Constraint Optimization in Call Admission Control Domain with a NeuroEvolution Algorithm

Xu Yang
MPI-QMUL Information Systems Research Centre
Macao Polytechnic Institute
Macao SAR, China
xuy@mpi-qmul.org

Dr. John Bigham
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, United Kingdom
john.bigham@elec.qmul.ac.uk

ABSTRACT

The objective for Call admission control (CAC) is to accept or reject request calls so as to maximize the expected revenue over an infinite time period and maintain the predefined QoS constraints. This is a non-linear constraint optimization problem. This paper analyses the difficulties when handling QoS constraints in the CAC domain, and implements two constraint handling methods that cooperate with a NeuroEvolution algorithm called NEAT to learn CAC policies. The two methods are superiority of feasible points and static penalty functions. The simulation results are compared based on two evolution parameters: the ratio of feasible policies, and the ratio of 'all accept' policies. Some researchers argue that superiority of feasible points may fail when the feasible region is quite small compared with the whole search space, however the speciation and complexification features of NEAT makes it a very competitive method even in such cases.

Keywords

Constraint Optimization, Call Admission Control, NeuroEvolution of Augmenting Topologies (NEAT)

1. INTRODUCTION

CAC schemes in wireless networks have been extensively studied during the last two decades due to the growing popularity of wireless communications and the central role that CAC plays in QoS provisioning. A CAC scheme aims at maintaining the delivered QoS to different calls (or users) at the target quality level by limiting the number of ongoing calls in the system. [1, 2]

Connection-level QoS measures the service connectivity and continuity of a wireless networks, and provides the basis of packet-level QoS. [2] It is often considered by the use of two parameters, the new call blocking rate (NBR) and handoff failure rate (HFR). The new call blocking probability is the probability of a new call request being rejected. The handoff failure probability is the probability of a handoff call being forced terminated. [2]

Generally dropping the handoff of an ongoing connection is considered more objectionable than blocking a new connection. However, designing a system with zero HFR is practically impossible. Hence, most CAC policies attempt to provide an acceptable HFR, called target QoS for HFR. For example, in second-generation cellular system, it is acceptable that the NBR is lower than 5% while the HFR is lower than 1 or 2% for voice service.[3]

To obtain optimal CAC policies a variety of artificial intelligence techniques have been proposed for the CAC domain, such as Genetic Algorithms (GAs). A genetic algorithm is used in [4] to find a near optimal CAC by minimizing a linear combination of NBR and HFR with a large weight given to HFR in the fitness function. It has been shown in [4] that the near-optimal CAC based on genetic algorithms has a very close performance to the optimal algorithm found by a MDP technique. However this research only considers one class of traffic, which is not suitable for a network with multiple classes of service. Additionally the fitness function uses a relative penalty factor to give high pressure to the handoff failure rate, which should be adjusted through extensive experiments to adapt to different traffic loads otherwise the QoS constraint may not be guaranteed.

Similarly, a genetic algorithm is used in [5] to maximize the resource utilization of multimedia multiple-class resource allocation schemes while maintaining a hard constraint on Handoff failure rate. The authors prohibit those policies which can not satisfy all the constraints, thereby creating offspring to handle the HFR constraint. This kind of constraint handling method may fail when the feasible policies are too difficult to be generated in initial populations. Therefore it is not suitable for cases in which QoS constraints are very difficult to be satisfied. Additionally the encoding method is quite complex. The authors code the call admission control decisions as a sequence of $2m-bit$ binary strings, where m denotes the number of classes of traffic. Each system state associates such a string. Therefore the size of the space encoding grows exponentially with m and the network capacity.

Neither of these GA approaches gives details of the fitness function formula and how to address the QoS constraints. Additionally the GA solutions require experts to make lots of effort on the algorithm design including encoding, decoding, mutation, crossover operators, and many other parameter adjustments.

One of our previous published research papers [6] provided a different approach performing the CAC scheme through a form of NeuroEvolution (NE) algorithm called NeuroEvolution of Augmenting Topologies (NEAT) [7]. The objective of the algorithm is to maximize network utility and satisfy predefined QoS constraints.

The fitness function is formulated as the average reward per request event obtained in an evaluating episodic. The penalty is controlled by a negative fitness score if a QoS constraint has been violated. The experiment results show the learned CAC policy can effectively guarantee the predefined QoS constraints. However the performance of the approach is very sensitive to the changes of value of penalty fitness, and there is no detailed analysis to address how to handle constraints in the CAC domain.

There are many papers that use Evolutionary Algorithms (for example, Genetic Algorithms) as a direct search algorithm to handle constraint optimization problem. [8-10] However no papers have been published to handle constraints when using NEAT.

This paper investigates the difficulties to handle QoS constraints in CAC domain and implements two constraint handling methods that can cooperate with NEAT to handle QoS constraints.

In order to evaluate the performance of our CAC algorithms, we simulate a mobile communication system using a discrete event simulator. The simulation model has been widely used in many research papers [5, 11, 12], and is described in the following sections.

In this paper only non-adaptive services are considered, i.e. the bandwidth of a connection is constant. We assume m classes of services: $\{1,2,\cdots,m\}$. A service in class i consumes b_i units of bandwidth or channels, and the network can obtain r_i rewards per logical second by carrying it. We also assume that service requests of each class arrive according to Poisson distribution. The holding time for each service class is exponentially distributed. All the arrival distributions and call holding distributions are independent of each other. The arrival events include new call arrival events and handoff arrival events. Since the call departures do not affect the CAC decisions, we only consider the arrival events in the CAC state space. Additionally, we only consider one cell with fixed capacity. The fixed total number of bandwidth (channels) is C.

Let $\lambda_{i,s}$ denote the arrival rate of new setup requests in class i (that is the number of setup arrivals per unit of time), and $\lambda_{i,h}$ as the arrival rate of new handover requests in class i, and $\mu_{i,s}^{-1}$ as the average holding time of setup calls in class i, and $\mu_{i,h}^{-1}$ as the average holding time of handover requests in class i.

The rest of this paper is organized as follows. Section 2 gives a brief introduction to NEAT, and how to apply the NEAT to the CAC application. Section 3 introduces the definition of constraint optimization problem and analyses the difficulties to solve QoS constraints in CAC domain. In section 4 two constraint handling methods are selected to cooperate with NEAT. Section 5 presents the experiment results. Section6 summarizes the paper.

2. HOW TO APPLY NEAT IN CAC [1]

NEAT is a kind of NeuroEvolution (NE) method that has been shown to work very efficiently in complex reinforcement learning problems. NE is a combination of neural networks and genetic algorithms where neural networks are the phenotype being evaluated. The genotype is a compact representation that can be translated into an artificial neural network [7].

The evolution of NEAT starts with a randomly generated small set of neural networks with simple topologies. Each of these neural networks is assigned a fitness value depending on how well it suits the solution. Once all the members of the population are assigned fitness values, a selection process is carried out where better individuals (high fitness value) stand a greater chance to be selected for the next operation. Selected individuals undergo recombination and mutation to result in new individuals. Low fitness individuals are discarded from the population and better ones are included. Structural mutations add new connections and nodes to networks in the population, leading to incremental growth. The whole process is repeated with this new population until some termination criteria is satisfied. The average fitness of the population is expected to increase over generations. [7]

Additionally NEAT protects innovation through speciation so that for a time individuals compete primarily within their own niches instead of with the whole population. Throughout evolution, NEAT maintains a list of species numbered in the order they appeared. The distance δ between two network genomes can be measured as a linear combination of the number of excess and disjoint genes, as well as the average weight differences of matching genes. If a genome's distance to a representative of any existing species is less than a compatibility threshold δ_t, it is placed into this species. Otherwise, a new species is created. [7]

NEAT can be seen as a black box, which can provide a neural network for receiving the inputs and generating the outputs. Normally the inputs are the perceived state of the environment that is essential to make the action decision. In the CAC domain, the inputs denote the type of new arriving request call and the consumed bandwidth by each kind of traffic. Generally the outputs are the possible actions that can be performed in the real application. In CAC, there are only two possible actions: Accept and Reject. The output is a real number, and its value is between 0 and 1, if it is larger than 0.5, then the action selected is Accept; otherwise, it is Reject.

The fitness function gives the goal of the learning system. The objective of CAC is to maximize the network revenue; the objective function is formulated as the average reward obtained per request service, which is the total rewards divided by the number of request services. Let N be the total number of service requests (includes setup and handoff calls) during the evaluation of each policy. The network obtains rewards by carrying accepted services and obtains nothing by rejecting all services. R is the reward that the network obtains by carrying each accepted service.

$$F_{objective} = \frac{\sum R_n}{N} \qquad (1)$$

To simplify the above formula, define the service demand parameter α as

$$\alpha_{i,s} = r_i \frac{\lambda_{i,s}\mu_{i,s}^{-1}}{\sum_{i=0}^{m}\left(\lambda_{i,s}+\lambda_{i,h}\right)}, \quad \alpha_{i,h} = r_i \frac{\lambda_{i,h}\mu_{i,h}^{-1}}{\sum_{i=0}^{m}\left(\lambda_{i,s}+\lambda_{i,h}\right)}$$

[1] Partially of this section is from our previous published paper[6] J. B. Xu Yang, "A Call Admission Control Scheme using NeuroEvolution Algorithm in Cellular Networks," in *the 20th International Joint Conference on Artificial Intelligence (IJCAI07)*, Hyderabad, India, 2007.

so, $$f_{objective} = \sum_{i=0}^{m} (\alpha_{i,s} p_{i,s} + \alpha_{i,h} p_{i,h}) \quad (2)$$

where $p_{i,s}$ ($p_{i,h}$) denotes the acceptance probability of a new setup (handover) request call in class i per request. Since in the current model calls are never dropped once accepted unless at handoff time, $NBR_{i,s} = 1 - p_{i,s}$, and $HFR_{i,h} = 1 - p_{i,h}$.

3. THE DIFFICULTIES TO HANDLE QOS CONSTRAINTS IN CAC DOMAIN

In general, a constrained numerical optimization problem is defined as [13]:

Find $\vec{x}$ which optimize $z = f(x_1, x_2, ..., x_n) \qquad x \in \Re^n$

subject to: $$g_j(\bar{x}) \le 0, \qquad j = 1,...J,$$ $$h_j(\bar{x}) = 0, \qquad k = 1,...K,$$

where $\vec{x}$ is the vector of solutions $\vec{x} = [x_1, x_2, ..., x_n]^T$, m is the number of inequality constraints and p is the number of equality constraints. Normally, equality constraints can be transformed into inequality constraints of the form:

$$\left| h_j(\bar{x}) \right| - \varepsilon \le 0$$

where ε is a very small value which is the tolerance allowed. Let the set $S \in \Re^n$ define the search space, and $F \subseteq S$ defines the feasible region of the search space, the infeasible part I is the set of remaining elements $I = S \setminus F$. If no constraints are given then F equals S . Figure 1 shows one example of a search space S and its feasible region F .

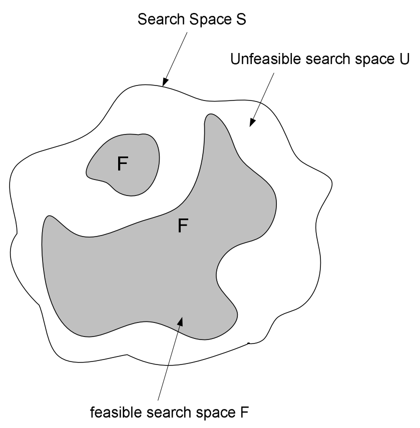

Figure 1 The feasible and unfeasible search space [13]

The main features which make a global constraint optimization problem difficult to be solved are described by Michalewicz & Schoenauer [13]:

- The type of objective function (such as if a linear or non linear function).
- The types of constraints (such as if linear or non linear constraints, equality or inequality).
- The number of constraints: the difficulty in satisfying constraints will increase (generally more than linearly) with the number of constraints.
- Connectivity of feasible region (disjoint or connected).
- Size of feasible region with respect to the whole search space. Many constraint handling methods fail when the ratio of feasible to infeasible area in its rectangular search space is too small.

The optimal CAC problem that considers the QoS constraints is a very difficult problem to solve. The main challenges are summarized as follows.

In the CAC domain, the objective function and constraints are non linear and not differentiable. The feasible area is very small compared with the whole search domain. For example, even if the HFR constraint is 1%, the possible HFR can be from 0 to 100%. So when there are total m classes of traffic, the feasible area is only 10^{-2m} of the search space, which makes the search for feasible policies very difficult.

Moreover, (a) the objective function to maximize the network revenue conflicts with the hard QoS constraints and (b) HFRs and NBRs constraints are conflicting requirements. Decreasing one of them may cause an increase of the other parameters. Furthermore, the environment is dynamic, a CAC action will affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. Additionally, to make an optimal decision, the network should not only consider the current state, but also the future coming traffic, which cannot be known in advance. Therefore a long run time is required for evaluating the performance of a selected CAC policy. However, for the CAC domain the state space is huge. Each policy can only be evaluated by a limited amount of events, and it is not possible to go through the whole state space in short time period. Therefore the fitness evaluation is noisy due to measurement limitations or incomplete training.

4. CONSTRAINT HANDLING METHODS AND NEAT

There are several approaches proposed in GAs to handle constrained optimization problems. These approaches can be grouped into four major categories [9]:

1. Methods based on penalty functions which penalize infeasible individuals;
2. Separation of objectives and constraints;
3. Methods using special representations and operators to preserve feasibility of solutions;
4. Hybrid methods.

Methods based on penalty functions and superiority of feasible policies, which belong to category 1 and 2, penalize policies which can not satisfy the constraints. These methods do not require the modification of evolution process and can cooperate with NEAT very well. Most methods falling into category 3 and 4 require modification of the genetic operators and evolution process, therefore can not exploit NEAT's advantages.

4.1 Penalty functions

The most common approach to handling constraints (particularly, inequality constraints) in the Evolution Algorithm community is to use penalties. In general, a penalty function approach can be defined as [14]:

$$F(\bar{x}) = \begin{cases} f(\bar{x}) & \bar{x} \in \textit{feasible region} \\ f(\bar{x}) + penalty(\bar{x}) & \bar{x} \in \textit{infeasible region} \end{cases} \quad (3)$$

$$penalty(\bar{x}) = \sum_{j=1}^{m} R_j \left\langle g_j(\bar{x}) \right\rangle^2$$

Where $f(\bar{x})$ is the objective function to evaluate a feasible individual, $g_j(\bar{x})$ is the constraint violation that measure the distance between a search point and the feasible region, and $\langle \ \rangle$ denotes that if the operand is negative a value zero is returned, otherwise the value returned is that of the operand. The parameter R_j is the penalty parameter of the j-th inequality constraint. The purpose of a penalty parameter R_j is to make the constraint violation $g_j(\bar{x})$ of the same order of magnitude as the objective function value $f(\bar{x})$ [14].

The results of many experiments indicate that the quality of the solutions using penalty functions are very sensitive to changes in values of penalty parameters, and that the parameters must be set right to obtain a feasible solution. [13]

There are many different penalty function methods, such as death penalty[12], in which all infeasible individuals are either rejected or get a zero fitness regardless of their amount of constraint violation; static penalty [15], in which penalties are functions of the degree of violation of constraints, and the penalty parameters never change during evolution; dynamic penalty [16], in which penalties are functions of the degree of violation of constraints as well as the generation number t; annealing penalties [10], a method based on the idea of simulated annealing: the penalty coefficients are only changed after the algorithm has been trapped in a local optima; adaptive penalties [17], the value of the penalty factors are updated based on information of the evolutionary process instead of a predefined variation function as in the case of dynamic penalties.

However Michalewicz and Schoenauer [18] concluded that the static penalty function method (without any sophistication) is a more robust approach than the sophisticated methods after comparing several variations of a simple penalty function techniques to more sophisticated mathematical techniques.

In the CAC domain, two kinds of penalties are chosen to punish the infeasible solutions:

- Penalties which are the functions of the distance between the learned CAC policies and the predefined constraints (the boundary of feasible region).
- Penalties which are functions of the number of violated constraints.

To follow these rules, the penalty function can be formulated as a linear function:

$$p(\bar{x}) = \sum_{i=1}^{m} \left(\eta + w(T - x_i)\right) \qquad if \quad x > T$$

w and η are constant penalty factors to push NEAT to search in the feasible area, and they are same for all QoS parameters to treat them fairly. The total penalties are the sum of the QoS constraints violation.

4.2 Superiority of feasible points

This method differentiates between feasible solutions and infeasible solutions, the feasible solutions have superiority over unfeasible ones, and infeasible solutions are penalized to provide a search direction towards the feasible region [8]. This approach has promising success in many GA applications.

The basic principle is:

- Any feasible solution is preferred to any infeasible solution;
- Among two feasible solutions, the one having better objective function value is preferred;
- Among two infeasible solutions, the one having smaller constraint violation is preferred.

One example of the fitness functions to follow these rules is defined as [8]:

$$F(\bar{x}) = \begin{cases} f(\bar{x}) & \bar{x} \in \textit{feasible region} \\ f_{worst} + penalty(\bar{x}) & \bar{x} \in \textit{infeasible region} \end{cases} \quad (4)$$

Where f_{worst} is the objective function value of the worst feasible solution in the last population. Thus the fitness of an infeasible solution not only depends on the amount of constraint violation, but also on the feasible solutions in the population.

The feasible solutions are compared by objective function, and two infeasible solutions are compared by the degree of constraint violation. Among two infeasible solutions, the one having smaller constraint violation is preferred. Careful comparisons among feasible and infeasible solutions are made so as to provide a search direction towards the feasible region.

However, Coello mentioned this approach may fail when the feasible region is too small [9]. Additionally a niching (speciation) method along with a controlled mutation operator (high mutation rate) is required to maintain diversity among feasible solutions. [13]

However, two features of NEAT can help to use this method to handle constraints: a) NEAT speciates the population so that individuals compete primarily within their own niches [7]. This feature can help to keep the diversity of population so as to avoid trapping into local optima. b) NEAT begins with a uniform population of simple networks with no hidden nodes and inputs connected directly to outputs, and new structure is introduced incrementally by adding new genes. In this way, NEAT tends to search through a minimal number of weight dimensions and find an appropriate complexity level for the problem, which can significantly reducing the search dimension and the number of generations necessary to find a feasible solution.

To apply this method in CAC domain, the penalty function only need to measure the distance of penalty violation.

$$p(\bar{x}) = \sum_{i=1}^{m} w(T - x_i) \qquad if \quad x_i > T$$

5. RESULTS OF EXPERIMENT

There are in total three different constraint handling methods that are compared: two methods use static penalty functions with different penalty factors, another uses superiority of feasible points [8].

1) *NEAT_penalty_soft*. The penalty function only measures the distance between the infeasible policies to the boundary of feasibility.

$$p(\bar{x}) = \sum_{i=1}^{m} 10.0 \times (T - x_i) \qquad if \quad x > T \qquad (5)$$

2) *NEAT_penalty_hard*. The penalty function not only measures the distance between the infeasible policies to the boundary of the feasible region, but also is proportional to how many constraints are violated. CHECK I HAVE NOT GOT IT WRONG

$$p(\bar{x}) = \sum_{i=1}^{m} (-5 + 10.0 \times (T - x_i)) \qquad if \quad x > T \quad (6)$$

3) *NEAT_Superiority*. The penalty function only needs to measure the distance between the infeasible policies to the boundary of feasible region, which is same as *NEAT_penalty_soft*.

Two evolution parameters are compared: *ratio of feasible solutions*, *ratio of all-accept solutions*.. Ratio of feasible is the percentage of feasible solutions in the solutions in one generation. Ratio of All-Accept policies is the percentage of All-Accept policies in one generation.

All-Accept policies are the policies that always accept arriving calls no matter how busy the network cell is. Since it does not reject any new setup calls to reserve bandwidth capacity for future handoff calls, the NBRs satisfy predefined NBR constraints and only HFR constraints are violated, the penalties may not big enough to out-weigh the value of the objective function. Additionally the All-Accept policies are randomly generated very easily. Therefore the All-Accept policies are quite competitive infeasible policies.

In the simulation, only two applications are considered: c_1 *and* c_2, c_1 requires 1 bandwidth unit, and c_2 requires 2 bandwidth units. The reward rates are the same for these two kinds of traffic. The traffic parameters are shown in Table 1.

Table 1.Traffic parameters

Parameter s	c_1		c_2	
	Setu	Handove	Setup	Handover
λ^{-1}	0.2	0.1	0.1	0.05
μ^{-1}	30	20	20	15

There are four QoS constraints. The setup and handoff constraints for each kind of traffic are 20% and 1%. The feasible region only takes 4×10^{-6} of the whole search region, which is quite small.

The simulation runs 300000000 logic seconds. The population size is 200, and each member of the population was evaluated by 40000 arriving setup or handoff events. To maintain the species of the population, the target of species size is 5 by using dynamic speciation. A policy with highest fitness score is selected as learning result among five simulation runs.

The experiment results are shown in figure 2 and 3. It can be seen that the ratio of feasible policies using the penalty_soft method is always around 0, there are almost no feasible policies generated during evolution. On the other hand its ratio of all_accept grows rapidly from the beginning of evolution, and reach 90% at the end of evolution. The evolution becomes trapped into the all-accept policies, and can not evolve good feasible policies. The evolution fails if the penalty only measures the distance between infeasible solutions to the feasible region.

The feasible ratio of the Superiority method grows faster than Penalty_hard method. After 20 generations it reached 25%. in 40 generations, 35% by the final generation. However, Penalty_hard reaches 20% at the 170th generation, and jump to 25% at final 5 generations. The Superiority method can generate more feasible policies in each generation.

Additionally the all_accept policies ratio of the Superiority method decreases dramatically after 30 generations from 60% to 5%, however Penalty_hard method decrease slowly: at the start 30 generations, the ratio of all_accept policies is around 70%, at the end of generations, it still maintain around 20%. The Penalty_hard method is more attracted by the all_accept policy, and spent lots of time on it.

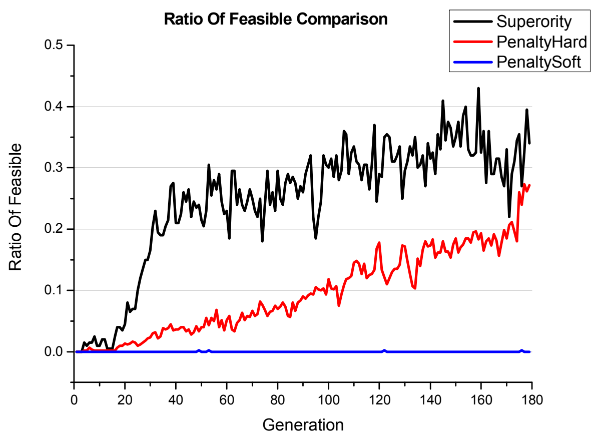

Figure 2 The comparison of ratio of feasible policies

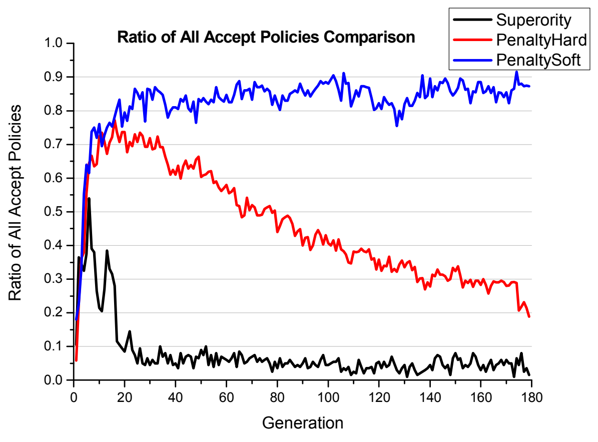

Figure 3 the comparison of ratio of all-accept policies

Table 2 shows the evaluated QoS parameters for each method. The Greedy method is a traditional CAC scheme that always accepts a request call if there is available bandwidth capacity to carry it, otherwise it rejects the call. This method does not consider maintaining any QoS parameters, and is used as a standard method to evaluate other CAC schemes.

It can be seen that learned policy for Superiority and Penalty_hard methods all satisfied the predefined constraints.

Table 2 The QoS parameter comparsion

The highest fitness policy	**C1-NBR**	**C2-NBR**	**C1-HFR**	**C1-HFR**
Superiority	4.95	17.24	0.21	0.97
Penalty-Hard	5.37	14.57	0.23	0.98
Penalty-Soft	1.95	5.40	2.15	5.41
Greedy	1.94	5.38	2.13	5.38

In conclusion, the Superiority method works best among the three constraint handling methods evaluated for the CAC domain.

6. CONCLUSION

To handle QoS constraints in the CAC domain is quite difficult, as the feasible region is very small compared with the whole search space, the environment is dynamic, and the QoS constraints are relative and conflict with each other. This paper investigates two constraint handling methods that cooperate with NEAT to handle constraint optimization in CAC domain. The experiment results show that using the superiority of feasible points to cooperate with NEAT works better than static penalty functions. The feasible ratio grows faster and many feasible policies can be generated. Using NEAT can fill a gap allowing the application of the superiority of feasible points even when the feasible region is very small; making it a very competitive constraint handling method.

7. ACKNOWLEDGMENTS

Thanks Kenneth O. Stanley very much for providing free software of NEAT in the website http://www.cs.utexas.edu/~nn/index.php.

8. REFERENCES

[1] E. S. Oliver Yu, Anfei Li, "Integrated connection-level and packet-level QoS controls over wireless mesh networks " *Journal of Parallel and Distributed Computing,* vol. 68, pp. 336-347, March 2008.

[2] L. H. Huan Chen, Sunil Kumar, C.-C.Jay Kuo, *Radio Resource Management For multimedia QoS Support in Wireless Networks*, Kluwer Academic Publishers ed.: Kluwer Academic Publishers, 2003.

[3] G. C. Sunho Lim , Chita R. Das "A unified bandwidth reservation and admission control mechanism for QoS provisioning in cellular networks," *Performance Evaluation of Wireless Networks,* vol. 4, pp. 3-18, 2004.

[4] A. R. Yener, C., "Genetic algorithms applied to cellular call admission: localpolicies," *Vehicular Technology, IEEE Transactions on,* vol. 46, pp. 72 - 79, Feb 1997.

[5] C. L. P. Y. W. Yang Xiao; Chen, "A near optimal call admission control with genetic algorithm formultimedia services in wireless/mobile networks," in *National Aerospace and Electronics Conference*, 2000, pp. 787 - 792.

[6] J. B. Xu Yang, "A Call Admission Control Scheme using NeuroEvolution Algorithm in Cellular Networks," in *the 20th International Joint Conference on Artificial Intelligence (IJCAI07)*, Hyderabad, India, 2007.

[7] K. O. Stanley, "Efficient Evolution of Neural Networks through Complexification," in *the Faculty of the Graduate School of.* vol. Doctor of Philosophy: The University of Texas at Austin, 2004.

[8] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering,* vol. 186:22, pp. 311-338, 2000.

[9] C. A. C. Coello, "Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art," *Computer Methods in Applied Mechanics and Engineering,,* vol. 191, pp. 1245-1287(43), 4 January 2002.

[10] a. N. A. Z. Michalewicz, "Evolutionary optimization of constrained problems," in *3rd Annual Conference of Evolutionary Programming*, 1994, pp. 84-97.

[11] R. Ramachandran, T. Don, and N. Ramesh, "On optimal call admission control in cellular networks," *Wirel. Netw.,* vol. 3, pp. 29-41, 1997.

[12] S. Sidi-Mohammed, Andr, B. Luc, and P. Guy, "Call admission control in cellular networks: a reinforcement learning solution," *Int. J. Netw. Manag.,* vol. 14, pp. 89-103, 2004.

[13] Z. Michalewicz, "A Survey of Constraint Handling Techniques in Evolutionary Computation Methods," in *4th Annual Conference on Evolutionary Programming*, 1995.

[14] Z.Michalewicz, "Genetic ALgorithm, Numerical Optimization, and Constraints. ," 1995.

[15] S. H. Y. L. A.Homaifar, and X.Qi, "Constrained Optimization via Genetic Algorithms," *SIMULATION,* vol. 62, pp. 242-253, 1994.

[16] C. H. J.Joines, "On the use of non-stationary penalty functions to solve nonlinear constrained optimained," in *problem with GAs, in: Proceedings of the First IEEE International Conference on Evolutionary Comutionary IEEE Press*, New York, 1994, pp. 579--584.

[17] A. E. E. a. Z. Ruttkay., "Self adaptivity for Constraint Satisfaction: Learning Penalty Functions. ," in *Artificial Evolution'97*, Berlin, 1998, pp. 95-106.

[18] D. D. Zbigniew Michalewicz, Rodolphe G. Le Riche, Marc Schoenauer, "Evolutionary Algorithms for Constrained Engineering Problems (1996)," *nternational Syposium on Methodologies for Intelligent Systems,* 1996.