8

A Memetic-Neural Approach to Discover Resources in P2P Networks

Ferrante Neri, Niko Kotilainen, and Mikko Vapa

Department of Mathematical Information Technology, Agora, University of Jyväskylä, P.O. Box 35 (Agora), FI-40014 University of Jyväskylä, Finland
neferran@cc.jyu.fi, niko.kotilainen@jyu.fi, mikko.vapa@jyu.fi

Summary. This chapter proposes a neural network based approach for solving the resource discovery problem in Peer to Peer (P2P) networks and an Adaptive Global Local Memetic Algorithm (AGLMA) for performing in training of the neural network. The neural network, which is a multi-layer perceptron neural network, allows the P2P nodes to efficiently locate resources desired by the user. The necessity of testing the network in various working conditions, aiming to obtain a robust neural network, introduces noise in the objective function. The AGLMA is a memetic algorithm which employs two local search algorithms adaptively activated by an evolutionary framework. These local searchers, having different features according to the exploration logic and the pivot rule, have the role of exploring decision space from different and complementary perspectives. Furthermore, the AGLMA makes an adaptive noise compensation by means of explicit averaging on the fitness values and a dynamic population sizing which aims to follow the necessity of the optimization process. The numerical results demonstrate that the proposed computational intelligence approach leads to an efficient resource discovery strategy and that the AGLMA outperforms an algorithm classically employed for executing the neural network training.

Keywords: Memetic Algorithms, Neural Networks, P2P Networks, Telecommunication, Noisy Optimization Problems.

8.1 Introduction

During recent years the use of peer-to-peer networks (P2P) has significantly increased. P2P networks are widely used to share files or communicate with each other using Voice over Peer-to-Peer (VoP2P) systems, for example Skype. Due to the large number of users and large files being shared communication load induced to the underlying routers is enormous and thus demand of high performance in peer-to-peer networks is constantly growing.

In order to obtain a proper functioning of a peer-to-peer network a crucial point is to efficiently execute the peer-to-peer resource discovery, meaning the search of information (files, users, devices etc.) within a network of computers connected by Internet. An improper resource discovery mechanism would lead to overwhelming query traffic within the P2P network and consequently to a waste of bandwidth of each single user connected to the network.

C. Cotta and J. van Hemert (Eds.): Recent Advances in Evol. Comp., SCI 153, pp. 113–129, 2008.
springerlink.com

Although several proposals are present in commercial packages (e.g., Gnutella [151] and KaZaA), this problem is still intensively studied in literature. Resource discovery strategies can be divided into two classes: breadth-first search and depth-first search. Breadth-First Search (BFS) strategies forward a query to multiple neighbors at the same time whereas Depth-First Search (DFS) strategies forward only to one neighbor. In both strategies, the choice of those neighbors receiving the query is carried out by heuristic methods. These heuristics might be stochastic e.g. random selection [152], or based on deterministic rules [153].

BFS strategies have been used in Gnutella [151], where the query is forwarded to all neighbors and the forwarding is controlled by a time-to-live parameter. This parameter is defined as the amount of hops required to forward the query. Two nodes are said to be n hops apart if the shortest path between them has length n [153]. The main disadvantage of the Gnutella's mechanism is that it generates a massive traffic of query messages when the time-to-live parameter is high thus leading to a consumption of an unacceptable amount of bandwidth.

In order to reduce query traffic, Lv et al. [152] proposed the *Expanding Ring*. This strategy establishes that the time-to-live parameter is gradually increased until enough resources have been found. Although use of the *Expanding Ring* is beneficial in terms of query reduction, it introduces some delay to resource discovery and thus implies a longer waiting time for the user. Kalogeraki et al. [154] proposed a Modified Random Breadth-First Search (MRBFS) as an enhancement of the Gnutella's algorithm. In MRBFS, only a subset of neighbors are selected randomly for forwarding. They also proposed an intelligent search mechanism which stores the performance of the queries previously done for each neighbor. This memory storage is then used to direct the subsequent queries. Following the ideas of Kalogeraki et al., Menascé [155] proposed that only a subset of neighbors are randomly selected for forwarding. Yang and Garcia-Molina [153] proposed the Directed BFS (DBFS), which selects the first neighbor based on one of several heuristics and further uses BFS for forwarding the query. They also proposed the use of local indices for replicating resources to a certain radius of hops from a node. In Gnutella2 [156] a trial query is sent to the neighbors and, on the basis of obtained results, an estimate of how widely the actual query should be forwarded is calculated.

In the DFS strategies, selection of the neighbor chosen for the query forwarding is performed by means of heuristics. The main problem related to use of this strategy is the proper choice of this heuristic. A popular heuristic employed with this aim is the random walker which selects the neighbor randomly. The random walker terminates when a predefined number of hops have been travelled or when enough resources have been found. Lv et al. [152] studied the use of multiple random walkers which periodically check the query originator in order to verify if the query should still be forwarded further. Tsoumakos and Roussopoulos [157] proposed an Adaptive Probabilistic Search (APS). The APS makes use of feedback from previous queries in order to tune probabilities for further forwarding of random walkers. Crespo and Garcia-Molina [158] proposed the routing indices, which provide shortcuts for random walkers in locating

resources. Sarshar et al. [159] proposed the Percolation Search Algorithm (PSA) for power-law networks. The idea is to replicate a copy of resources to a sufficient number of nodes and thus ensure that resource discovery algorithm locates at least one replica of the resource.

The main limitation of the previous studies, for both BFS and DFS strategies, is that all the approaches are restricted to only one search strategy. On the contrary, for the same P2P network, in some conditions it is preferable to employ both BFS and DFS strategies. In order to obtain a flexible search strategy, which intelligently takes into account the working conditions of the P2P network, Vapa et al. [160] proposed a neural network based approach (NeuroSearch). This strategy combines multiple heuristics as inputs of a neural network in order to classify among all its neighbors those which will receive the query, thus it does not fix a priori the search strategy (breadth-first or depth-first) to be employed. Depending on the working conditions of the P2P network, NeuroSearch can alternate between both search strategies during a single query.

Since NeuroSearch is based on a neural network, it obviously follows that an initial training is needed. The resulting optimization problem is very challenging because neural networks have a large number of weights varying from minus to plus infinity. In addition, in order to obtain a robust search strategy it is required that training is performed in various working conditions of a P2P network. It is therefore required that many queries are executed, thus making the training problem computationally expensive and the optimization environment noisy.

8.2 NeuroSearch - Neural Network Based Query Forwarding

As highlighted above, NeuroSearch [160] is a neural network-based approach for solving the resource discovery problem. NeuroSearch combines different local information units together as an input to multi-layer perceptron (MLP) neural network [161]. Multi-layer perceptron is a non-linear function approximator, which is organized into different layers: an input layer, one or more hidden layers and an output layer. Adjacent layers are connected together with weights, these weights are the parameters of the function approximator to be determined by the learning process. Hidden and output layers contain neurons, which take a weighted sum of outputs from the previous layer and use a non-linear transfer function to produce output to the next layer. NeuroSearch uses two hidden layers, both having 10 neurons and two different transfer functions in hidden and output layers. The structure of this neural network has been selected on the basis of previous studies carried out by means of the P2PRealm simulation framework [162].

We characterize the query forwarding situation with a model consisting of 1)the previous forwarding node, 2)the currently forwarding node and 3)the receiver of the currently forwarding node. Upon receiving a query, the currently forwarding node selects the first of its neighbors and determines the inputs,

related to that neighbor, of the neural network. Then, the neural network output is calculated. This output establishes whether or not the query will be forwarded to the neighbor. Next, all other neighbors including the previous forwarding node, are processed in a similar manner by means of the same neural network. If some of the neighbors were forwarded, then new query forwarding situations will occur until all forwarding nodes have decided not to forward query further.

Fig. 8.1 shows the functioning of a P2P network with neural network based forwarding.

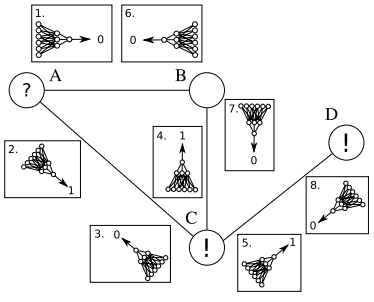

Fig. 8.1. Query Forwarding

The circles shown in the figure represent the peers of the P2P network. The arcs between the peers represent the Transmission Control Protocol (TCP) communication links between the peers. The rectangles represent a neural network evaluation for different neighbors. More specifically, node A denoted with a question mark begins a query. It attempts to forward the query to node B. The neural network in rectangle 1. outputs zero and therefore the query is not forwarded. Instead the second evaluation for node C, shown in rectangle 2, outputs one and the query is forwarded to node C. Then node C attempts to forward the query to neighbor nodes and the nodes B and D receives the query. In the last steps nodes B and D do not forward the query further and the query ends. The query enters nodes C and D denoted with an exclamation mark thereby locating two resource instances.

8.2.1 The Set of the Neural Network Inputs

The MLP uses constant, binary and discrete valued inputs as an information for making forwarding decisions. Each input I_j is a neuron and all 22 inputs I form the input layer.

The following input is constant:

(1) *Bias* takes value 1. Bias is needed in MLP neural networks to approximate functions with non-zero output in case of zero input.

The following inputs are binary:

(2) *Sent* scores 1 if the query has already been forwarded to the receiver. Otherwise it scores 0.
(3) *CurrentVisited* scores 1 if the query has already been received by the currently forwarding node, else it scores 0.
(4) *From* is a binary variable indicating whether a query was received from this receiver. *From* scores 1 if the current query was received from this receiver. Otherwise it scores 0.
(5) *RandomNeighbor* scores 1 for a randomly selected receiver and 0 for other receivers in the current node.
(6) *EnoughReplies* scores 1, if through the query path used by the current query an equal number or more resources have been found as were given in RepliesToGet input parameter (see below). Otherwise *EnoughReplies* scores 0.
The following inputs are discrete:
(7) *Hops* is the number of edges the query has travelled in a P2P network (see definition of *Hops* in section 8.1).
(8) *ToNeighbors* is the number of neighbors connected to the receiver.
(9) *CurrentNeighbors* is the number of neighbors connected to the currently forwarding node.
(10) *FromNeighbors* is the number of neighbors connected to the previous forwarding node.
(11) *InitiatorNeighbors* is the number of neighbors connected to the query initiator.
(12) *NeighborsOrder* is a number associated to each neighbor connected to the forwarding peer. The *NeighborsOrder* is assigned by ascent sorting and enumerating $(0, 1, 2...)$ the neighbors according to their degree. By degree of a peer node we mean the number of neighbors connected to it.
(13) *FromNeighborsOrder*, indicates the *NeighborsOrder* of the previous forwarding node.
(14) *RepliesNow* is the number of replies the query locates in its query path.
(15) *PacketsNow* is the number of packets the query produces in its query path.
(16) *RepliesToGet* is the number of resources that need to be located.
(17) *Forwarded* is the number of times the currently forwarding node has forwarded the query.
(18) *NotForwarded* is the number of times the current node did not forward the query.
(19) *DecisionsLeft* is the number of forwarding "decisions" the current node will still make for the current query message i.e. how many neighbors have not yet been evaluated for forwarding the query message.
(20) *SentCounter* is the number of times the current query has already been forwarded to the receiver.
(21) *CurrentVisitedCounter* is the number of times the query has already been received by the currently forwarding node.
(22) *BranchingResourcesMissing* estimates how many resources on average should still be located from the current query path. First the estimate is set

to the value of *RepliesToGet*. The estimate is updated each time the current node has made all the forwarding decisions. If the current node contained the queried resource, the value is decreased by one. The estimate is then updated depending on whether the current value is positive or negative. In case of a positive value, the current value is divided with the number of neighbors receiving the query. In case of a negative value, the current value is multiplied by the number of neighbors, which will receive the query.

8.2.2 Input Scaling

To ensure that all inputs are in the range of $[0, 1]$ the discrete inputs need to be scaled. The discrete inputs can be classified into three categories according to their original range of variability.

(a) Inputs in the range of $[0, \infty]$ are *Hops*, *NeighborsOrder*, *FromNeighborsOrder*, *RepliesNow*, *PacketsNow*, *RepliesToGet*, *Forwarded*, *NotForwarded*, *DecisionsLeft*, *SentCounter* and *CurrentVisitedCounter* and they are scaled with the function $s(I_j) = \frac{1}{I_j+1}$
(b)Inputs in the range of $[1, \infty]$ are *ToNeighbors*, *CurrentNeighbors*, *FromNeighbors* and *InitiatorNeighbors* and they are scaled with $s(I_j) = \frac{1}{I_j}$
(c)*BranchingResourcesMissing* is in the range of $[-\infty, \infty]$ and it is scaled with the sigmoid function $s(I_{22}) = \frac{1}{1+e^{-I_{22}}}$

The scaled inputs I are then given to the neural network.

8.2.3 Calculation of the Neural Network Output

The neurons on the hidden layers contain the transfer function $t(a) = \frac{2}{1+e^{-2a}} - 1$ where a is the sum of the outcoming weighted outputs from the previous (input or first hidden) layer and *Bias*.

The output layer neuron contains a transfer function

$$u(a) = \begin{cases} 0 \text{ if } a < 0 \\ 1 \text{ if } a \geq 0 \end{cases} \tag{8.1}$$

where a is the sum of the outcoming weighted outputs from the second hidden layer. The output function is thus defined as follows:

$$O = f(I) = u\left(w_{3,1} + \sum_{l=2}^{L} w_{3,l} t\left(w_{2,1,l} + \sum_{k=2}^{K} w_{2,k,l} t\left(\sum_{j=1}^{J} w_{1,j,k} s_j(I_j)\right)\right)\right) \tag{8.2}$$

where J is the number of inputs, K is the number of neurons on the second layer, L is the number of neurons on the third layer, $w_{1,j,k}$ is the weight from the j^{th} input to k^{th} neuron on the first hidden layer, $w_{2,k,l}$ is the weight from the k^{th} neuron on the first hidden layer to l^{th} neuron on the second hidden layer and $w_{3,l}$ is the weight from the l^{th} neuron on the second hidden layer to the output

neuron, $w_{2,1,l}$ is the bias weight associated to the second hidden layer and $w_{3,1}$ is the bias weight associated to the output layer.

Output O can take a boolean value indicating whether the query is forwarded to the neighbor node currently being taken into consideration. The neural network output is calculated separately for each neighbor node and after the calculations, the query is sent to neighbor nodes which had an output value 1.

8.3 The Optimization Problem

The neural network described above is supposed to handle the communication and data transfer between a couple of peers. As in all cases of the neural networks, its proper functioning is subject to correctly executed training. The training of a neural network consists of determination of the set of weight coefficients W. As shown in formula (8.2), the weights can be divided into three categories on the basis of the layer to which they belong to. There are 22 input neurons and 10 neurons on both the hidden layers. Since one input is constant (*Bias*) the total amount of weights is $22 * 9 + 10 * 9 + 10 = 298$. The weights can take values in the range $[-\infty, \infty]$.

8.3.1 Fitness Formulation

In order to estimate the quality of a candidate solution, the performance of the P2P network is analyzed with the aid of a simulator whose working principles are described in [162]. More specifically, the set of weights is given to the simulator and a certain number n of queries are performed. The total fitness over the n queries is given by:

$$F = \sum_{j=1}^{n} F_j(W) \tag{8.3}$$

where F_j is the fitness contribution from each query. It is important to remark that multiple queries are needed in order to ensure that the neural network is robust in different query conditions. In addition, for each query the amount of Available Resources (AR) instances is known. Thus, AR is a constant value given a priori.

For each query, the simulator returns two outputs:

(a) the number of query packets P used in the query
(b) the number of found resource instances R during the query

For details regarding the simulation see [162]. These outputs are combined in the following way in order to determine each F_j:

$$F_j = \begin{cases} 0 & \text{if } P > 300 \\ 1 - \frac{1}{P+1} & \text{if } P \leq 300 \text{ AND } \quad R = 0 \\ 50 * R - P & \text{if } P \leq 300 \text{ AND } 0 < R < \frac{AR}{2} \\ 50 * \frac{AR}{2} - P & \text{if } P \leq 300 \text{ AND } \quad \frac{AR}{2} < R \end{cases} \tag{8.4}$$

In formula (8.4), the constant values 300 and 50 have been set according to the criterion explained in [160].

The first condition in (8.4) ensures that the neural network should eventually stop forwarding the queries. The second condition controls that if no resources are found then the neural network increases the number of query packets sent to the network. The third condition states that if the number of found resources is not enough then the neural network develops only by locating more resources. The fourth condition ensures that when half of the available resource instances are found from the network the fitness grows if the neural network uses fewer query packets. The fourth condition also upperbounds F_j to $50 * \frac{AR}{2} - \frac{AR}{2} = 49 * \frac{AR}{2}$, because it is imposed that when half of the resource instances are found P is at minimum $\frac{AR}{2}$.

Thus, the problem of discovering resources in the P2P network consists of the maximization of F in a 298-dimension continuous space:

$$\max\left(F\left(W\right)\right) \text{in} \left[-\infty, \infty\right]^{298} \tag{8.5}$$

Due to the necessity of ensuring robustness of the neural network in different queries, the fitness value varies with the chosen query. The querying peer and the queried resource need to be changed to ensure that the neural network is not just specialized for searching resources from one part of the network or one particular resource alone. Since n (in our case $n = 10$) queries are required and they are chosen at random, fitness F is noisy. This noise does not have any peculiarity and therefore it can hardly be approximated by a known distribution function. Let us indicate with PN the distribution of this noise and thus re-formulate the problem in equation (8.5)

$$\max\left(F\left(W\right) + Z\right) \text{in} \left[-\infty, \infty\right]^{298}; Z \sim PN \tag{8.6}$$

8.3.2 Features of the Decision Space and the Fitness Landscape

As highlighted above, the optimization problem is highly multivariate and is defined in a continuous domain. It obviously follows that the problem is quite challenging due to a high dimensionality. In addition, presence of the noise enhances the difficulty of the problem because it introduces some "false" optima into the landscape which disturb the functioning of any optimization algorithm [163,164].

Due to the structure of each F_j (see equation (8.4)), the fitness landscape contains discontinuities. In particular, it is relevant to observe that due to the first condition in (8.4) the fitness landscape contains some plateaus with a null value as well as some other areas which take non-null values and contain a variability. In order to give a rough description of the fitness landscape, the following test has been designed. 2 million candidate solutions have been pseudo-randomly sampled by means of a uniform distribution within the decision space. Fig. 8.2 and 8.3 show the histogram and distribution curve, respectively, related to this test. It should be noted that the y-axis has a logarithmic scale. Fig. 8.2 shows that about half the points take a null fitness value and Fig. 8.3 shows that

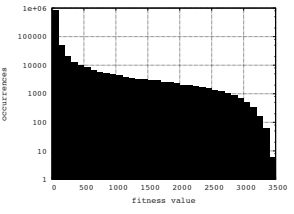

Fig. 8.2. Histogram of Fitness Values

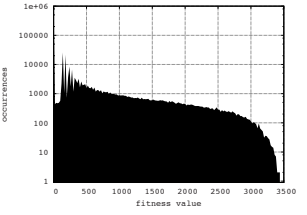

Fig. 8.3. Distribution of Fitness Values

the distribution curve contains a very high, sharp peak in zero and other lower sharp peaks before 500. This obviously means the fitness landscape contains some plateau areas for low fitness values (up to 500) and a variational area for high fitness values. In other words the fitness landscape is locally flat and contains several areas having a small variation in fitness values [165, 166]. This feature of the fitness landscape makes the optimization problem very challenging since many optimization algorithms can easily stagnate or prematurely converge in a suboptimal plateau.

8.4 The Adaptive Global-Local Memetic Algorithm

In order to solve the problem in (8.5), an Adaptive Global-Local Memetic Algorithm (AGLMA) has been implemented.

8.4.1 Initialization

An initial sampling made up of S_{pop}^{i} individual has been executed pseudo-randomly with a uniform distribution function over the interval $[-0.2, 0.2]$. This choice can be briefly justified in the following way. The weights of the initial set of neural networks must be small and comparable among each other in order to avoid one or a few weights dominating with respect to the others as suggested in [167, 168].

8.4.2 Parent Selection and Variation Operators

All individuals of the population S_{pop} undergo recombination and each parent generates an offspring. The variation occurs as follows. For each candidate solution i is associated a self-adaptive vector h_i which represents a scale factor for the exploration. More specifically, at the first generation the self-adaptive vectors h_i are pseudo-randomly generated with uniform distribution within $[-0.2, 0.2]$.

At the subsequent generations each self-adaptive vector is updated according to [167, 168]:

$$h_i^{k+1}(j) = h_i^k(j)\, e^{(\tau N_j(0,1))} \quad \text{for} \quad j = 1, 2...n \tag{8.7}$$

where k is the index of generation, j is the index of variable ($n = 298$), $N_j(0,1)$ is a Gaussian random variable and $\tau = \frac{1}{\sqrt{2\sqrt{n}}} = 0.1659$. Each corresponding

candidate solution W_i is then perturbed according to the following formula [167, 168]:

$$W_i^{k+1}(j) = W_i^k + h_i^{k+1}(j) N_j(0,1) \quad \text{for} \quad j = 1, 2...n \tag{8.8}$$

It is interesting to observe that each component $h_i^k(j)$ of the self-adaptive vector at the k^{th} generation can be seen as the standard deviation of a Gaussian perturbation.

8.4.3 Fitness Function

In order to take into account the noise, function F is calculated n_s times (where n_s stands for number of samples) and an *Explicit Averaging* technique is applied [164, 169]. More specifically each set of weights of a neural network (candidate solution) is evaluated by means of the following formula:

$$\hat{F} = F_{mean}^i - \frac{\sigma^i}{\sqrt{n_s}} \tag{8.9}$$

where F_{mean}^i and σ^i are respectively the mean value and standard deviation related to the n_s samples performed to the i^{th} candidate solution.

The penalty term $\frac{\sigma^i}{\sqrt{n_s}}$ takes into account the distribution of the data and the number of performed samples [170]. Since the noise strictly depends on the solution under consideration, it follows that for some solutions the value of σ^i is relatively small (stable solutions) and so the penalization is small. On the other hand, other solutions could be unstable and score 0 during some samples and a high performance value during other samples. In these cases σ^i is quite large and the penalization must be significant.

8.4.4 Local Searchers

Two local search algorithms with different features in terms of search logic and pivot rule have been employed. These local searchers have the role of supporting the evolutionary framework, offering new search directions and exploiting the available genotypes [171, 172]. The **Simulated Annealing** (SA) metaheuristic [173], [174] has been chosen since it offers an exploratory perspective in the decision space which can choose a search direction leading to a basin of attraction different from where starting point W_0 is. The exploration is performed by using the same mutation scheme as was described in equations (8.7) and (8.8) for an initial self-adaptive vector h_0 pseudo-randomly sampled in $[-0.2, 0.2]$. The main reason for employing the SA in the AGLMA is that the evolutionary framework should be assisted in finding better solutions which improve the available genotype while at the same time exploring areas of the decision space not yet explored. It accepts with a certain probability solutions with worse performance in order to obtain a global enhancement in a more promising basin of attraction. In addition, the exploratory logic aims to overcome discontinuities of the fitness landscape and to "jump" into a plateau having better performance. For these reasons the SA has been employed as a "global" local searcher.

The application of the SA local searcher can be successful in most cases, in the early generations, and in the late generations as well. Moreover, due to its structure the SA can efficiently offer solutions in unexplored basins of attractions and, therefore, prevent an undesired premature convergence. The most delicate issue related to the SA is choice of parameters. The SA has two parameters which are the budget and the initial temperature $Temp^0$. The budget has been fixed at 600 fitness evaluations (in order to have a constant computational cost for the SA). The setting of the initial temperature $Temp^0$ is performed as explained in section 8.4.5. The temperature $Temp$ is reduced according to a hyperbolic law following the suggestions in [175].

The **Hooke-Jeeves Algorithm** (HJA) [176, 177] is a deterministic local searcher which has a steepest descent pivot rule. Briefly the implemented HJA consists of the following. An initial radius d_0 (in our implementation $d_0 = 0.5$) an initial candidate solution W_0 and a direction exploratory matrix are required. In this implementation a standard identity matrix I has been chosen due to the hypercubic features of the decision space. Let us indicate with $I(m,:)$ the m^{th} row of the direction matrix with $m = 1, 2..n$ ($n = 298$).

The HJA consists of an exploratory move and a pattern move. Indicating with W_{cb} the current best candidate solution and with d the generic radius of the search, the HJA during the exploratory move samples the points $W_{cb}(m)+dI(m,:)$ with $m = 1, 2..n$ and the points $W_{cb}(m) - dI(m,:)$ with $m = 1, 2..n$ only along those directions which turned out unsuccessful during the "+" move. Then, if a new current best is found W_{cb} is updated and the pattern move is executed. If a new current best is not found, d is halved and the exploration is repeated.

The HJA pattern move is an aggressive attempt of the algorithm which aims to exploit promising search directions. Rather than centering the following exploration at the most promising explored candidate solution (W_{cb}), the HJA tries to move further [178]. The algorithm centers the subsequent exploratory move at $W_{cb} \pm dI(m,:)$ ("+" or "-" on the basis of the best direction). If this second exploratory move does not outperform $\hat{F}(W_{cb})$ (the exploratory move fails), then an exploratory move with W_{cb} as the center is performed. The HJA stops either when $d < 0.01$ or when the budget condition of 1000 fitness evaluation is reached.

The HJA is supposed to efficiently exploit promising solutions enhancing their genotype in a meta-Lamarckian logic and thus assist the evolutionary framework in quickly climbing the basin of attractions. In this sense the HJA can be considered as a kind of "local" local searcher integrated in the AGLMA.

8.4.5 Adaptation

An adaptation has been implemented taking into account the features of this kind of fitness landscape in order to design a robust algorithm [179, 171]. At the end of each generation the following parameter is calculated:

$$\psi = 1 - \left| \frac{\hat{F}_{avg} - \hat{F}_{best}}{\hat{F}_{worst} - \hat{F}_{best}} \right| \tag{8.10}$$

where $\hat{F}_{worst}$, $\hat{F}_{best}$, and $\hat{F}_{avg}$ are the worst, best, and average of the fitness function values in the population, respectively.

As highlighted in [166], ψ is a fitness-based measurement of the fitness diversity which is well-suited for flat fitness landscapes. The employment of this parameter, taking into account the presence of plateaus in the fitness landscape. ψ, measures the population diversity in terms of fitness and is relative to the range of the fitness values $[\hat{F}_{best}, \hat{F}_{worst}]$ in the population. Thus, even when all fitness values are very similar, leading to $\hat{F}_{best}$ and $\hat{F}_{worst}$ being close to each other, ψ still gives a well scaled measure, since it uses the relative distance of $\hat{F}_{avg}$ from $\hat{F}_{best}$. The population has high diversity when $\psi \approx 1$ and low diversity when $\psi \approx 0$. A low diversity means that the population is converging (possibly in a suboptimal plateau). This parameter has been used in order to control coordination among the local searchers and a dynamic population size.

8.4.6 Coordination of the Local Searchers

ψ has been employed in order to execute an adaptive coordination of the local searchers so as to let them assist the evolutionary framework in the optimization process.

The SA is activated by the condition $\psi \in [0.1, 0.5]$. This adaptive rule is based on the observation that for values of $\psi > 0.5$, the fitness diversity is high and then the evolutionary framework needs to have a high exploitation of the available genotypes (see [180], [166] and [181]). In other words, under this condition the evolutionary framework does not require the assistance of a local searcher. On the other hand, if $\psi < 0.5$ the fitness diversity is decreasing and the application of the SA can introduce a new genotype in the population which can prevent a premature convergence. Basically, the SA has the potential to detect new promising solutions outside a suboptimal plateau into which the population could have fallen. In this sense, the SA has been employed as a local searcher with "global" exploratory features. The condition regarding the lower bound of usability of the SA ($\psi > 0.1$) is due to the consideration that if $\psi < 0.1$ convergence is approaching and the fitness value has already been drastically reduced.

Thus, the SA has the role of exploiting already existing good genotypes but nevertheless to explore other areas of the decision space. Due to its structure, the SA could lead new search directions but its application can lead to a solution which is worse than that which it started with. For this reason, in our implementation it is applied to the second best individual. The initial temperature $Temp^0$ has to be chosen for this local searcher. It is adaptively set to be $Temp^0 = \left|\hat{F}_{avg} - \hat{F}_{best}\right|$. This means that the algorithm does not accept worse solutions when the convergence has practically occurred.

The HJA is activated when $\psi < 0.2$ and is applied to the solution with best performance. The basic idea behind this adaptive rule is that the HJA has the role of quickly improving the best solution while staying in the same basin of attraction. In this light, the action of the HJA can be seen as purely "local". The condition $\psi < 0.2$ means that the HJA is employed when there are some chances that optimal convergence is approaching. An early application of this

local searcher can be inefficient since a high exploitation of solutions having poor fitness values would not lead to significant improvements of the population.

It should be noted that in the range $\psi \in [0.1, 0.2]$ both the local searchers are applied to the best two individuals of the population. This range is very critical for the algorithm because the population is tending towards a convergence but still has not reached such a condition. In this case, there is a high risk of premature convergence due to the presence of plateaus and suboptimal basins of attraction or false minima introduced by noise. Thus, the two local searchers are supposed to "compete and cooperate" within the same generation, merging the "global" search power of the SA and the "local" search power of the HJA under supervision of the evolutionary framework.

An additional rule has been implemented. When the SA has succeeded in enhancing the starting solution, the algorithm attempts to further enhance it by the application of the HJA. This choice can be justified by the consideration that when the SA succeeds, it returns a solution having better performance with a genotype (usually) quite different from the starting one and, therefore, belonging to a region of the decision space which has not yet been exploited.

8.4.7 Dynamic Population Size in Survivor Selection

The adaptation controls the population size whose dynamic variation has two combined roles. The first is to massively explore the decision space and thus prevent a possible premature convergence (see [182], [180]), the second is to *Implicitly Average* in order to compensate for noise by means of the evaluations of similar individuals [169]. The population is resized at each generation according to the formula:

$$S_{pop} = S^f_{pop} + S^v_{pop} \cdot (1 - \psi)\,, \tag{8.11}$$

where S^f_{pop} and S^v_{pop} are the fixed minimum and maximum sizes of the variable population S_{pop}, respectively.

The coefficient ψ is then used to dynamically set the population size [183,184] in order to prevent a premature convergence and stagnation. According to the first role, when the population is highly diverse a small number of solutions need to be exploited. When $\psi \approx 0$ the population is converging and a larger population size is required to increase the exploration. The main idea is that if a population is in a suboptimal plateau an increase of the population size enhances the chances of detecting new promising areas of the decision space and thus prevent premature convergence. On the other hand, if the population is spread out in the decision space it is highly desirable that the most promising solution leads the search and that the algorithm exploits this promising search direction.

According to the second role, it is well-known that large population sizes are helpful in defeating the noise (*Implicitly Averaging*) [185,186]. Furthermore, recent studies [187,170] have noted that the noise jeopardizes proper functioning of the selection mechanisms, especially in cases of low fitness diversity since the noise introduces a disturbance in pair-wise comparison. Therefore, the AGLMA

```
Pseudo-Random Initial Sampling of the weights W and self-adaptive parameters h;
Fitness evaluation of the initial population by F̂ = F^i_mean − σ^i/√n_s;
Calculate ψ = 1 − |(F̂_avg − F̂_best)/(F̂_worst − F̂_best)|;
while budget conditions and ψ > 0.01
  for all the individuals i
    for all the variables j
      h_i(j) = h_i(j) e^(τ N_j(0,1));
      W_i(j) = W_i + h_i(j) N_j(0,1);
    end-for
  end-for
  Fitness evaluation of the population by F̂ = F^i_mean − σ^i/√n_s;
  Sort the population made up of parents and offsprings according to their fitness values;
  if ψ ∈ [0.1, 0.5]
    Execute the SA on the individual with the 2^nd best performance;
    if ψ < 0.2
      Execute the HJA on the individual with the best performance;
    end-if
    if the SA succeeds
      Execute the HJA on the individual enhanced by the SA;
    end-if
  end-if
  Calculate S_pop = S^f_pop + S^v_pop · (1 − ψ);
  Select the S_pop best individuals to the subsequent generation;
  Calculate ψ = 1 − |(F̂_avg − F̂_best)/(F̂_worst − F̂_best)|;
end-while
```

Fig. 8.4. AGLMA pseudo-code

aims to employ a large population size in critical conditions (low diversity) and a small population size when a massive averaging is unnecessary.

After the calculation of S_{pop} in equation (8.11), the AGLMA selects for the subsequent generation, among parents and offspring, the S_{pop} candidate solutions having the best performance.

The algorithm stops when either a budget condition on the number of fitness evaluations is satisfied or ψ takes a value smaller than 0.01.

Fig. 8.4 shows the pseudo-code of the AGLMA.

8.5 Numerical Results

For the AGLMA 30 simulation experiments have been executed. Each experiment has been stopped after 1500000 fitness evaluations. At the end of each generation, the best fitness value has been saved. These values have been averaged over the 30 experiments available. The average over the 30 experiments defines the Average Best Fitness (ABF). Analogously, 30 experiments have been carried out with the Checkers Algorithm (CA) described in [167, 168] according to the implementation in [160], and the ACA which is the CA with the fitness as shown in (8.9) and the adaptive population size as shown in (8.11). In addition a standard real valued Genetic Algorithm (GA) has been run for the problem under study. The GA employs an arithmetic blend crossover and a Gaussian mutation. For the same P2P network, the BFS according to the implementation in

Gnutella and the random walker DFS proposed in [152] have been applied. Table 8.1 shows the parameter settings for the three algorithms and the optimization results. The final fitness $\hat{F}^b$ obtained by the most successful experiment (over the 30 sample runs), the related number of query packets P used in the query and the number of found resource instances R during the query are given. In addition the average best fitness at the end of the experiments $< \hat{F} >$, the final fitness of the least successful experiment $\hat{F}^w$ and the related standard deviation are shown. Since the BFS follows a deterministic logic, thus only one fitness value is shown. On the contrary, the DFS under study employs a stochastic structure and thus the same statistic analysis as that of GA, CA, ACA and AGLMA over 30 experiments has been carried out.

Numerical results in Table 8.1 show that the methods employing the neural network approach are more promising than the classical methods for P2P networks. Moreover, AGLMA and ACA outperform the CA and the AGLMA slightly outperformed the ACA in terms of final solution found. The GA performed significantly worse than the other optimization algorithms.

Fig. 8.5 shows a graphical representation of the solution in the most successful experiment (over the 30 carried out) returned by the proposed AGLMA. An index of the weights are shown on the x-axis and the corresponding weight values are shown on the y-axis (see the crosses in figure).

As shown in Fig. 8.5, according to AGLMA, we propose a neural network having a set of 298 weights, which take small values. More specifically, the proposed neural network contains 296 weight values between -1 and 1. On the contrary, two weights belonging to the first hidden layer take the values of around -1.5 and 1.5.

Table 8.1. Parameter setting and numerical results

PARAMETER	AGLMA	CA	ACA	GA	BFS	DFS
EVOLUTIONARY FRAMEWORK						
S^i_{pop}	30	30	30	30	–	–
S_{pop}	$\in [20, 40]$	30	$\in [20, 40]$	30	–	–
sample size n_s	10	–	10	–	–	–
SIMULATED ANNEALING						
initial temperature $Temp^0$	adaptive	–	–	–	–	–
temperature decrease	hyperbolic	–	–	–	–	–
maximum budget per run	600	–	–	–	–	–
HOOKE-JEEVES ALGORITHM						
exploratory radius	$\in [0.5, 0.01]$	–	–	–	–	–
maximum budget per run	1000	–	–	–	–	–
NUMERICAL RESULTS						
P	350	372	355	497	819	514
R	81	81	81	85	81	81
$\hat{F}^b$	3700	3678	3695	3366	3231	3536
$< \hat{F} >$	3654	3582	3647	2705	–	3363
$\hat{F}^w$	3506	3502	3504	0	–	3056
std	36.98	37.71	36.47	1068	–	107.9

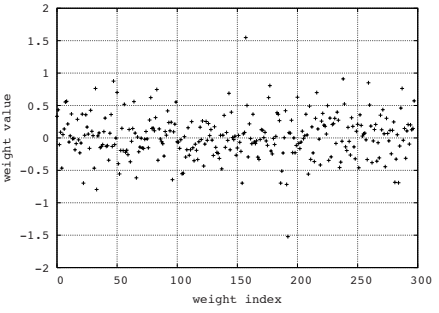

Fig. 8.5. Distribution of Neural Network Weights

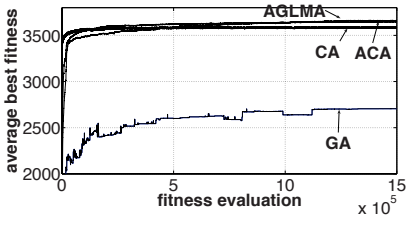

Fig. 8.6. Comparison of the algorithmic performance

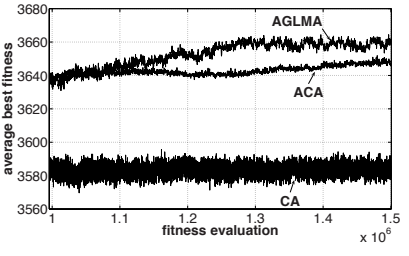

Fig. 8.7. Comparison of the algorithmic performance (zoom)

Fig. 8.6 shows the comparison of the performance over the 1.5×10^6 fitness evaluations and Fig. 8.7 shows a zoom detail of the algorithmic performance.

Fig. 8.6 shows that the AGLMA has a slower convergence than the CA and the ACA but it reaches a final solution having better performance. It is also clear that the ACA has intermediate performance between the CA and AGLMA. The ACA trend, in early generations, has a rise quicker than the AGLMA but slower than the CA. On the other hand, in late generations, the ACA outperforms the CA but not the AGLMA. As shown in Fig. 8.6, the GA performed much worse than the CA structured algorithms (CA, ACA, AGLMA) also in terms of convergence speed.

It can be remarked that the ACA can be seen as an AGLMA which does not employ local searchers but only executes *Implicit* (dynamic population size) and *Explicit Averaging* (n_s re-samples and modified fitness). In other words, the ACA does not contain the memetic components but does contain the noise filtering components. Fig. 8.7 shows that the ACA and the AGLMA are much more robust to noise than the CA. In fact, as shown in Fig. 8.7, the trend of the CA performance contains a high amplitude (about 20) and frequency ripple around a mean value, while the ACA and AGLMA performance are roughly monotonic. The oscillatory trend of the CA performance is due to an incorrect estimation of candidate solutions. The quick initial rise of the CA performance

is, according to our interpretation, also due to an overestimation of an unstable solution. On the contrary, the ACA and the AGLMA efficiently filter the noise and select only reliable solutions for the subsequent generations.

Regarding effectiveness of the local searchers, the comparison between the ACA and the AGLMA shows that the AGLMA slightly outperforms the ACA tending to converge to a solution having a better performance. Moreover it is shown that after 1.5×10^6 fitness evaluations, the trend of the AGLMA still continues to grow whilst the other trends seem to have reached a final value.

8.6 Conclusion

This chapter proposes an Adaptive Global Local Memetic Algorithm (AGLMA) for performing the training of a neural network, which is employed as computational intelligence logic in P2P resource discovery. The AGLMA employs averaging strategies for adaptively executing noise filtering and local searchers in order to handle the multivariate fitness landscape. These local searchers execute the global and local search of the decision space from different perspectives. The numerical results show that the application of the AGLMA leads to a satisfactory neural network training and thus to an efficient P2P network functioning. The comparison with two popular metaheuristics present in literature shows that the proposed approach seems to be promising in terms of final solution found and reliability in noise environment. Matching with another algorithm with intermediate features highlights the effectiveness of each algorithmic component integrated in the proposed algorithm.

The proposed neural network along with the learning strategy carried by the AGLMA allows the efficient location of resources with little query traffic. Thus, the user of the P2P network obtains plentiful amounts of information about resources without consuming a large portion of his own bandwidth for query traffic.

Acknowledgements

We wish to thank Teemu Keltanen and Andrea Caponio for their kind support in analyzing the data.