of TA depends heavily on the proper modelling of a neighbourhood $N(x_i)$ for any possible solution x_i. There are both deterministic and non-deterministic variants of TA depending on the adaptation of T_i .

Algorithm 3. *Threshold Accepting*

```
Input: Initial solution $x_j$, initial parameter $T_1 \in \mathbb{R}$ , $T_1 > 0$,
    iteration limit $i_{max} \in \mathbb{N}$
$i := 1$ (iteration counter)
$x_{best} := x_j$ (best solution found so far)
Repeat
    Set current solution $x_i := x_j$
    Randomly choose a solution $x_j \in N(x_i)$
    If $\varphi(x_j) > \varphi(x_{best})$ Then
        $x_{best} := x_j$
    If $\varphi(x_i) - \varphi(x_j) < T_i$ Then
        TerminateSearch := False
        $i := i + 1$
        Adapt $T_i$ according to predefined rule
    Else
        TerminateSearch := True
Until $i > i_{max}$ Or TerminateSearch = True
Terminate
Output: Best solution found $x_{best}$
```

The convergence results for TA are similar to the results for SA (cf. Theorem 1), i.e., they use Markov chain results to derive convergence properties of the algorithm. A good source for many aspects of TA including theoretical aspects as well as econometric applications is Winker [23].

Both Dueck & Winker [20] and Gilli & Kellezi [21] applied TA to portfolio selection problems. Their respective objective was to find the optimal asset allocation that is efficient concerning the aggregate portfolio risk as well as the aggregate portfolio return for a mean-variance approach or a mean/downside risk approach. The studies incorporated the successful application of TA to analytically intractable optimization problems.

Gilli & Kellezi [22] reported the application of TA to an index tracking problem, where the goal of a so-called *passive investor* was to minimize the difference between his tracking portfolio and a given market index (a stock index like Dow Jones Industrial Average for instance) under constraints, e.g., rebalancing and transaction cost, round lots etc. Even for a large number of 528 assets, the TA heuristic found a good approximation for the optimal tracking portfolio within less than one minute on a Personal Computer.

5. Tabu Search

The main ideas of *Tabu Search (TS)* were formulated by Glover [24] and Hansen [25]. This heuristic is different from the other approaches discussed in this survey because it explicitly keeps track of the activities that were performed in the problem solving process so far and tries to conduct this process towards relevant solutions by forbidding certain activities that have already been performed. To achieve this, the definition of a *move* is the central concept in TS. A move from a current solution x_i is the operator that yields a neighbourhood $N(x_i)$, i.e., the (usually small) modification applied to x_i to obtain $N(x_i)$. TS keeps a *Tabu List* of maximum length k that contains the most recently performed moves throughout the problem solving process. These moves are not considered for the next move to be performed, i.e., they are tabu (cf. taboo). An obvious advantage of memorizing recently performed moves instead of recently investigated solutions is that the former method usually requires less memory. The length k of the Tabu List is a crucial parameter for a given problem. If the list is too short the search for optimal solutions might run into a loop (recently visited solutions are revisited again and again), and if the list is too long the algorithm may not find optimal solutions that require the repeated application of certain moves. Algorithm 4 shows an overview of the TS local search method.

Algorithm 4. *Tabu Search*

```
Input: Initial solution $x_j$, iteration limit $i_{max} \in \mathbb{N}$
$i := 1$ (iteration counter)
$x_{best} := x_j$ (best solution found so far)
$T := \emptyset$ (tabu list)
Repeat
    Set current solution $x_i := x_j$
    Generate neighbourhood $N(x_i)$ for $x_i$ using a move $m_i \notin T$
    Choose the best solution $x_j \in N(x_i)$
    If $\varphi(x_j) > \varphi(x_{best})$ Then
        $x_{best} := x_j$
    $i := i + 1$
    Update $T$ according to predefined rule
Until $i > i_{max}$
Terminate
Output: Best solution found $x_{best}$
```

The update procedure for T in Algorithm 4 usually consists of the operation $T := T \cup \{m_i\}$, and the least recently used move m_j is removed from the list by $T := T \setminus \{m_j\}$ (First-In-First-Out principle). In addition, different strategies (*aspiration* of tabu criteria, and *intensification* as well as *diversification* concerning search regions) are discussed in the literature to enhance the performance of the algorithm and to avoid the above mentioned problems of a too restricted search for better solutions. See e.g., Glover & Laguna [26] for a detailed coverage of these and further details of TS. A short introduction and a selection of non-finance TS applications are e.g., given by Hertz et al. [27].

Glover et al. [28] reported a successful application of TS to a multi-period asset allocation problem using a time-dependent mean-variance approach with non-convex constraints which allow for modelling taxes, transaction cost etc. For the absence of taxes and transaction cost in a problem containing 8 assets and 20 time periods, the results of the TS algorithm were nearly identical to an ε-analytical approximative approach for the efficient frontier. When these constraints were included to obtain an analytically intractable problem structure, the TS algorithm also found an approximation of the mean-variance efficient frontier within 17 minutes on a workstation.

The study of Chang et al. [17], which has already been cited in the Simulated Annealing section, also contains results of a TS application for mean-variance portfolio selection problems. TS showed a similar performance like Simulated Annealing in the study, being slightly better in some cases and slightly worse in other concerning the quality of the solutions found, and having a similar runtime of e.g., about 10 minutes for an approximation of the mean-variance efficient frontier in a constrained portfolio selection problem for 10 assets from the Nikkei-225 stock market index.

6. Evolutionary Computation

Some problem solving mechanisms like selection, reproduction and mutation which can be observed in natural environments and populations are the basic working principles for *Evolutionary Computation (EC)*. The history of EC started in the 1950s (see DeJong et al. [29] for an overview of such early work). In the 1960s, Fogel et al. [30] developed the concepts of *Evolutionary Programming*, which was followed later by Koza's introduction of *Genetic Programming (GP)* [31,32,33], Holland [34] proposed the *Genetic Algorithm (GA)* and Rechenberg [35] and Schwefel [36] introduced the *Evolution Strategies (ES)* (cf. also [29]). Since the 1990s these evolutionary approaches have attracted many research activities due to their interesting theoretical properties as well as their successful applications. We will concentrate on the GA and GP methodology because of their dominance concerning finance applications. A good source for many aspects of EC are Baeck et al. [37,38] and the annual conference proceedings of GECCO (e.g., [39]).

6.1. Genetic Algorithms

GAs are randomised heuristic search algorithms reflecting the Darwinian *survival of the fittest* principle that can be observed in many natural evolution processes. A GA works on a set of n potential solutions to a problem rather than on a single solution. The current set of solutions being processed by a GA at each time step t of the algorithm is called *population* or *generation* $P(t) = \{x, y, \ldots\}$, $|P(t)| = n$, and each $x \in P(t)$ is called an *individual*. To apply a GA to a problem, the decision variables have to be transformed into *gene strings*, i.e. each element from the decision variable space D has to be transformed into a string consisting of digits or characters in the search space of the algorithm S by applying a *1-1* function $g : D \rightarrow S$. The original representation $x \in D$ is called *phenotype*, the genetic counterpart $g(x) \in S$ is called *genotype*. For the sake of simplicity, we will not distinguish between x and $g(x)$ in the following text. We will denote the length of the gene string x by *length*(x). A simple GA scheme is shown in Algorithm 5.

Algorithm 5. *Genetic Algorithm*

```
Input: Iteration limit $t_{max} \in \mathbb{N}$
$t := 0$ (population counter)
Generate initial population $P(t)$
Evaluate $P(t)$
Repeat
    Select individuals from $P(t)$
    Recombine selected individuals
    Mutate recombined individuals
    Create offspring population $P'(t)$
    Evaluate $P'(t)$
    Generate $P(t + 1)$
    $t := t + 1$
Until $t > t_{max}$
Output: $P(t)$
```

The initial population $P(0)$ can be generated e.g., by random initialisation of every individual. For evaluation of each individual $x \in P(t)$ the GA requires a so-called *fitness function f* which is defined either on the set of possible genotypes S or the set of phenotypes D. Usually, the fitness function is defined on D and takes real values, i.e., $f : D \rightarrow \mathbb{R}$. This function expresses the quality of the solution represented by the individual. It is problem specific and therefore, it has to be designed according to the problem parameters and constraints. During the evolution process the GA selects individuals for reproduction from the current population $P(t)$ according to their fitness value, i.e., the probability of surviving or creating offspring for the next population $P(t + 1)$ is higher for individuals having higher fitness values. The intention of *selection* is to guide the evolution process towards the most promising solutions. A common method is *tournament selection*, where the fitness values $f(x)$, $f(y)$ of two randomly drawn individuals $x, y \in P(t)$ are compared, and the winner is determined by the rule $f(x) > f(y) \Longrightarrow x$ survives, $f(x) < f(y) \Longrightarrow y$ survives, $f(x) = f(y) \Longrightarrow x, y$ survive. For an overview and a comparison of selection methods, see part 3 of Baeck et al. [37].

Since the GA's task is to explore the search space S to find globally optimal and feasible solutions, e.g., $x^* = \max_{x \in S'} f(x)$ in a constrained maximisation problem where $S' \subseteq S$ specifies the space of feasible solutions, the selected individuals from each $P(t)$ are modified using *genetic operators*, sometimes called *variation operators* (cf. Fogel & Michalewicz [8], p. 173). A typical variation operator for recombination is the one-point crossover, i.e., the gene strings of two selected individuals are cut at a randomly chosen position and the resulting tail parts are exchanged with each other to produce two new offspring. This variation operator is applied to the selected individuals using a problem specific crossover probability $Prob_{cross}$. Common values are $Prob_{cross} \in (0.6, 1)$. The main goal of this operator is to conduct the simulated evolution process through the search space S. Most GAs implement a second variation operator called *mutation*. In analogy to natural mutation, this operator randomly changes the genes of selected individuals with the parameter $Prob_{mut}$ per gene to allow the invention of new, previously undiscovered solutions in the population. Its second task is the prevention of the GA stalling in local optima because of high selection pressure since there is always a positive probability to leave a local optimum if $Prob_{mut} > 0$. Usually, $Prob_{mut}$ is set small, e.g., $Prob_{mut} \leq \frac{1}{length\ (x)}$. For a survey of different variation operators see e.g., part 6 of Baeck et al. [37]. The creation of $P(t + 1)$ is usually performed by selecting the n best individuals either from the offspring population $P'(t)$ or from the joint population containing both the parent and the offspring individuals $P(t) \cup P'(t)$.

The evolution of individuals during the GA's search process can be modelled by Markov chains (cf. e.g., Rudolph [40]). There are different results concerning the convergence of the GA population towards global optimal solutions depending on the fitness function, properties of the search space etc., see e.g., Muehlenbein [41], Droste et al. [42], Vose [43], Wegener [44]. Despite the theoretical value of these results, they can only provide some rules of thumb for proper genetic modelling of most application-oriented problems, for choosing adequate fitness functions, selection mechanisms, variation operators and parameters like n, $Prob_{cross}$, $Prob_{mut}$. These choices are not necessarily independent from each other, e.g., the choice of the fitness function is important for the selection mechanism and vice versa. As a consequence, there is currently no GA fitting each application-oriented context. Therefore, most application studies focus on empirical evaluations of problem specific GAs.

A very natural application of GAs is the modelling of a group of individual entities, e.g., traders in financial markets, to observe the emerging macro-level output (e.g., asset prices) from individual decisions. Such approaches benefit from the fact that the GA provides a built-in adaptation mechanism through its evolution process which can be used to model individuals that try to improve their financial decisions by processing historical information. An introduction to such approaches in economics is given by Riechmann [45]. The study by Rieck [46] and the Santa Fe artificial stock market (see e.g., Tayler [47]) are two examples for early studies that analyzed asset prices resulting from individual decisions made by artificial traders which are improved by a GA. Meanwhile, there is a large number of similar approaches which analyze e.g., asset price time series properties (cf. LeBaron et al. [48]), the Efficient Market Hypothesis (cf. Coche [49], Farmer & Lo [50]) and further questions related to real-world asset markets.

Another common field of GA applications in finance is the discovery and the classification of patterns in financial data, e.g., for evaluation of counterparties in credit scoring models (cf. e.g., Walker et al. [51]), for detecting insider trading (cf. e.g., Mott [52]) or for the identification of successful trading strategies in stock markets (cf. e.g., Frick et al. [53]). The book by Bauer [54] covers many aspects of such approaches. An interesting application in this context is Tsang & Lajbcygier's [56] modified GA framework for the discovery of successful foreign exchange trading strategies in historical data. It uses a split population that is divided into a group of individuals which is modified using a higher mutation probability, and another group of individuals which is changed with a low mutation probability. The modified GA found strategies yielding higher mean returns while having similar return standard deviations than a standard GA, but both approaches showed advantages over the other in certain test criteria.

6.2. Genetic Programming

The concept of GP is very similar to the GA paradigm, since it uses the same elements like selection, reproduction and mutation in its core and implements the same scheme like Algorithm 5. However, there is a significant difference concerning the genetic representation of the individuals in the considered population as e.g., pointed out by Koza et al. [33], p. 31:

Genetic programming is an extension of the genetic algorithm in which the genetic population contains computer programs.

Rather than being genetic representations of single solutions, the genotypes in a GP algorithm represent programs that are candidates for solving the considered problem. The most widely used representation is the *Automatically Defined Function* by Koza [32] where a program is represented by a syntax tree containing variables and operators that can be used to formulate mathematical expressions which are similar to those that can be formulated in the programming language LISP. These expressions are evaluated by setting the variables' contents to different input parameter sets which are instances of the problem to be solved. The output of this evaluation yields a fitness value for the program concerning the problem to be solved. For a further discussion of different representations see e.g., Langdon & Poli [57], p. 9 ff.

Therefore, at each step t the population $P(t)$ contains different programs each of which is evaluated by running it on a number of instances (i.e., input parameter values) for the given problem type and observing its performance (cf. the fitness evaluation in a GA). The methods to prove convergence of $P(t)$ to globally optimal solutions (i.e., programs which solve the given problem instances well concerning pre-defined performance criteria), are similar to the GA convergence analysis methods and partially, yield even the same convergence properties like GAs. For a detailed discussion of these topics see also [57].

An obvious application of GP is the approximation of a priori unknown functional dependencies: Given different sets of input parameter values and their associated output values, the goal is to find a function that describes the dependency of the output on the input approximately. This kind of GP application was e.g., used by Keber [58] to find closed-form approximations for valuing American put options on non-dividend paying stocks instead of using finite differences (cf. Brennan & Schwarz [59]) or the tree approach by Cox et al. [60]. Keber compared the results of a GP-based search for an approximation formula to frequently quoted approximation procedures in the literature which differ from the two numerical approaches cited above. The best formula found by his GP approach was quite similar in its structure to some of the existing approximations, but it outperformed the other approximations concerning the numerical accuracy of the resulting option values both on a standard test data set used by several other studies and on another large sample of theoretical American put option prices.

In a series of papers (see [61] for further references), Li & Tsang developed a GP approach to support investment decisions. They used a decision tree containing rules like 'IF Condition Is Met THEN Buy Asset' which was improved by GP using daily asset price data of 10 US stocks. They compared the results on the test data to the results of both a linear classification approach and problem-specific Artificial Neural Networks obtained by another study and found that the GP approach yielded better results (i.e., higher excess returns) on their test data set. For a related problem setting, where a GP algorithm was used to solve a multi-period constrained asset allocation problem for an Italian pension fund see Baglioni et al. [62].

Similar to the studies cited in the GA section which model individual decision-making and adaptation of rules in a dynamic environment, there are also applications of the GP methodology to study properties of artificial financial markets. See e.g., Chen & Kuo [63] and Chen [11] for a more detailed coverage of this subject.

7. Artificial Neural Networks

An *Artificial Neural Network (ANN)* is a computing model which is built in analogy to the structure of the human brain. A remarkable work which introduced such a computing model based on artificial neurons was published by McCulloch and Pitts in 1943 [64]. But it took about 40 years until the ANN approach (sometimes called *connectionist* approach) reached a wide-spread interest during the 1980s. Since then, many successful real-world applications have been reported, from which we will cite some finance-related examples below after a brief introduction to some methodic details.

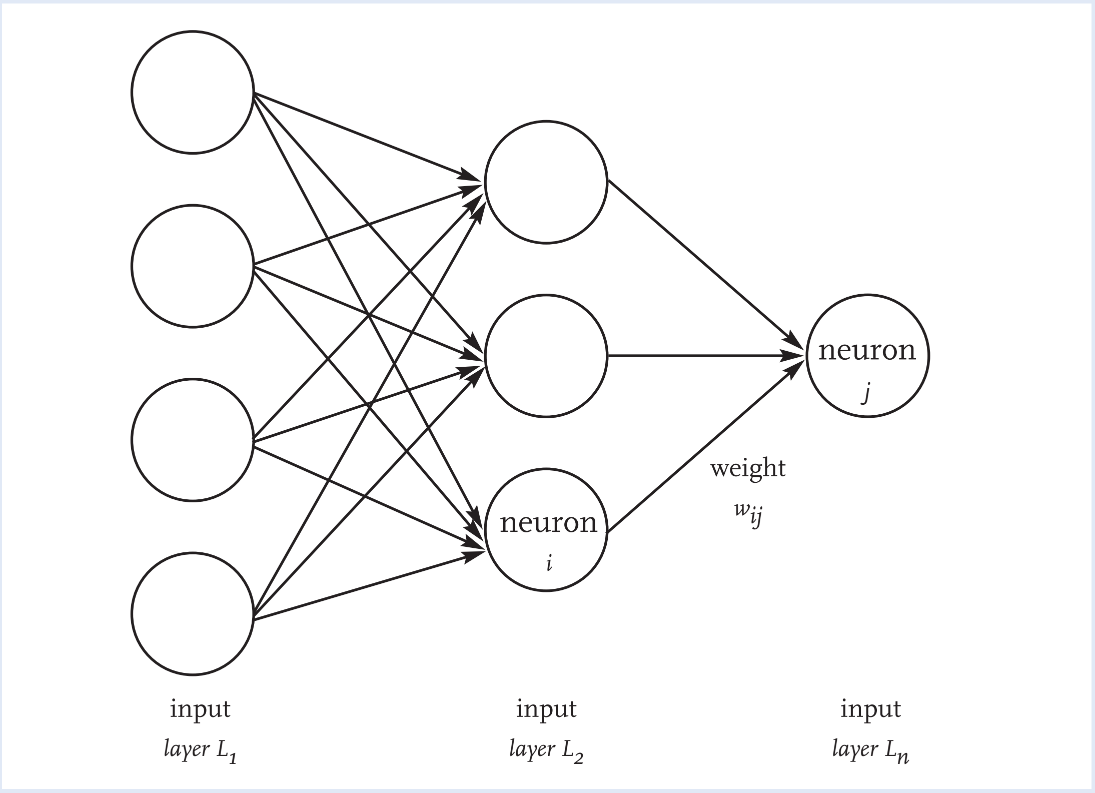
An ANN consists of $m \in \mathbb{N}$ artificial *neurons*, where each neuron $j \in \{1, \ldots, m\}$ represents a small computation unit performing basic functionality, e.g., taking the values of a vector x containing $s_j \in \mathbb{N}$ variables $x_i \in \mathbb{R}$, $i \in \{1, \ldots, s_j\}$ as input, which are modified by an *activation function* a_j, and calculating its output, denoted by $output_j$, from the sum of the activation function values $a_j(x_i)$ according to a specified *output function* o_j as follows:

$$output_j := o_j\left(\sum_{i=1}^{s_j} a_j(x_i)\right). \tag{7.1}$$

In many applications, the functions a_j are linear functions of the type $f(x) := w_{ij}x_i$ and the o_j are non-linear functions like $f(x) = tanh(x)$ *(tangens hyperbolicus)* or $f(x) = \frac{1}{1-e^{-x}}$ *(logistic function)*. Each w_{ij} is called *weight* of the input value i for neuron j.

To obtain more computational power, the neurons can be connected pairwise by directed links, e.g., connecting the output of neuron i with the input of neuron j. From a graph theoretic point of view, each neuron in an ANN can be represented by a vertex and each link between two neurons can be modelled by a directed edge between their vertices. There are different kinds of ANN paradigms depending on the theoretical and/or application context. We focus on the *Multi-Layer-Perceptron (MLP)* paradigm here since it is a widely used neural computing model in finance applications. A discussion of major ANN computing models is e.g., given by Schalkoff [65]. Arbib's handbook [66] contains a large collection of papers concerning all kinds of ANNs. Kohonen's book [67] is the standard reference for the *Self Organizing Map* ANN paradigm which we do not cover here, while Deboeck & Kohonen [68] contains some interesting financial applications of this paradigm. A survey of recent work concerning this ANN model is given in Seiffert & Jain [69].

Figure 1: Multi-Layer-Perceptron Network



A MLP network M usually consists of $n \geq 3$ layers $L_1, L_2, \ldots, L_n$, each containing at least one neuron. The first layer L_1 is called *input layer*, the last layer L_n is called *output layer*. All possible layers between the input and the output layers are called *hidden layers*. The connections between the neurons build a simple structure: The output of each neuron in layer L_k is connected to the input of each neuron in its succeeding layer L_{k+1} except for all neurons in L_n. There are no other links in this ANN topology. An example is shown in Figure 1.

A single computation step in a MLP network M works as follows: Each component value of an input vector $x = (x_i)_{i=1,\ldots,|L_1|}$ is put directly into the corresponding neuron i of the input layer L_1. The output value from every input neuron i resulting by evaluation of $o_i(x_i)$ is propagated to each neuron j of the first hidden layer L_2. Each neuron $j \in L_2$ calculates its output value from these input values according to equation (7.1). These results are propagated to the next layer and so on. Finally, the output values of the neurons in layer L_n are the *total output* $M(x)$ of the MLP network for the given input vector x. Due to this computing method, the MLP is a *feed forward* network.

To use this computation scheme for an approximation of a given target function g, a proper *learning* method *(training)* is needed. Training a MLP network means starting from a random initialisation of the weights w_{ij} which have to be adapted in a way so that for any *training vector* x the total output of the network $M(x)$ gets very close to the known result of $g(x)$, i.e., $|M(x) - g(x)| < \varepsilon$ for a given $\varepsilon < 0$. It is interesting to note here that g is not needed in functional form, since it is sufficient to know the correct output $M(x)$ for each training vector x. This is particularly important for *data mining* applications in finance where a priori unknown information has to be learned from data sets and for other applications like nonparametric prediction problems, where the functional dependencies between exogeneous and endogeneous variables are a priori unknown.

There are different methods discussed in the literature to adapt the weights according to an error function during training. Most of these methods use the gradient of the quadratic error function (7.2) for neuron j:

$$err_j(x_i) = \frac{1}{2}\,(output_j\,(x_i) - t_j\,(x_i))^2 \qquad (7.2)$$

where x_i is the input vector and $t_j(x_i)$ is the desired output from neuron j for this input. Probably due to its simplicity, the most popular method for training a MLP net using this scheme is *backpropagation*. It works basically as follows: A training vector x is presented, the net computes its output $M(x)$, and the error is computed by applying (7.2) to all output neurons. Afterwards, the gradient values of this error function are computed for the weights of the incoming links of the output neurons, and these weights are adapted by

$$\Delta\, w_{ij} = -\ \eta \frac{\partial}{\partial w_{ij}} err_j \qquad (7.3)$$

where $\eta \in (0, 1)$ is a global parameter of the MLP net called *learning rate*. This process is repeated from the output neuron layer L_n to the preceding hidden neuron layer L_{n-1}, and so on until the input neuron layer is reached (see [70] for further details). As the name of this training method states, the network error is propagated backwards through the MLP network. It is easy to see that backpropagation and related gradient descent methods can particularly run into potential local minima of the error function or oscillate in steep valleys of the error function, therefore a number of modifications are discussed in the literature to restrict these problems, e.g., by using second-order derivatives or conjugate gradient approaches, see e.g., Hecht-Nielsen [71] for an overview.

After training of an appropriately constructed MLP network, the network should be able to 'know' the functional dependence between an input vector x and the resulting output $g(x)$. Therefore, the network can be used for *generalisation*, i.e., a number of new input vectors x not used during the training process can be presented to the network and the network outputs $M(x)$ will be considered as an approximation of the usually unknown values of $g(x)$. This is comparable to the application of non-linear regression models, e.g., for time series prediction.

To obtain a good generalisation capability of a MLP, the problem of *overfitting* has to be avoided: Many instances show that the generalisation capabilities of trained MLPs will be sub-optimal if the MLP memorises the training vectors perfectly. There are different strategies to overcome this situation, e.g., by an early termination of the learning process before final convergence of the weights. For a discussion of this problem and its avoidance see e.g., [65], p. 195ff.

An interesting result concerning the general approximation capabilities of MLP networks is the following theorem by Hornik [72]:

Theorem 2. *Given a measure μ, a distance $\varepsilon > 0$ and assuming the presence of one hidden layer using bounded, non-constant activation functions a_j, there is an MLP which can ε-approximate any function $f \in L^p(\mu)$, where $L^p(\mu) := \{ f : \int_{\mathbb{R}^n} |f(x)|^p \, d\mu(x) < \infty \}$.*

There are many other results proving the approximation capabilities of ANNs for other restrictions on the target functions, the activation or output functions (see Anthony and Bartlett [73] for a detailed theoretical analysis). A little disadvantage of all these proofs is that they do not provide exact construction guidelines for ANNs in applications. So the topology of the network, particularly the number of necessary hidden layer neurons which has a strong influence on the approximation properties and further parameters have to be chosen e.g., by empirical tests or by experiences from similar cases in each application context. However, a general guideline for adequate ANN modelling and validation in finance applications is provided by Zapranis and Refenes in [74].

Due to the difficulty of selecting an adequate ANN topology and the necessary parameters for a given problem, many recent studies combine ANNs with other machine learning or optimization concepts in the following sense: The given problem is solved by different ANNs and a meta-learning algorithm observes the performance of the ANNs on the problem. The meta-learning algorithm tries to improve the ANNs by varying their topology, their parameters etc. We will return to this point in section 9.

Many finance-related ANN applications used MLP networks for the prediction of time series, e.g., daily stock returns [75], futures contracts prices [76], real estate returns [77] or three-year default probabilities of companies [78]. Refenes [79] provides a good survey on selected ANN applications for equities (e.g., modelling stock returns, testing the Efficient Markets Hypothesis), foreign exchange prices (e.g., prediction of exchange rates, development of trading rules), bond markets and other economic data.

A standard application of ANNs for classification purposes in finance are bankruptcy prediction and related tasks like bond rating, obligor classification into rating categories, etc. Besides other finance-related ANN topics, Trippi and Turban [80] offer a collection of such classification applications. Other examples are Odom and Sharda [81], Coleman et al. [82], McLeod et al. [83], Baetge and Krause [84], Wilson and Sharda [85]. Most of these studies reported better empirical classification performance of Neural Networks in comparison to standard models like discriminant analysis (for a description of this method see e.g., [86]) or linear regression, except for the study of Altman et al. [87]. A better performance of ANNs is not surprising from a theoretical view if we consider the non-linear approximation abilities of the ANNs (see Theorem 2 above) compared to the limited capabilities of linear models.

The learning capabilities of ANNs can be used for modelling adaptive traders in artificial markets to study the macro-level behaviour of such markets, e.g., measured by asset prices, depending on the micro-level decision strategies and learning capabilities of the traders (cf. also our remarks in the GA section). See e.g., Beltratti et al. [88] for examples of such models.

A number of articles studied the approximation capabilities of MLP networks for option valuation purposes, e.g., Mallaris and Salchenberger [89], Hutchinson et al. [90], Lajbcygier et al. [91], Hanke [92], Herrmann and Narr [93]. Lajbcygier [94] gives a review of related work. An interesting observation in [90] was that a small MLP network using only four hidden neurons in one hidden layer and backpropagation training yielded a good approximation of the results obtained by applying the Black/Scholes formula (cf. [95], [96]) to given option pricing parameters (e.g., the mean R^2 over different option pricing parameters was $R^2 = .9948$). Furthermore, a narrative conclusion of all the above studies is that the results from the training of ANNs using market data can yield a significantly better approximation of option prices compared to closed form option pricing formulas.

Locarek-Junge and Prinzler [97] applied MLPs and backpropagation to market risk (here: *Value-at-Risk*) estimation where the goal of the calculation is to estimate a certain α-percentile (e.g., $\alpha = .99$) of the distribution of future losses from a portfolio whose value depends on a fixed number of market risk factors. They used a Gaussian Mixture Density Network (cf. [98] for details) defined by the following equations:

$$f_{X|Z=z}(x) = (\sum_{i=1}^{m} a_i(z)h_i(x|z), \qquad (7.4)$$

$$h_i(x|z) = (2\pi)^{1/2}\sigma_i^{-1}(z)e^{-\frac{\|x-\mu_i(z)\|^2}{2\sigma_i^2(z)}}, \qquad (7.5)$$

where $a_i \geq 0$, $\Sigma_i a_i = 1$ were the mixture parameters to be estimated by a MLP network backpropagation training that minimised the maximum likelihood error function of the specified model given historical observations of market risk factor changes Z and historical target function values (market risk returns) X. The results from the Mixture Density Network estimation for a USD/DEM exchange rate dependent sample portfolio were compared to the results from a historical simulation (see e.g., Jorion [99]) and from a variance-covariance approach proposed by the *RiskMetrics*^{TM} model [100]. The Neural Network based approach yielded a better estimate particularly for the $\alpha = .99$ percentile if at least two different a_i had to be determined by the learning process, i.e., if there were two or more densities to be included in the mixture.

Naim et al. [101] combined MLP networks as a forecasting tool for financial markets with ideas from asset allocation for portfolio management. Their considerations rely on the predictability of real-world asset returns by ANNs. An interesting remark pointed out by Naim et al. is the explicit difference between a two-step approach that separates the prediction of the necessary asset parameters by ANNs from the portfolio choice problem, and an integrated approach that optimizes both the prediction error and the portfolio choice alternatives in one backpropagation training process. The results of an empirical study simulating a strategic asset allocation problem using data from G7 countries' capital markets indicated that the two-step approach was dominated by the integrated approach. The annualised return was higher and the standard deviation was even smaller for the portfolio calculated by the latter approach compared to the portfolio resulting from the two-step approach running on the same data set. Both methods were significantly better than a standard Markowitz [18] mean-variance portfolio selection procedure based on historical estimation of parameters from the data set.

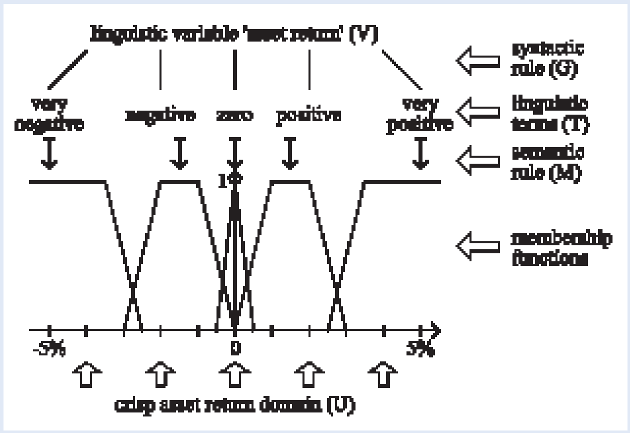
Bonilla et al. [102] used MLP nets trained by backpropagation to forecast exchange rate volatilities of six currencies against the Spanish Peseta from historical observations and compared the results to different parametric GARCH-type models (see e.g., [103] for an overview of volatility modelling). In their study the MLP nets' forecasting accuracy was superior for three currencies and inferior for the other three currencies, but the results of the cases where the MLP nets were superior to the parametric models were significantly better than the cases where the MLP networks were inferior.

In general, ANNs were successfully applied to many non-linear and complex finance problems, e.g., time series prediction, classification and modelling of learning entities in financial simulations. This is especially true for MLPs which are the most widely used ANN computing paradigm in finance. The success of ANNs is mainly caused by their excellent approximation capabilities for non-linear dependencies that are even theoretically justified for many classes of functions. But besides the achieved results, an exact algorithm for constructing a suitable ANN for a given, arbitrary problem has not been found yet. It is still a matter of experiments and experiences to construct an adequate ANN, therefore this is considered to be a challenge for hybrid approaches discussed later in this article. And it must be emphasized that many approximation capability proofs assume a large or even infinite number of neurons in the ANN resulting in a significantly larger number of weights to be adapted. The dimension of the corresponding weight optimization problem usually grows over-proportional in the number of neurons, therefore the resulting learning problem will get very complex if we have to use many neurons in our finance applications. More precisely, the problem of learning in an MLP is **NP**-*complete* (see [104] for an overview of related results). Finally, it has to be kept in mind that an ANN approach is mainly a data driven problem solving method, i.e., the availability of sufficiently representative, high-quality data sets is a crucial point in all ANN applications.

8. Fuzzy Logic

In contrast to the approaches which have been presented so far in this survey, *Fuzzy Logic (FL)* is not a local search algorithm, but a heuristic approach invented in 1965 by Zadeh [105] that supports the use of vaguely defined variables in calculations and logical inferences.

Figure 2: Linguistic variable for asset return



The building block of the FL framework is a so-called *fuzzy* set which is a generalization of a set in the traditional sense of Cantor. In traditional (in Fuzzy theory also called *crisp*) sets, each element e from a universe of discourse U can either be member of a set $A \subseteq U$ (denoted by $e \in A$) or not be member of A ($e \notin A$). Using a *membership function* $m_A : U \rightarrow \{0, 1\}$ this can be modelled as follows:

$$\forall e \in U : m_A(e) = \begin{cases} 1 & e \in A, \\ 0 & \text{otherwise} \end{cases} \qquad (7.6)$$

A fuzzy set generalizes this definition by allowing arbitrary functions of the type $m_A : U \rightarrow [0,1]$ to express the degree of membership of an element e from U to the set A. Therefore, a fuzzy set over U is defined by the pair $F_A := (U, m_A)$ where m_A is the membership function of F_A. The usual set operations were originally generalized by Zadeh [105] as follows:

Given are two fuzzy sets $F_A := (U, m_A)$ and $F_B := (U, m_B)$.

- The complement of F_A is $F^c_A := (U, m^c_A)$ satisfying $\forall e \in U : m^c_A(e) := 1 - m_A(e)$.
- The union $F_C = F_A \cup F_B$ where $F_C := (U, m_C)$ is obtained by setting $\forall e \in U : m_C(e) := \max\{m_A(e), m_B(e)\}$.
- The intersection $F_C = F_A \cap F_B$ where $F_C := (U, m_C)$ is defined by $\forall e \in U : m_C(e) := \min\{m_A(e), m_B(e)\}$.

Based on these properties, relations between fuzzy sets and an arithmetic for fuzzy numbers can be defined, see e.g., Klir [107,108] for a detailed coverage of these topics.

An important goal of fuzzy modelling in applications is *computing with words*. Since computations on a machine have to be done by using accurate numbers while human problem solving methods are often more heuristically, FL is commonly used as an interface between problem solving knowledge stated in a natural language and corresponding computations in exact arithmetic to be performed by a machine. For this functionality, the concept of *linguistic variables* is essential. A linguistic variable L is a tupel $L := (V, T, U, G, M)$ where V is the name of the variable, T is the domain of linguistic terms for V, U is the domain of crisp values for V (universe of discourse), G is the syntax rule (grammar) for building the linguistic terms and M is the semantic rule that assigns a fuzzy set F_t to each $t \in T$. Figure 2 shows an example for a linguistic variable that describes asset returns.

Using such linguistic variables, many FL applications incorporate *fuzzy rules*. Assume we have defined the following linguistic variables:

$$L_{input} := (V_{input}, T_{input}, U_{input}, G_{input}, M_{input}), \qquad (7.7)$$
$$L_{output} := (V_{output}, T_{output}, U_{output}, G_{output}, M_{output}). \qquad (7.8)$$

Then we can formulate rules of the following form:

$$R_i : IF\ V_{input} = t_1 \text{ THEN } V_{output} = t_2 \qquad (7.9)$$

where $t_1 \in T_{input}$ and $t_2 \in T_{output}$. The input (independent) variable is described by the linguistic variable L_{input} , and the output (dependent) variable is defined using the linguistic variable L_{output} . A *fuzzy system* uses a rule base RB containing $k \in \mathbb{N}$ such rules $R_1, \ldots, R_k$. Note that both the input and the output variables in (7.9) can be aggregations of different linguistic variables using the fuzzy set operators (negation, intersection, union etc.) for combining the fuzzy values of linguistic variables. To use the fuzzy system e.g., for the approximation of the crisp output $g(e)$ that belongs a given crisp input $e \in U$ for a function $g : U \rightarrow \mathbb{R}$, the rule base has to contain rules describing the functional dependence between the input and the output using fuzzy rules like (7.9). These rules are usually heuristic interpretations of the dependence between the crisp input and the crisp output variables. They can either be maintained by a human expert, or be generated and adapted for instance by a local search algorithm.

Algorithm 6. *Fuzzy System Computation*

```
Input: crisp value $e \in U$, fuzzy rule base $RB$
Fuzzificate $e$
$\forall R_i \in RB$: Aggregation of IF condition to determine rule fulfilment of $R_i$
$\forall R_i \in RB$: Activation of $R_i$ to calculate output activation (THEN part)
Accumulation of the output of all $R_i \in RB$ to obtain output fuzzy set
Defuzzification of the output fuzzy set to obtain crisp $output(e)$
Output: $output(e)$
```

Assuming the presence of adequate rules in the rule base, the fuzzy system computes the crisp $output(e)$ which approximates $g(e)$ for a given arbitrary, but fixed crisp input $e \in U$ using the scheme shown in Algorithm 6 (cf. Nelles [10], p. 304). There are a number of aggregation, activation, accumulation and defuzzification schemes in the literature depending on the type of fuzzy system and the application context. See also Nelles [10], p. 304 ff. for an overview.

Concerning the approximation capability of fuzzy systems, Wang [109] proved the following interesting result (cf. his original work for a more precise formulation that includes the specification of the fuzzy system used in the proof, and also Kosko [110] for further approximation results):

Theorem 3. *Given is an arbitrary function* $g : U \rightarrow \mathbb{R}$ *where* U *is a (crisp) compact subset of* $\mathbb{R}^n$, *and a real number* $\varepsilon > 0$. *Then there is a fuzzy system such that* $\sup_{e \in U} |output(e) - g(e)| < \varepsilon$.

A natural field for FL applications in finance are rule-based systems which have to deal with vague, non-quantitative or uncertain inputs. The book by von Altrock [111] covers the basic fuzzy system methodology which is necessary for real-world implementations, and a variety of applications concerning the creditworthiness check of obligors, fraud detection etc.

Rommelfanger [112] describes a fuzzy system for checking the credit solvency of small companies. It is based on lingustic variables e.g., for inputs like market share, market growth, rate of innovation and contains rules like 'IF MarketShare = positive AND MarketGrowth = medium AND RateOfInnovation = medium THEN Sales = positive' which were provided by human experts. The system is used by a German bank to evaluate potential obligors in a credit rating process. Similar applications were reported e.g., by Weber [113], and in the book by Ruan et al. [114]. This book contains a selection of other recent FL applications in finance and risk management, e.g., several articles concerning the interesting difference between using probabilistic decision theory or making fuzzy decisions in risky situations, an application of FL in electricity market pricing, and a contribution that proposes a Discounted Fuzzy Cash Flow model for capital budgeting.

In the same volume, Korolev et al. [115] extended Merton's model [116] of the valuation of a premium for bank deposit insurance which had originally used the Black/Scholes option pricing formula [95, 96] to determine the risk-based premium. In their study the Black/Scholes valuation framework is extended by considering a so-called *Fuzzy Valued Asset* as the underlying, i.e., the crisp stock price in the Black/Scholes framework is replaced using a fuzzy set on the payoff scale as the universe of discourse. According to Korolev et al., the use of the fuzzy underlying particularly avoids the criticism against the original Merton model which used a rather unrealistic crisp payoff structure generated by the crisp stock price, and makes the model more useful for real-world applications.

9. Hybrid Approaches

All problem solving approaches discussed in the preceding sections (and of course, also other methods which were not mentioned above) inhibit certain strengths and certain weaknesses. For instance, HC is a very fast local search method that causes low computational cost but suffers from the danger of getting stuck in sub-optimal solutions. In contrast to this trade-off, randomized algorithms like SA, TA and EC use heuristic mechanisms to reduce the risk of early convergence to sub-optimal solutions at the price of slower convergence speed and potentially higher computational cost to find reasonably good solutions. This observation suggests the combination of different problem solving methods to emphasize the overall benefits of the resulting, so-called *hybrid approach* and obtain less weakness of the combined approach compared to the sum of individual weaknesses of its ingredients. Of course, the combination should use the minimum number of different methods which leads to the desired properties of the hybrid approach, otherwise it will be difficult to analyse and predict its behaviour. See e.g., Fogel & Michaelwicz [8], p. 391 ff. for a discussion of these issues.

Moreover, all problem solving methods presented in the preceding sections require an adequate representation of the parameters of the problem to be solved as well as initial parameter values which lead to a desired performance of the respective problem solving method. An appropriate combination of methods can be used to derive both the necessary parameters to obtain good solutions and the solutions themself, which will provide strong support to the user who has to determine the parameters manually, otherwise.

For an overview of different general aspects of hybridization as well as a classification of different approaches, see Goonatilake & Khebbal [117]. The book by Abraham & Koeppen [118] contains a selection of recent developments in the area of hybrid systems, while Rutkowska's recent book [119] particularly covers the fusion of ANN and FL. This topic is also addressed by Jin [120], as well as the integration of EC and FL. While these and many other surveys concentrate on non-finance problems, we will now consider some financial applications.

Since the mid- and late 1990s, many hybrid systems for the support of asset trading in financial markets have been created and integrated into commercial trading platforms. Besides the huge number of commercial products and software in this area there are also many academic publications which report successful applications of hybrid methods. For instance, Herrmann et al. [123] created a combination of a GA and a fuzzy system to support equity traders. The GA worked on a population of individuals, each of which represented a FL rule base consisting of rules like 'IF PriceEarningsRatio = high AND AverageTradingVolume = high THEN buy'. For each individual, both the rules themself and the fuzzy membership functions used in the linguistic variables were optimized by the GA using daily stock price information of the 30 companies listed in the German DAX index over a period of seven years. The hybrid system e.g., yielded a positive excess return over the DAX index performance in 66% to 75% of the fuzzy rule applications.

Siekmann et al. [124] used a combination of ANN and FL (*NeuroFuzzy system*) to predict the sign of the daily returns of the German DAX index. The fuzzy system rule base contained rules based on different technical indicators in the 'IF'-condition which were evaluated using historical stock price information, and the fuzzy conclusion in the 'THEN' part was either 'NextDailyDAXReturn = positive', 'NextDailyDAXReturn = neutral' or 'NextDailyDAXReturn = negative'. An MLP network was used to represent the fuzzy system rules, which were initially provided by a human expert, and this network was trained using daily DAX returns. The goal of the MLP training was to remove inappropriate rules from the fuzzy system by pruning their representing nodes from the MLP network, and to adapt the membership functions of the linguistic variables in the remaining fuzzy rules. After the training, the remaining rules were applied to an additional test set containing daily DAX returns not used in the training, and the trading rules based on the prediction by the NeuroFuzzy system e.g., yielded a much higher daily return than naive trading rules based on the previous change of the DAX or than the fuzzy system rules originally provided by the human expert.

For another study that has a similar focus, but discusses the hybridisation of ANN and GA to support trading decisions in the US 30-year T-bond future market, see Harland [125].

In a series of recent papers we proposed a hybrid approach that combined GA and HC to compute risk-return efficient portfolio structures for a discrete set of credit portfolio investment alternatives under constraints. The hybrid approach was implemented and empirically tested both using a single objective function that related the total net risk adjusted return to the total unexpected loss (measured by downside risk) of a given credit portfolio (cf. Schlottmann & Seese [126]), and using two separate objective functions for risk and return (cf. Schlottmann & Seese [127])). For instance, in the latter study the hybrid approach required 3 minutes to find an approximation for a constrained, global Pareto-efficient set based on 20 assets on a standard Personal Computer, while the upper computational bound for this problem determined by a complete enumeration of the search space was 72 minutes. Moreover, the hybrid approach showed higher convergence speed towards feasible, optimal solutions while consuming low additional computational resources compared to its GA counterpart without the additional HC component.

The above examples of successful hybridisation in financial contexts are the basis for some conclusions and possible future research directions which are derived in the final section below.

10. Conclusions

In the preceding sections we have discussed different heuristic approaches which were successfully applied to a variety of complex financial problems. The respective results underline the fact that heuristic approaches are an interesting alternative to other problem solving algorithms e.g., if a problem is computationally very hard, if the problem's parameters cannot be defined using exact bounds or if the functional dependency between input and output and/or the set of input parameters is a priori unknown.

However, we have also pointed out that besides the advantages, there are certain design problems when choosing a heuristic approach. The representation of the exogeneous variables (e.g., decision variables), the parameters of the heuristic algorithm, etc. have to be chosen carefully to obtain good results, and this is itself a non-trivial and in many cases complex task. Beyond that, the early success of certain heuristic approaches, e.g., ANNs, caused too high expectations towards the results, which simply had to create disappointment due to the complexity and sometimes the dynamics of the problems to be solved, for example in stock market analysis and prediction.
Moreover, none of the heuristic approaches fits into all problem contexts and some were especially useful in certain problem settings. For example, the stochastic search heuristics like Simulated Annealing, Threshold Accepting, Tabu Search as well as the different methods from the Evolutionary Computation paradigm were particularly successful in combinatorial problems while Artificial Neural Networks mainly yielded good results for function approximation, e.g., in non-parametric, non-linear regression. Evolutionary Computation is also a natural approach to modelling evolving and learning financial entities, and Fuzzy Logic is the first choice for problems which cannot be modelled well using crisp numbers or crisp sets.

Much recent academic effort in the area of heuristics has been spent on hybrid methods, which try to combine the strengths of different approaches. These methods seem to be promising for future studies developing modern heuristics in the finance context since many of them incorporate an estimation mechanism that determines necessary algorithmic parameters instead of relying on trial-and-error experiments by the user. We think that another crucial point for future success is the explicit exploitation of problem-specific financial knowledge in the algorithms instead of applying the standard heuristic algorithm scheme to the problem under consideration. The flexibility of many of the heuristic approaches discussed in our survey concerning the integration of problem-specific knowledge is the true strength and one of the best justifications for choosing such methods.

Dipl. Wi.-Ing. Dr. Frank Schlottmann
Studium und Promotion an der Universität Karlsruhe (TH). Parallel dazu unternehmerische Tätigkeit im Bereich Informationstechnologie-Consulting und -Training. Seit 1994 bei GILLARDON tätig in den Bereichen Entwicklung, Beratung und Projekte mit aktuellem Schwerpunkt Kreditrisiko und Research. Zahlreiche internationale Publikationen und Vorträge im Bereich des finanziellen Risikomanagements.

Prof. Dr. Detlef Seese
Leiter der Forschungsgruppe Komplexitätsmanagement am Institut für Angewandte Informatik und Formale Beschreibungsverfahren der Universität Karlsruhe (TH).

References:

[1] M. Garey, D. Johnson, Computers and intractability, New York, W. H. Freeman & Company, 1979.

[2] C. Papadimitriou, Computational complexity, Reading, Addison-Wesley, 1994.

[3] D. Seese, F. Schlottmann, The building blocks of complexity: a unified criterion and selected applications in economics and finance, presented at Sydney Financial Mathematics Workshop 2002, http://www.qgroup.org.au/SFMW

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and approximation, Heidelberg, Springer, 1999.

[5] C. Reeves (ed.), Modern heuristic techniques for combinatorial problems, Oxford, Blackwell Scientific Publishers, 1993.

[6] I. Osman, J. Kelly (eds.), Meta-heuristics: theory and applications, Dordrecht, Kluwer, 1996.

[7] E. Aarts and J. Lenstra (eds.), Local search in combinatorial optimization, Chichester, John Wiley & Sons, 1997.

[8] D. Fogel, Z. Michalewicz, How to solve it – modern heuristics, Heidelberg, Springer, 2000.

[9] D. Pham, D. Karaboga, Intelligent optimization techniques, London, Springer, 2000.

[10] O. Nelles, Nonlinear system identification, Heidelberg, Springer, 2001.

[11] S. Chen (ed.), Evolutionary computation in economics and finance, Heidelberg, Springer, 2002.

[12] S. Kirkpatrick, C. Gelatt and M. Vecchi, Optimization by simulated annealing, Science 220 (1983), 671–680.

[13] V. Cerny, Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm, Journal of Optimization Theory and Applications 45 (1985) 41–51.

[14] W. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller, Equation of the state calculations by fast computing machines, Journal of Chemical Physics 21 (1953), 1087–1092.

[15] E. Aarts and J. Korst, Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing, Chichester, John Wiley & Sons, 1989.

[16] E. Aarts, J. Korst and P. van Laarhoven, Simulated annealing, in: E. Aarts and J. Lenstra, Local search in combinatorial optimization, Chichester, John Wiley & Sons, 1997, 91–120.

[17] T. Chang, N. Meade, J. Beasley, Y. Sharaiha, Heuristics for cardinality constrained portfolio optimization, Computers & Operations Research (2000), 1271–1302.

[18] H. Markowitz, Portfolio Selection: Efficient diversification of investments, New York, John Wiley & Sons, 1959.

[19] G. Dueck and T. Scheurer, Threshold accepting: A general purpose algorithm appearing superior to simulated annealing, Journal of Computational Physics 90 (1990), 161–175.

[20] G. Dueck and P. Winker, New concepts and algorithms for portfolio choice, Applied Stochastic Models and Data Analysis 8 (1992), 159–178.

[21] M. Gilli and E. Kellezi, Portfolio selection, tail risk and heuristic optimization, Research paper, University of Geneva, http://www.unige.ch/ses/metri/gilli/evtrm/Aix.ps.

[22] M. Gilli and E. Kellezi, Threshold accepting for index tracking, Research paper, University of Geneva, http://www.unige.ch/ses/metri/gilli/portfolio/Yale-2001-IT.pdf.

[23] P. Winker, Optimization heuristics in econometrics, Chichester, John Wiley & Sons, 2001.

[24] F. Glover, Future paths for integer programming and links to artificial intelligence, Computers and Operations Research 13 (1986), 533–549.

[25] P. Hansen, The steepest ascent mildest descent heuristic for combinatorial programming, presented at Congress on Numerical Methods in Combinatorial Optimization, Capri, 1986.

[26] F. Glover, M. Laguna, Tabu Search, Dordrecht, Kluwer, 1997.

[27] A. Hertz, E. Taillard and D. de Werra, Tabu search, in: E. Aarts and J. Lenstra, Local search in combinatorial optimization, Chichester, John Wiley & Sons, 1997, 121–136.

[28] F. Glover, J. Mulvey and K. Hoyland, Solving dynamic stochastic control problems in finance using tabu search with variable scaling, in: H. Osman and J. Kelly (eds.), Meta-heuristics: theory and applications, Dordrecht, Kluwer, 1996, 429–448.

[29] K. DeJong, D. Fogel and H. Schwefel, A history of evolutionary computation, in: T. Baeck, D. Fogel and Z. Michalewicz (eds.), Evolutionary computation 1, Bristol, IOP Publishing, 2000, 40–58.

[30] L. Fogel, A. Owens and M. Walsh, Artificial Intelligence through simulated evolution, New York, John Wiley & Sons, 1966.

[31] J. Koza, Genetic programming, Cambridge, MIT Press, 1992.

[32] J. Koza, Genetic programming II, Cambridge, MIT Press, 1994.

[33] J. Koza, F. Bennett, D. Andre, M. Keane, Genetic programming III, San Francisco, Morgan Kaufmann, 1999.

[34] J. Holland, Adaptation in natural and artificial systems, Ann Arbor, Michigan University Press, 1975.

[35] I. Rechenberg, Cybernetic solution path of an experimental problem, Royal Aircraft Establishment Library Translation , 1965.

[36] H. Schwefel, Evolution and optimum seeking, Chichester, John Wiley & Sons, 1995.

[37] T. Baeck, D. Fogel, Z. Michalewicz (eds.), Evolutionary computation 1, Bristol, IOP Publishing, 2000.

[38] T. Baeck, D. Fogel, Z. Michalewicz (eds.), Evolutionary computation 2, Bristol, IOP Publishing, 2000.

[39] W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, R. Smith (eds.), Proc. of the Genetic and Evolutionary Computation Conference, San Francisco, Morgan Kaufmann, 1999.

[40] G. Rudolph, Finite Markov chain results in evolutionary computation: A tour d'horizon, Fundamentae Informaticae, 1998, 1–22.

[41] H. Muehlenbein, Genetic Algorithms, in: E. Aarts and J. Lenstra (eds.), Local search in combinatorial optimization, Chichester, John Wiley & Sons, 1997, 137–172.

[42] S. Droste, T. Janses and I. Wegener, Perhaps not a free lunch but at least a free appetiser, in: W. Banzaf et al. (eds.), Proceedings of First Genetic and Evolutionary Computation Conference, San Francisco, Morgan Kaufmann, 1999, 833–839.

[43] M. Vose, The simple Genetic Algorithm, Cambridge, MIT Press, 1999.

[44] I. Wegener, On the expected runtime and the success probability of Evolutionary Algorithms, Lecture Notes in Computer Science , Heidelberg, Springer, 2000.

[45] T. Riechmann, Learning in economics, Heidelberg, Physica, 2001.

[46] C. Rieck, Evoluationary simulation of asset trading strategies, in: E. Hillebrand, J. Stender (eds.): Many-agent simulation and artificial life, IOS Press, 1994, 112–136.

[47] P. Tayler, Modelling artificial stock markets using genetic algorithms, in: S. Goonatilake, P. Treleaven (eds.), Intelligent systems for finance and business, New York, John Wiley & Sons, 1995, 271–287.

[48] B. LeBaron, W. Arthur, R. Palmer, Time series properties of an artificial stock market, Journal of Economic Dynamics & Control 23 (1999), 1487–1516.

[49] J. Coche, An evolutionary approach to the examination of capital market efficiency, Evolutionary Economics 8, 357–382.

[50] J. Farmer, A. Lo, Frontiers of finance: Evolution and efficient markets, Santa Fe Institute, 1999, http://www.santafe.edu/ jdf.

[51] R. Walker, E. Haasdijk, M. Gerrets, Credit evaluation using a genetic algorithm; in: S. Goonatilake, P. Treleaven (eds.), Intelligent systems for finance and business, New York, John Wiley & Sons, 1995, 39–59.

[52] S. Mott, Insider dealing detection at the Toronto Stock Exchange Modelling artificial stock markets using genetic algorithms, in: S. Goonatilake, P. Treleaven (eds.), Intelligent systems for finance and business, New York, John Wiley & Sons, 1995, 135–144.

[53] A. Frick, R. Herrmann, M. Kreidler, A. Narr, D. Seese, A genetic based approach for the derivation of trading strategies on the German stock market, in: Proceedings ICONIP '96, Heidelberg, Springer, 1996, 766–770.

[54] R. Bauer, Genetic algorithms and investment strategies, New York, John Wiley & Sons, 1994.

[55] J. Kingdon, Intelligent systems and financial forecasting, Heidelberg, Springer, 1997.

[56] R. Tsang, P. Lajbcygier, Optimization of technical trading strategy using split search Genetic Algorithms, in: Y. Abu-Mostafa, B. LeBaron, A. Lo, A. Weigend (eds.), Computational finance 1999, Cambridge, MIT Press, 2000, 690–703.

[57] W. Langdon, R. Poli, Foundations of genetic programming, Heidelberg, Springer, 2002.

[58] C. Keber, Option valuation with the Genetic Programming approach, in: Y. Abu-Mostafa, B. LeBaron, A. Lo, A. Weigend, Computational finance 1999, Cambridge, MIT Press, 2000, 370–386.

[59] M. Brennan, E. Schwarz, The valuation of American put options, Journal of Finance 32 (1977), 449–462.

[60] J. Cox, S. Ross, M. Rubinstein, Option pricing: a simplified approach, Journal of Financial Economics 7 (1979), 229–263.

[61] J. Li and E. Tsang, Reducing failures in investment recommendations using Genetic Programming, presented at th Conference on Computing in Economics and Finance, Barcelona, 2000.

[62] S. Baglioni, C. da Costa Pereira, D. Sorbello and A. Tettamanzi, An evolutionary approach to multiperiod asset allocation, in: R. Poli, W. Banzhaf, W. Langdon, J. Miller, P. Nordin and T. Fogarty (eds.), Genetic Programming, Proceedings of EuroGP 2000, Heidelberg, Springer, 2000, 225–236.

[63] S. Chen, T. Kuo, Towards an agent-based foundation of financial econometrics: An approach based on Genetic-Programming financial markets, in: W. Banzhaf et al. (eds.), Proc. of the Genetic and Evolutionary Computation Conference, San Francisco, Morgan Kaufmann, 1999, 966–973.

[64] W. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics (1943), 115–133

[65] R. Schalkoff, Artificial Neural Networks, New York, McGraw-Hill, 1997.

[66] M. Arbib, The handbook of brain theory and Neural Networks, Cambridge, MA, MIT Press, 1995.

[67] T. Kohonen, Self-organising maps, Heidelberg, Springer, 1995.

[68] G. Deboeck and T. Kohonen, Visual explorations in finance, Heidelberg, Springer, 1998.

[69] U. Seiffert, L. Jain (eds.), Self-organising neural networks, Heidelberg, Springer, 2002.

[70] D. Rumelhart, J. McClelland, Parallel distributed processing: Explorations in the microstructure of cognition, Vol. 1: Foundations, Cambridge, MA, MIT Press, 1986.

[71] R. Hecht-Nielsen, Neurocomputing, Reading, MA, Addison-Wesley, 1990.

[72] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Networks (1991), 251–257.

[73] M. Anthony, P. Bartlett, Learning in neural networks, Cambridge, UK, University Press, 1999.

[74] A. Zapranis, P. Refenes, Priciples of neural model identification, London, Springer, 1999.

[75] D. Witkowska, Neural Networks application to analysis of daily stock returns at the largest stock markets, in: P. Szczepaniak (ed.), Computational intelligence and applications, Heidelberg, Physica, 1999, 351–364.

[76] M. Azoff, Neural Network time series forecasting of financial markets, New York, John Wiley & Sons, 1994.

[77] R. Bharati, V. Desai, M. Gupta, Predicting real estate returns using Neural Networks, Journal of Computational Intelligence in Finance (1999) 1, 5–15.

[78] J. Baetge, A. Jerschensky, Measurement of the probability of insolvency with Mixture-of-Expert Networks, in: W. Gaul, H. Locarek-Junge (eds.), Classification in the information age, Heidelberg, Springer, 1999, 421–429.

[79] A. Refenes, Neural Networks in the capital markets, Chichester, John Wiley & Sons, 1995.

[80] R. Trippi and E. Turban, Neural Networks in Finance and Investing, Chicago, Probus Publishing, 1993.

[81] M. Odom, R. Sharda, A Neural Network model for bankruptcy prediction, Proceedings of the IEEE International Joint Conference an Neural Networks, Vol. 2, 1990, 163–167.

[82] K. Coleman, T. Graettinger and W. Lawrence, Neural Networks for bankruptcy prediction: The power to solve financial problems, in: AI review (1991) 4, 48–50.

[83] R. McLeod, D. Malhotra and R. Malhotra, Predicting credit risk, A Neural Network Approach, Journal of Retail Banking (1993) 3, 37–44.

[84] J. Baetge and C. Krause, The classification of companies by means of Neural Networks, Journal of Information Science and Technology (1993) 1, 96–112.

[85] R. Wilson, R. Sharda, Bankruptcy prediction using Neural Networks, Decision Support Systems (1994), 545–557.

[86] E. Altman, Financial ratios, discriminant analysis and the prediction of corporate bankruptcy, Journal of Finance (1968), 189–209.

[87] E. Altman, G. Marco and F. Varetto, Corporate distress diagnosis: Comparisions using linear discriminant analysis and Neural Networks, Journal of Banking and Finance (1994) 3, 505–529.

[88] A. Beltratti, S. Margarita and P. Terna, Neural Networks for economic and financial modelling, London, International Thomson Computer Press, 1994.

[89] M. Malliaris and L. Salchenberger, Beating the best: A Neural Network challenges the Black-Scholes formula, Applied Intelligence (1993) 3, 193–206.

[90] J. Hutchinson, A. Lo and T. Poggio, A nonparametric approach to pricing and hedging derivative securities, Journal of Finance (1994) 3, 851–889.

[91] P. Lajbcygier, A. Flitman, A. Swan and R. Hyndman, The pricing and trading of options using a hybrid Neural Network model with historical volatility, NeuroVest Journal (1997) 1, 27–41.

[92] M. Hanke, Neural Network approximation of analytically intractable option pricing models, Journal of Computational Intelligence in Finance (1997) 5, 20–27.

[93] R. Herrmann, A. Narr, Risk neutrality, Risk (1997) 8.

[94] P. Lajbcygier, Literature review: The non-parametric models, Journal of Computational Intelligence in Finance (1999) 6, 6–18.

[95] F. Black, M. Scholes, The valuation of option contracts and a test of market efficiency, Journal of Finance (1972), 399–417.

[96] F. Black, M. Scholes, The pricing of options and corporate liabilities, Journal of Political Economy (1973), 637–654.

[97] H. Locarek-Junge and R. Prinzler, Estimating Value-at-Risk using Artificial Neural Networks, in: C. Weinhardt, H. Meyer zu Selhausen and M. Morlock (eds.), Informationssysteme in der Finanzwirtschaft, Heidelberg, Springer, 1998, 385–399.

[98] C. Bishop, Neural Networks for pattern recognition, Oxford, Clarendon Press, 1995.

[99] P. Jorion, Value-at-Risk: The new benchmark for controlling market risk, Chicago, Irwin, 1997.

[100] J. P. Morgan and Reuters, RiskMetrics1exTM Technical Document, New York, 1996, http://www.rmg.com.

[101] P. Naim, P. Herve and H. Zimmermann, Advanced adaptive architectures for asset allocation, in: C. Dunis (ed.), Advances in quantitative asset management, Norwell, Kluwer Academic Publishers, 2000, 89–112.

[102] M. Bonilla, P. Marco, I. Olmeda, Forecasting exchange rate volatilities using Artificial Neural Networks, in: M. Bonilla, T. Casasus and R. Sala, Financial Modelling, Heidelberg, Physica, 2000, 57–68.

[103] C. Alexander, Volatility and correlation: Measurement, models and applications, in: C. Alexander (ed.), Risk management and analysis, Vol. 1: Measuring and modelling financial risk, New York, John Wiley & Sons, 1998, 125–171.

[104] S. Judd, Time complexity of learning, in: M. Arbib (ed.), Handbook of brain theory and Neural Networks, Cambridge, MA, MIT Press, 1995, 984–990.

[105] L. Zadeh, Fuzzy sets, Information and Control , (1965) 338–352.

[106] L. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, IEEE Transactions on Systems, Man and Cybernetics, SMC-3 (1973) 1, 28–44.

[107] G. Klir and B. Yuan, Fuzzy sets and fuzzy logic: Theory and applications, Upper Saddle River, Prentice-Hall, 1995.

[108] G. Klir and B. Yuan (eds.), Fuzzy sets, fuzzy logic and fuzzy systems, Singapore, World Scientific, 1995.

[109] L. Wang, Fuzzy systems are universal approximators, in: Proceedings of the First IEEE International Conference on Fuzzy Systems, San Diego, 1992, 1163–1169.

[110] B. Kosko, Fuzzy systems as universal approximators, in: IEEE Transactions on Computers, (1994) 9, 1329–1333.

[111] C. von Altrock, Fuzzy logic and NeuroFuzzy applications in business and finance, Upper Saddle River, Prentice-Hall, 1997.

[112] H. Rommelfanger, Fuzzy logic based systems for checking credit solvency of small business firms, in: R. Ribeiro, H.-J. Zimmermann, R. Yager and J. Kacprzyk (eds.), Soft computing in financial engineering, Heidelberg, Physica, 1999, 371–387.

[113] R. Weber, Applications of Fuzzy logic for credit worthiness evaluation, in: R. Ribeiro, H.-J. Zimmermann, R. Yager and J. Kacprzyk (eds.), Soft computing in financial engineering, Heidelberg, Physica, 1999, 388–401.

[114] D. Ruan, J. Kacprzyk, M. Fedrizzi, Soft computing for risk evaluation and management, Heidelberg, Physica, 2001, 375–409.

[115] K. Korolev, K. Leifert and H. Rommelfanger, Fuzzy logic based risk management in financial intermediation, in: D. Ruan, J. Kacprzyk, M. Fedrizzi, Soft computing for risk evaluation and management, Heidelberg, Physica, 2001, 447–471.

[116] R. Merton, An analytic derivation of the cost of deposit insurance and loan guarantees, Journal of Banking in Finance (1977) 1, 3–11.

[117] S. Goonatilake, S. Khebbal (eds.), Intelligent hybrid systems, Chichester, John Wiley & Sons, 1995.

[118] A. Abraham, M. Koeppen (eds.), Hybrid information systems, Heidelberg, Springer, 2002.

[119] D. Rutkowska, Neuro-fuzzy architectures and hybrid learning, Heidelberg, Springer, 2002.

[120] Y. Jin, Advanced fuzzy systems design and applications, Heidelberg, Springer, 2002.

[121] J. Balicki, Evolutionary Neural Networks for solving multiobjective optimization problems, in: P. Szczepaniak, Computational intelligence and applications, Heidelberg, Springer, 1999, 108–199.

[122] M. Gupta, Fuzzy neural computing, in: P. Szczepaniak, Computational intelligence and applications, Heidelberg, Springer, 1999, 34–41.

[123] R. Herrmann, M. Kreidler, D. Seese and K. Zabel, A fuzzy-hybrid approach to stock trading, in: S. Usui, T. Omori (eds.), Proceedings ICONIP '98, Amsterdam, IOS Press, 1998, 1028–1032.

[124] S. Siekmann, R. Neuneier, H.-J. Zimmermann and R. Kruse, Neuro-Fuzzy methods applied to the German stock index DAX, in: R. Ribeiro, H.-J. Zimmermann, R. Yager and J. Kacprzyk (eds.), Soft computing in financial engineering, Heidelberg, Physica, 1999, 186–203.

[125] Z. Harland, Using nonlinear Neurogenetic models with profit related objective functions to trade the US T-Bond future, in: Y. Abu-Mostafa, B. LeBaron, A. Lo, A. Weigend (eds.), Computational finance 1999, Cambridge, MIT Press, 2000, 327–343.

[126] F. Schlottmann, D. Seese, A hybrid genetic-quantitative method for risk-return optimization of credit portfolios, Proc. QMF'2001 (abstracts), Sydney, 2001, http://www.business.uts.edu.au/resources/qmf2001/F_Schlottmann.pdf.

[127] F. Schlottmann, D. Seese, Finding Constrained Downside Risk-Return Efficient Credit Portfolio Structures Using Hybrid Multi-Objective Evolutionary Computation, presented at Computing in Economics and Finance Conference 2002, Aix-en-Provence, 2002, http://www.cepremap.ens.fr/sce2002/papers/paper78.pdf.

GILLARDON – innovative Lösungen für die Finanzwirtschaft

Die Lösungen

Unsere Kernkompetenzen umfassen die Bereiche Kundenberatung, Produktkalkulation und Gesamtbanksteuerung.

Kundenberatung

evenit™ ist das themenorientierte Beratungssystem für alle Vertriebskanäle für die Themen Altersvorsorge, Baufinanzierung, Vermögensanalyse und Financial Planning.

Produktkalkulation

MARZIPAN™ ist die Lösung zur Produktberatung und -kalkulation von Aktiv- und Passivgeschäften auf Basis der Marktzins- und Barwertmethode.

FinanceFactory™ ist das regelbasierte Kalkulationssystem für die Absatzfinanzierung, das alle Darlehensvarianten der Absatzfinanzierung inklusive Restkreditversicherung und Subventionsrechnung abdeckt.

Gesamtbanksteuerung

THINC™ ist die integrierte Softwarelösung zur wertorientierten Gesamtbanksteuerung und deckt die Themen Markt- und Vertriebssteuerung, Bilanzstrukturmanagement, Risikocontrolling, Treasury, Adressrisikosteuerung, Basel II und IAS / IFRS ab. THINC unterstützt Sie bei der Erfüllung der Anforderungen aus den MaRisk.

GILLARDON ist Branchenspezialist für Softwarelösungen, Consulting und Seminare in den Themenbereichen Kundenberatung, Produktkalkulation und Gesamtbanksteuerung.

GILLARDON – ein Unternehmen der msg systems ag