Solving Symmetric and Asymmetric TSPs by Ant Colonies

Luca Maria Gambardella
IDSIA
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale
Corso Elvezia 36, 6900 Lugano, Switzerland
luca@idsia.ch
http://www.idsia.ch/~luca

Marco Dorigo
IRIDIA, Université Libre de Bruxelles
Avenue Franklin Roosevelt 50
CP 194/6, 1050 Bruxelles, Belgium, EU
mdorigo@ulb.ac.be
http://iridia.ulb.ac.be/dorigo/dorigo.html

Abstract - **In this paper we present ACS, a distributed algorithm for the solution of combinatorial optimization problems which was inspired by the observation of real colonies of ants. We apply ACS to both symmetric and asymmetric traveling salesman problems. Results show that ACS is able to find good solutions to these problems.**

I. Introduction

In this paper we present Ant Colony System (ACS), a novel distributed approach to combinatorial optimization based on the observation of real ant colonies behavior. ACS finds its ground in one of the authors previous work on the so-called Ant System (AS) [1],[2],[5],[7] and in Ant-Q [8] an extension of AS with Q-learning [12], a reinforcement learning technique. In particular, ACS is a revisited version of Ant-Q where a different way to update the experience accumulated by the artificial ants has been introduced [6].

All the mentioned systems belong to the Artificial Ant Colonies (AAC) family of algorithms that has been applied to various combinatorial optimization problems like the symmetric and asymmetric traveling salesman problems (TSP and ATSP respectively), the quadratic assignment problem [10], and the job-shop scheduling problem [3].

This paper is centered on the presentation of the ACS algorithm and on its application to both the symmetric and asymmetric versions of the TSP.

II. The ACS algorithm

We introduce the ACS algorithm by its application to the traveling salesman problem (TSP) or to the more general asymmetric traveling salesman problem (ATSP). They are defined as follows.

TSP

Let $V = \{v_1, \ldots, v_n\}$ be a set of cities,
$A = \{(r,s) : r,s \in V\}$ be the edge set, and $d_{rs} = d_{sr}$ be a cost measure associated with edge $(r,s) \in A$.
The TSP is the problem of finding a minimal length closed tour that visits each city once.
In the case cities $v_i \in V$ are given by their coordinates (x_i, y_i) and d_{rs} is the Euclidean distance between r and s then we have an Euclidean TSP.

ATSP

If $d_{rs} \neq d_{sr}$ for at least one (r,s) then the TSP becomes an ATSP.

In the following of this section we will talk generically of ATSP problems, which includes TSP as a special case.

Let k be an agent whose task is to make a tour: visit all the cities and return to the starting one. Associated to k there is the list $J_k(r)$ of cities still to be visited, where r is the current city (this is equivalent to say that agent k remembers already visited cities). An agent k situated in city r moves to city s using the following rule, called *pseudo-random-proportional* action choice rule (or state transition rule) :

$$s = \begin{cases} \arg\max_{u \in J_k(r)} \left\{ [\tau(r,s)] \cdot [\eta(r,s)]^{\beta} \right\} & \text{if } q \leq q_0 \quad \text{(exploration)} \\ S & \text{otherwise} \quad \text{(exploitation)} \end{cases} \tag{1}$$

where:

- $\tau(r,s)$, is a positive real value associated to edge (r,s) and is the ACS algorithm counterpart of pheromone left by the real ants. $\tau(r,s)$'s are changed at run time and are intended to indicate how useful it is to make move s (i.e., to go to city s) when in state r.

0-7803-2902-3/96/ $4.00 © 1996 IEEE

Authorized licensed use limited to: IEEE Transactions on SMC Associate Editors. Downloaded on May 16, 2009 at 07:48 from IEEE Xplore. Restrictions apply.

- $\eta(r,s)$, is a heuristic function which evaluates the utility of move s when in city r. For example, in the ATSP $\eta(r,s)$ is the inverse of the distance between cities r and s.
- Parameter β weigh the relative importance of the heuristic function.
- q is a value chosen randomly with uniform probability in [0, 1], and q_0 ($0 \leq q_0 \leq 1$) is a parameter: the smaller q_0 the higher the probability to make a random choice. In short q_0 determines the relative importance of exploitation versus exploration in formula (1).
- S is a random variable selected according to the distribution given by formula (2) which gives the probability with which an agent in city r chooses the city s to move to.

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)]\cdot[\eta(r,s)]^{\beta}}{\sum_{z \in J_k(r)}[\tau(r,z)]\cdot[\eta(r,z)]^{\beta}} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

This state transition rule will favor transitions towards nodes connected by short edges with an high amount of trail. Formula (1) shows how a transition can either exploit accumulated knowledge about the problem (knowledge accumulates in the form of different amount of trail on edges) or explore new edges (exploration is biased towards short and high trail edges).

The local-updating rule

While building a solution (i.e., a tour) of the TSP, ants visit edges and change their trail by applying the following local updating rule:

$$\tau(r,s) := (1-\rho)\cdot\tau(r,s) + \rho\cdot\tau_0 \quad (3)$$

where τ_0 is a parameter. The effect of the application of formula (3) will be discussed in Section III.

The global-updating rule

Once all ants have completed their solutions, edges (r,s) belonging to the shortest tour made by an ant have their trail changed by applying the following global updating rule:

$$\tau(r,s) := (1-\alpha)\cdot\tau(r,s) + \alpha\cdot(L_{best-iter})^{-1} \quad (4)$$

Global trail updating provides a higher amount of trail to shorter tours. In a sense, this is similar to a reinforcement learning scheme in which better solutions get a higher reinforcement (as it happens, for example, in genetic algorithms).

We have defined two different ways to choose the ant that is allowed to perform the global updating: *iteration-best updating* and *global-best updating*. In the *iteration-best* method the selected agent is the agent who did the shortest tour in the current iteration while in the *global-best* method the selected agent is the agent who did the shortest tour since the beginning of the computation[1]. In all the experiments presented in this paper we will apply the *global-best updating* strategy.

In words the ACS algorithm is presented in Figure 1 and can be described as follows.

First, at Phase 1 there is an initialization phase in which an initial value τ_0 is given to τ-values, and each agent k is placed on a city r_{k1} chosen according to some policy (in the following experiments we place maximum one agent for each city). Also, the set $J_k(r_{k1})$ of the still to be visited cities is initialized.

Then, at Phase 2, a cycle, in which each of the m agents makes a move and the $\tau(r,s)$'s are updated according to formula (3), is repeated until each agent has finished its tour and is back in the starting city.

At Phase 3, the length L_{best} of the tour done by agent who made the shortest tour is computed, and it is used to compute the delayed reinforcements $(L_{best\text{-}iter})^{-1}$. Then $\tau(r,s)$'s are updated using formula (4).

Finally, Phase 4 checks whether a termination condition is met, and if it is not the case the algorithm returns to Phase 2.

Usually the termination condition is verified after a fixed number of cycles, or when no improvement is obtained for a fixed number of cycles. (In experiments in which the optimal value was known *a priori* the algorithm was stopped as soon as the optimum was found.)

We would like to point out that in ACS (and in general in any application of AAC systems to ATSP) the only knowledge related to ATSP problems is the heuristic function $\eta(r,s)$ that represents the inverse distance between node r and node s. In ACS we do not use any local tour improvement heuristics (like r-opt, Lin&Kerningan, see [9] and [11] for a complete presentation of TSP heuristics) to modify the results of our computation and we do not maintain any explicit notion of tour. Tours are globally used (see formula 4) to reinforce a set of edges in the ATSP graph but, later on, we only use the accumulated trail to generate new tours. Possible extensions of ACS that includes dedicated ATSP heuristics will be discussed in section V.

1 An analisys of the behaviour of Ant-Q (the predecessor of ACS) using these two different updating policies has been presented in [8].

Authorized licensed use limited to: IEEE Transactions on SMC Associate Editors. Downloaded on May 16, 2009 at 07:48 from IEEE Xplore. Restrictions apply.

```
1./* Initialization phase */
   For each pair (r,s) τ(r,s):= τ0  End-for
     For k:=1 to m do
         Let rk1 be the starting city for agent k
         Jk(rk1):= {1, ..., n} - rk1
         /* Jk(rk1) is the set of yet to be visited cities for
            agent k in city rk1 */
         rk:= rk1  /* rk is the city where agent k is located */
     End-for
2. /* This is the phase in which agents build their tours. The
   tour of agent k is stored in Tourk. */
   For i:=1 to n do
     If i<n
       Then
         For k:=1 to m do
          Choose the next city sk according to formula (1)
          and formula (2)
          If i<n-1 Then Jk(sk):= Jk(rk) - sk
          If i=n-1 Then Jk(sk):= Jk(rk) - sk + rk1
          Tourk(i):=(rk ,sk)
         End-for
       Else
         For k:=1 to m do  /* In this cycle all the agents go
                              back to the initial city rk1 */
          sk := rk1
          Tourk(i):=(rk ,sk)
         End-for
     /* In this phase local updating is computed and
       τ-values are updated using formula (3)*/
     For k:=1 to m do
         τ(rk ,sk):=(1-ρ)τ(rk ,sk)+ ρτ0
         rk := sk /* New city for agent k */
     End-for
   End-for
3. /* In this Phase delayed reinforcement is computed and
     τ-values are updated */
   For k:=1 to m do
     Compute Lk  /* Lk is the length of the tour done
                    by agent k*/
   End-for
   Compute Lbest-iter
   /*Update edges belonging to Lbest-iter using formula(4) */
   For each edge (r,s)
         τ(rk ,sk):=(1-α)τ( rk ,sk)+ α (Lbest-iter)-1
   End-for
4. If (End_condition = True)
     then Print shortest of Lk
     else goto Phase 2
```

Figure 1: The ACS algorithm

III. Algorithm Analysis

In this section we present results of a micro-level investigation in which we observe how trail changes on edges as a function of ACS performance. In all the experiments of this and of the following sections we set parameter values, if not differently indicated, as follows: q_0=0.9, β=2, ρ=α=0.1, m=10, and τ_0=a very small constant (we found that a good value was $(n \cdot L_{nn})^{-1}$, where L_{nn} is the tour length produced by the nearest neighbor heuristic[2] and n is the number of cities).

In order to try to understand which mechanism ACS uses to direct the search we study how the trail–closeness product $[\tau(r,s)] \cdot [\eta(r,s)]^{\beta}$ changes at run-time. Consider Figure 2, where we show how the trail–closeness product changes with the number of steps while ants are building a solution[3] (steps refer to the Phase 2 of the ACS algorithm): the abscissa goes therefore from 1 to n, where n is the number of cities).

We consider three family of edges (see Figure 2):

1. those belonging to the last best tour (BE, Best Edges),
2. those which do not belong to the last best tour, but which recently did (TE, Testable Edges),
3. those that either have never or haven't for a long time belonged to a best tour (UE, Uninteresting Edges).

The average trail–closeness product is then computed as the average of trail–closeness values of all the edges within a family. The graph clearly shows that ACS favors exploitation of edges in BE (BE edges are chosen with probability q_0=0.9) and exploration of edges in TE (remind that, since formula (2), edges with higher trail–closeness product have a higher probability of being explored).

An interesting aspect is that while edges are visited by ants, the application of the local updating rule, formula (3), makes their trail diminish, making them less and less attractive, and favoring therefore the exploration of not yet visited edges.

Experimental observation has shown that edges in BE, when ACS achieves a good performance, will be approximately downgraded to TE after an iteration of the algorithm and that edges in TE will soon be downgraded to UE, unless they happen to belong to a new shortest tour.

In Figures 3 and 4 we report two typical behaviors of trail when the system has respectively a good or a bad performance.

---

2 To be true, any very rough approximation of the optimal tour length would do.

3 Note that the graph in Figure 2 is an abstraction of graphs obtained experimentally. Examples of these are given in Figure 3 and Figure 4.

Authorized licensed use limited to: IEEE Transactions on SMC Associate Editors. Downloaded on May 16, 2009 at 07:48 from IEEE Xplore. Restrictions apply.

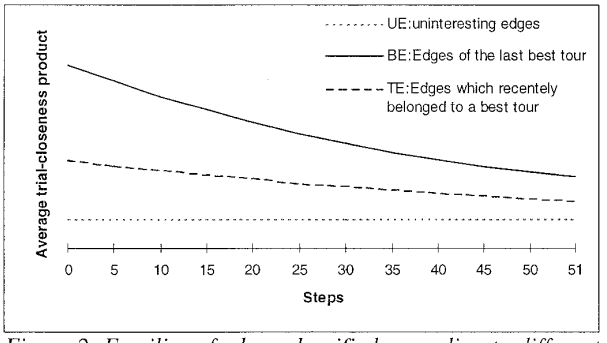

Figure 2. Families of edges classified according to different behavior with respect to the amount of trail. In this figure we show how the average trail level changes in each family during one iteration of the algorithm (i.e., during n steps).

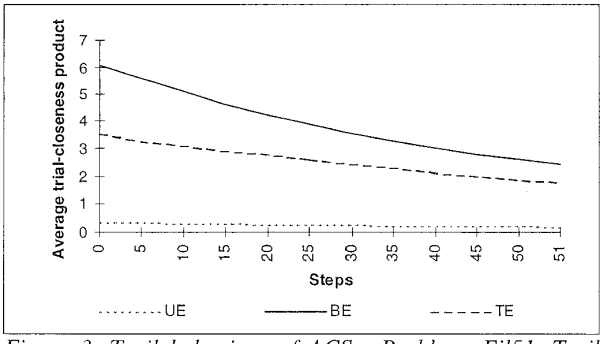

Figure 3. Trail behavior of ACS. Problem: Eil51. Trail behavior when the system performance is good. Best solution found after 1000 iterations: 426, $\alpha=\rho=0.1$.

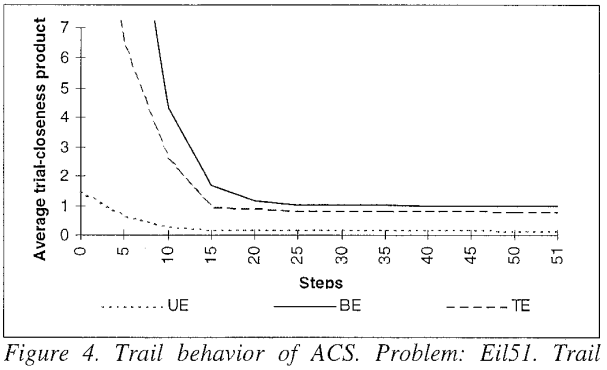

Figure 4. Trail behavior of ACS. Problem: Eil51. Trail behavior when the system performance is bad. Best solution found after 1000 iterations: 465, $\alpha=\rho=0.9$.

IV. Experiments

In this section we presents ACS results in the solution of different instances of TSP and ATSP proposed in the "First International Contest on Evolutionary Optimization"

In the following experiments we set ACS parameters in the following way: q_0=0.9, β=2, $\rho=\alpha$=0.1, m=10, and $\tau_0=(n\cdot L_{nn})^{-1}$. For each problem we performed a total number of evaluations[4] (generated tours) given by the following formula:

$$evaluations = 100 * problem\_size * problems\_type$$

where *problems_type*=100 for TSP and *problems_type*=200 for ATSP problems. For TSP problems att532, rat783, fl1577 we performed 1,000,000 evaluations.

In Table 1 and Table 2 we report the results obtained for TSP an ATSP problems. In the first column we report the problem name, the number of cities (in parentheses) and the total number of evaluations (in square brackets). In the second column we report the best result obtained by ACS out of 15 trials: we give the integer length of the shortest tour, and the number of evaluations required to find it (in square brackets). In the third column we report ACS average on 15 trials and its standard deviation in square brackets. In the fourth column we report the optimal result (for fl1577 we give, in square brackets, the known lower and upper bounds, given that the optimal solution is not known). In the last column we give the error percentage, a measure of the quality of the best result found by ACS:
100*((ACS_Best_Result - Optimal_Result)/Optimal_Result).

Table 1. ACS performance for TSP problems.

Problem (cities) [#evaluations]	ACS Best Result	ACS Average	Best Known Result	% Error
eil51 (51) [510,000]	426 [1,080]	428.06 [2.48]	426	0.0 %
kroa100 (100) [1,000,000]	21,282 [36,610]	21,420 [141.72]	21,282	0.0 %
d198 (198) [1,980,000]	15,888 [585,000]	16,054 [71.15]	15,780	0.68 %
att532 (532) [1,000,000]	28,147 [830,658]	28,522.80 [275.37]	27,686	1.67 %
rat778 (778) [1,000,000]	9,015 [991,276]	9,066 [28.25]	8,806	2.37%
fl1577 (1577) [1,000,000]	22,977 [942,000]	23,163 [116.55]	[22,137 – 22,249]	3.27÷ 3.79 %

4 In case of m agents ACS is executed for *evaluation*/m iterations.

Authorized licensed use limited to: IEEE Transactions on SMC Associate Editors. Downloaded on May 16, 2009 at 07:48 from IEEE Xplore. Restrictions apply.

Table 2. ACS performance for ATSP problems.

Problem (cities) [#evaluations]	ACS Best Result	ACS Average	Best Known Result	% Error
p43 (43) [860,000]	2,810 [16,850]	2,811.95 [1.61]	2,810	0.0 %
ry48p (48) [960,000]	14,422 [233,140]	14,565.45 [115.23]	14,422	0.0 %
ft70 (71) [1,400,000]	38,781 [996,020]	39,099.05 [170.32]	38,673	0.28 %
kroa124p (100) [2,000,000]	36,241 [536,170]	36,857.00 [521.19]	36,230	0.03 %
ftv170 (170) [3,400,000]	2,774 [939,100]	2,826.47 [33.84]	2,755	0.69 %

In these runs we implemented a slightly modified version of ACS which incorporates a more advanced data structure known as *candidate list*, a data structure normally used when trying to solve big TSP problems [9],[11]. A candidate list is a list of preferred cities to be visited; it is a static data structure which contains, for a given city i, the cl closest cities, where cl is a parameter that we set to cl =20 in our experiments. In practice, an ant in ACS with candidate list first chooses the city to move to among those belonging to the candidate list. Only if none of the cities in the candidate list can be visited (failure) then it considers the rest of the cities.

In Table 3 we report the results related to the application of different candidate list size to *eil51* problems. In the first column we report the size of the candidate list. In the second column we report the average result of ACS over 15 trials for 500 iterations with 10 agents. In the third column we report the best integer result over the same 15 trials. In the fourth column we report the average time for each trial, performed on a Sun UltraSparc1, and in the fifth column we report the average number of failures for each agents during tour construction. It is interesting to note that, the performance of the system is at best both in term of average time per trial and in term of the quality of the generated tours when cl=10.

Table 3. Comparison between candidate list size.

Candidate list size	ACS Average	ACS Best	Average Time (sec)	Number of failures
0	433.87	428	35.33	50.00
10	431.00	426	13.93	0.73
20	431.27	427	23.93	0.48
30	435.27	429	33.93	0.36
40	433.47	426	44.26	0.11
50	433.87	429	55.06	0.01

V. Discussion and Conclusions

In this paper we presented ACS a novel approach to combinatorial optimization based on the cooperation of a set of agents. The research was first inspired by a study of ant colonies behavior [4] which gave rise first to the Ant System [7], then to Ant-Q [8], an hybridization of AS with Q-learning. ACS is an extension of Ant-Q where we experimented a different local trial updating policy in order to improve the performance of the system in term of speed and quality of results (see [6] for a comparison between Ant-Q and ACS).

Although the results presented in this paper are very encouraging we intend to study a specialized version of ACS for the solution of TSP and ATSP problems. In Figure 5 we report the typical behavior of ACS during the experiments presented in this paper. Usually, the length of the best solution is improved very fast in the initial phase of the algorithm (10% of the total number of iterations). Later on, (until the 50% of the total number of iterations) new good solutions are discovered but phenomena of local stagnation starts to appear. In the last phases of the computation new improved solutions are discovered more rarely and situations of local minima appears more frequently.

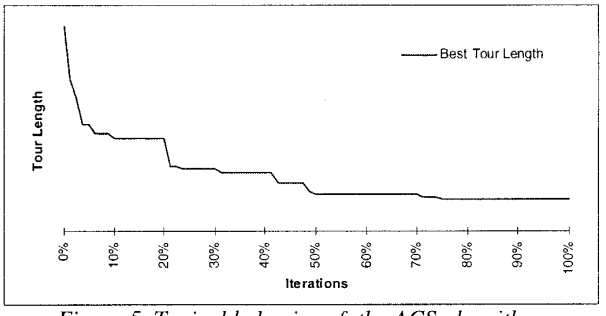

Figure 5. Typical behavior of the ACS algorithm.

We believed that, in order to escape from local minima, and to increase the speed of the search, one possible extension of ACS is the introduction of local optimization heuristics during agent's computation. Our idea is to add to the artificial ant colony some new ant explicitly dedicated to local optimization. They will try to improve the best path applying their local optimization heuristics. In the case they find a solution that decreases the length of the tour, they will apply formula (4) changing the trial on their new best tour.

An other possible improvement is related to the organization of the colony of agents. Until now, the population of agents that perform the search in parallel, is composed of identical individuals (they all have the same parameters). This limitation requires sometimes parameter recalibration in order to avoid local stagnation and to improve the speed of the search.

Authorized licensed use limited to: IEEE Transactions on SMC Associate Editors. Downloaded on May 16, 2009 at 07:48 from IEEE Xplore. Restrictions apply.

Our idea is to take inspiration from the observation of natural phenomena (and in some way from Genetic Algorithms) and to define a population of agents with different structural parameters. Then, a new mechanisms will be introduced in ACS that will allow agents which perform better than others to survive and reproduce. Analysis will be also carried out to understand the role of the parameters in order to identify when and how new agents must be introduced into the system.

VI. Acknowledgments

This work has been partially supported by a CEC Human Capital and Mobility (HCM) contract for the years 1994/1996 to Marco Dorigo.

References

[1] Colorni A., M. Dorigo and V. Maniezzo, 1991. Distributed Optimization by Ant Colonies. *Proceedings of ECAL91 - European Conference on Artificial Life*, Paris, France, F.Varela and P.Bourgine (Eds.), Elsevier Publishing, 134–142.

[2] Colorni A., M. Dorigo and V. Maniezzo, 1992. An Investigation of some Properties of an Ant Algorithm. *Proceedings of the Parallel Problem Solving from Nature Conference (PPSN 92)*, Brussels, Belgium, R.Männer and B.Manderick (Eds.), Elsevier Publishing, 509–520.

[3] Colorni A., M. Dorigo, V. Maniezzo and M. Trubian, 1994. Ant system for Job-shop Scheduling. *JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science*, 34, 1, 39–53.

[4] Denebourg J.L., J.M. Pasteels and J.C. Verhaeghe, 1983. Probabilistic Behaviour in Ants: a Strategy of Errors? *Journal of Theoretical Biology*, 105, 259–271.

[5] Dorigo M., 1992. *Optimization, Learning and Natural Algorithms*. Ph.D.Thesis, Politecnico di Milano, Italy, EU. (In Italian.)

[6] Dorigo M. and L.M. Gambardella, 1996. Ant Colony system *Tech. Rep. IRIDIA/96-01*, Université Libre de Bruxelles, Belgium, EU..

[7] Dorigo M., V. Maniezzo and A. Colorni, 1996. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part-B*, 26, 1, 29-41

[8] Gambardella L. and M. Dorigo, 1995. Ant-Q: A Reinforcement Learning approach to the traveling salesman problem. *Proceedings of ML-95, Twelfth International Conference on Machine Learning*, Tahoe City, CA, A. Prieditis and S. Russell (Eds.), Morgan Kaufmann, 252–260.

[9] Johnson D.S. and L.A. McGeoch, in press. The Travelling Salesman Problem: A Case Study in Local Optimization. In *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra (Eds.), Wiley, New York.

[10] Maniezzo V., A.Colorni and M.Dorigo, 1994. The Ant System Applied to the Quadratic Assignment Problem. *Tech. Rep. IRIDIA/94-28*, Université Libre de Bruxelles, Belgium, EU.

[11] Reinelt G., 1994. *The traveling salesman: Computational solutions for TSP applications*. Springer-Verlag.

[12] Watkins C.J.C.H., 1989. Learning with delayed rewards. *Ph. D. dissertation*, Psychology Department, University of Cambridge, England.

Authorized licensed use limited to: IEEE Transactions on SMC Associate Editors. Downloaded on May 16, 2009 at 07:48 from IEEE Xplore. Restrictions apply.