An External Partial Permutations Memory for Ant Colony Optimization

Adnan Acan

Eastern Mediterranean University, Computer Engineering Department
Gazimagusa, T.R.N.C., Via Mersin 10, Turkey
adnan.acan@emu.edu.tr

Abstract. A novel external memory implementation based on the use of partially complete sequences of solution components from above-average quality individuals over a number of previous iterations is introduced. Elements of such variable-size partial permutation sequences are taken from randomly selected positions of parental individuals and stored in an external memory called the partial permutation memory. Partial permutation sequences are associated with lifetimes together with their parent solutions' fitness values that are used in retrieving and updating the contents of the memory. When a solution is to be constructed, a partial permutation sequence is retrieved from the memory based on its age and associated fitness value, and the remaining components of the partial solution is completed with an ant colony optimization algorithm. Resulting solutions are also used to update some elements within the memory. The implemented algorithm is used for the solution of a difficult combinatorial optimization problem, namely the quadratic assignment problem, for which significant performance achievements are provided in terms of convergence speed and solution quality.

1 Introduction

Ant colony optimization (ACO) is a nature-inspired general purpose computation method which can be applied to many kinds of optimization problems [1, 2]. Among many efforts on the development of new variants of ACO algorithms toward improving their efficiency under different circumstances, recently the idea of knowledge incorporation from previous iterations became attractive and handled by a number of researchers. Mainly, these population- or memory-based approaches take their inspiration from studies in genetic algorithms (GAs). In memory-based GA implementations, information stored within a memory is used to adapt the GAs behavior either in problematic cases where the solution quality is not improved over a number of iterations, or a change in the problem environment is detected, or to provide further directions of exploration and exploitation. Memory in GAs can be provided externally (outside the population) or internally (within the population) [3].

External memory implementations store specific information within a separate population (memory) and reintroduce that information into the main population at a later moment. In most cases, this means that individuals from memory

G.R. Raidl and J. Gottlieb (Eds.): EvoCOP 2005, LNCS 3448, pp. 1–11, 2005.

are put into the initial population of a new or restarted GA. Case-based memory approaches, which are actually a form of long term elitism, are the most typical form of external memory implemented in practice. In general, there are two kinds of case-based memory implementations: in one kind, case-based memory is used to re-seed the population with the best individuals from previous generations [4], whereas a different kind of case-based memory stores both problems and solutions [5]. Case-based memory aims to increase the diversity by reintroducing individuals from previous generations and achieves exploitation by reusing individuals from case-based memory when a restart from a good initial solution is required.

Other variants of external memory approaches are provided by several researchers for both specific and general purpose implementations. Simoes and Costa introduced an external memory method in which gene segments are stored instead of the complete genomes [6, 7]. Acan et al. proposed a novel external memory approach based on the reuse of insufficiently utilized promising chromosomes from previous generations for the production of current generation offspring individuals [8].

The most common approaches using internal memory are polyploidy structures. Polyploidy structures in combination with dominance mechanisms use redundancy in genetic material by having more than one copy of each gene. When a chromosome is decoded to determine the corresponding phenotype, the dominant copy is chosen. This way, some genes can shield themselves from extinction by becoming recessive. Through switching between copies of genes, a GA can adapt faster to changing environments and recessive genes are used to provide information about fitness values from previous generations [9].

In ACO the first internally implemented memory-based approach is the work of Montgomery et al. [10]. In their work, named as AEAC, they modified the characteristic element selection equations of ACO to incorporate a weighting term for the purpose of accumulated experience. This weighting is based on the characteristics of partial solutions generated within the current iteration. Elements that appear to lead better solutions are valued more highly, while those that lead to poorer solutions are made less desirable. They aim to provide, in addition to normal pheromone and heuristic costs, a more immediate and objective feedback on the quality of the choices made. Basically, considering the TSP, if a link (r, u) has been found to lead to longer paths after it has been incorporated into a solution, then its weight $w(r, u) < 1$. If the reverse is the case, then $w(r, u) > 1$. If the colony as a whole has never used the link (r, u), then its weight is selected as 1. The authors suggested simple weight update procedures and proposed two variations of their algorithm. They claimed that the achieved results for different TSP instances are either equally well or better than those achieved using normal ACS algorithm.

The work of Guntsch et al. [11] is the first example of an external memory implementation within ACO. Their approach, P-ACO, uses a population of previously best solutions from which the pheromone matrix can be derived. Initially the population is empty and, for the first k iteration of ants, the best

solutions found in each iteration enters the population. After that, to update the population, the best solution of an iteration enters the population and the oldest one is removed. That is, the population is maintained like a FIFO-queue. This way, each solution in the population influences the decisions of ants over exactly k iterations. For every solution in the population, some amount of pheromone is added to the corresponding edges of the construction graph. The whole pheromone information of P-ACO depends only on the solutions in the population and the pheromone matrix is updated as follows: whenever a solution π enters the population, do a positive update as $\forall i \in [1,n] : \tau_{i\pi(i)} \rightarrow \tau_{i\pi(i)} + \Delta$ and whenever a solution σ leaves the population, do a negative update as $\forall i \in [1,n] : \tau_{i\sigma(i)} \rightarrow \tau_{i\sigma(i)} - \Delta$. These updates are added to the initial pheromone value τ_{init}. The authors also proposed a number of population update strategies in [12] to decide which solutions should enter the population and which should leave. In this respect, only the best solution generated during the past iteration is considered as a candidate to enter the population and the measures used in population update strategies are stated as age, quality, prob, age and prob, and elitism. In age strategy, the oldest solution is removed from the population. In quality strategy, the population keeps the best k solutions found over all past iterations, rather than the best solutions of the last k iterations. Prob strategy probabilistically chooses the element to be removed from the population and the aim is to reduce the number of identical copies that might be caused the quality strategy. Combination of age and prob strategies use prob for removal and age for insertion into the population. In elitist strategy, the best solution found by the algorithm so far is never replaced until a better solution is found.

Recently, Acan proposed two novel external memory strategies for ACO [13]. In this approach, a library of variable size solution segments cut from elite individuals of a number of previous generations is maintained. There is no particular distribution of ants in the problem space and, in order to construct a solution each ant retrieves a segment from the library based on its goodness in its parent solution, takes the end component of the segment as its starting point, and completes the rest to form a complete feasible solution. The proposed approach is used for the solution of traveling salesman problem (TSP) and the quadratic assignment problem (QAP) for which significantly better solutions are achieved compared to conventional ACO algorithms.

This paper introduces another population based external memory approach where the population includes variable-size partial permutation sequences taken from elite individuals of previous iterations. Initially, the memory is empty and an ant colony optimization algorithm runs for a small number of iterations to initialize the memory of partial permutations. Each stored sequence is associated with a lifetime and its parent's objective function value that will be used as measures for retrieving and updating partial solutions within the memory. In order to construct a solution, a particular ant retrieves a partial solution (a partial permutation sequence) from the memory based on a defined performance measure and fills in the unspecified components within it. Constructed solutions are also used to update the memory. The details of the practical implementation

are given in the following sections. The proposed ACO strategy is used to solve the well-known quadratic assignment problem for which significant performance improvements are achieved, compared to the well-known Max-Min AS algorithm, in terms of both solution quality and the convergence speed.

This paper is organized as follows. The basics of ACO-based solution construction procedures for the the quadratic assignment problem (QAP) are presented in Section 2. The proposed approach is described with its implementation details in Section 3. Section 4 covers the results and related discussions. Conclusions and future research directions are given in Section 5.

2 ACO for Quadratic Assignment Problem

In this section, the basic solution construction procedure of ACO algorithms for the solution of QAP will be briefly described. Given a set of N facilities, a set of N locations, distances between pairs of locations, and flows between pairs of facilities, QAP is described as the problem of assigning each facility to a particular location so as to minimize the sum of the product between flows and the distances. More formally, if $D = [d_{ij}]$ is the $N \times N$ distance matrix and $F = [f_{pq}]$ is the $N \times N$ flow matrix, where d_{ij} is the distance between locations i and j and f_{pq} is the amount of flow between facilities p and q, QAP can be described by the following equation:

$$\min_{\pi \in \Pi} \sum_{i=1}^{N} \sum_{j=1}^{N} d_{\pi(i)\pi(j)} f_{ij} \tag{1}$$

where Π is the set of all permutations of integers from 1 to N, and $\pi(i)$ gives the location of facility i within the current solution (permutation) $\pi \in \Pi$. The term $d_{\pi(i)\pi(j)} f_{ij}$ is the cost of simultaneously assigning facility i to location $\pi(i)$ and facility j to location $\pi(j)$.

Quadratic assignment problem belongs to the class of NP-hard combinatorial optimization problems and the largest instances that can be solved with exact algorithms are limited to instances of size around 30 [14]. Hence, the only feasible way to deal with the solution of large QAP instances is to use heuristic approaches which guarantee to reach a locally optimal solution in reasonable computation times. Several modern heuristics, mostly evolutionary or nature-inspired, are developed since 1975 and successfully applied for the solution of many provably hard optimization problems including the QAP. In this respect, ant colony optimization is successfully used for the solution of QAP and, compared to other metaheuristic approaches, better results are obtained for many difficult problem instances [15, 16, 17, 18, 19].

The QAP is one of the most widely handled problem for the illustration and testing of ACO strategies. An ACO algorithm uses a number of artificial ants for the construction of solutions such that each ant starts from a particular initial assignment and builds a solution in an iterative manner. An ant builds a solution by randomly selecting an unassigned facility and placing it into one

of the remaining free locations, until no unassigned facilities are left. In the construction of a solution, the decision of an ant for its next assignment is affected by two factors: the pheromone information and the heuristic information which both indicate how good it is to assign a facility to the free location under consideration. No heuristic information is used in the implementation of MMAS-QAP and pheromone concentration τ_{ij} refers to the desire of assigning facility of i to location j. Very detailed steps of implementation for MAX-MIN AS on the solution of QAP can be found in [16], that is also considered as the main reference for ACO-strategy and parameter value selections in the proposed approach.

3 The Proposed External Memory Approach

The basic inspiration behind the use of a partial-permutations memory is to store values of some solution components that result in above-average fitness so that they can be reused in future iterations to provide further intensification around potentially promising solutions without destroying diversification capabilities of the underlying ACO algorithm.

In the proposed approach, there are m ants that build the solutions. An external memory of variable-size partial solution sequences from a number of elite solutions of previous iterations is maintained. Initially the memory is empty and a number ACO iterations is performed to fill in the memory. In this respect, after every iteration, the top k-best solutions are considered and a number of randomly positioned solution components are selected from these individuals, one sequence per solution, and stored in the memory. Uniform probability distribution is used in all random number generations. The memory size M is fixed and a number of ACO iterations are repeated until the memory is full. Generation of a partial permutation from a given parent solution is illustrated in Figure 1. Simply, an arbitrary number of solution components are selected from the parent solution and stored as a partially complete permutation within the memory. Since we need the locus of the stored solution components when they will be used, partial permutation sequences are stored as fixed-length sequences where unselected parts are filled with zeros. With this description of individual partial permutation sequences, the external memory is an MxN array where M is the number of elements and N is the length of a solution.

When a new solution is to be created, a partially complete permutation is retrieved from the memory selects a partial permutation sequence from the memory based on its lifetime and fitness values. In this respect, each partial solution sequence i in the memory is assigned with a performance score as $P\_Score(i) = Fitness(i) + Age^2(i)$. With this assignment, higher P_Score values are given to those with higher age attributes in cases of closer fitness values. The main idea behind this selection strategy is to give higher priority of being used to older sequences before they are removed from the memory. This also helps to provide further diversification by trying some of the previously promising search directions.

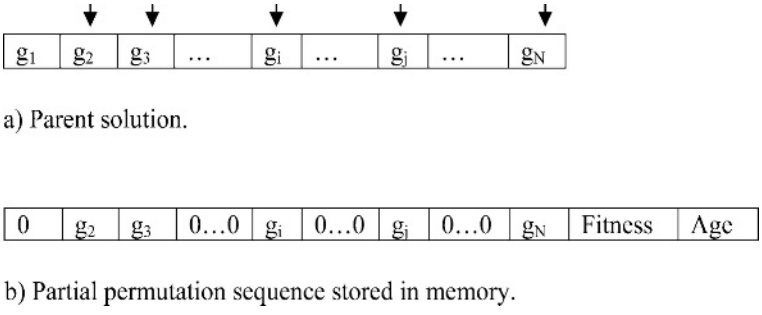

Fig. 1. a) A parent solution and b) the partial permutation sequence containing randomly located solution components stored in the memory

After the initial phase, MAX-MIN AS algorithm works in conjunction with the implemented external memory as follows: there is no particular assignment of ants over the problem space and, in order to construct a solution, each ant selects a partial permutation sequence from memory using a tournament selection strategy and the starting point for the corresponding ant becomes the first unassigned position of the partial solution. That is, the ant builds the rest of the solution by assigning the unassigned components of the retrieved partial solution in a random order. In the selection of partial solution sequences from memory, each ant makes a tournament among Q randomly selected partial permutation sequences and the best one is selected as the seed to start for construction. The solution construction procedure is the same as the one followed in MAX-MIN AS algorithm. After all ants complete their solution construction procedures, pheromone updates are carried in exactly the same way it is done in MAX-MIN AS algorithm. Based on these descriptions, the proposed ACO algorithm can be put into an algorithmic form described in Algorithm 1.

In memory update procedure, the solutions constructed by all ants are sorted in ascending order and the top k-best are considered as candidate parents from which new partial permutation sequences will be extracted and inserted into memory. One random-length and randomly-positioned partial permutation sequence is taken from each elite solution and it replaces either the worst of those memory elements having worse fitness values than of the extracted sequence or the worst of those oldest sequences within the memory.

4 Experimental Results

To study the performance of the proposed external memory ACO approach, it is compared with a well-known classical implementation of ACO algorithms, namely the MAX-MIN AS algorithm, for the solution of a widely known and difficult combinatorial optimization problem, the QAP. The QAP problem instances used in evaluations are taken from the web-site http://www.opt.math.tu-

Algorithm 1: ACO with an external partial permutations memory

1. Initialize the external memory.
2. Repeat
   (a) For each of m ants, find an initial assignment by assigning a randomly selected facility to a randomly selected location.
   (b) Let all ants construct a solution iteratively by randomly selecting an unassigned facility and placing it into one of the free locations.
   (c) Compute the objective function values for all the constructed solutions.
   (d) Use local search to improve the solutions of a number of elite ants.
   (e) Sort the solutions in ascending order of their objective function values.
   (f) Consider the top k-best and extract a randomly-positioned and randomly-sized partial permutation from each solution and insert the partial solution sequences into the memory. Also, store the length of each sequence and the cost of its parent. Meanwhile, set the age of the inserted element to 1.
3. Update the pheromone matrix.
4. Until the memory is full.
5. ITER=1
6. Repeat
   (a) Let all ants select a partial solution sequence from the external memory using tournament selection.
   (b) Let all ants construct a solution starting from the partial permutation (sequence) they retrieved from the memory.
   (c) Compute the objective function values for all the constructed solutions.
   (d) Use local search to improve the solutions of a number of elite ants.
   (e) Update pheromone matrix.
   (f) Increase ages of memory elements by 1.
   (g) Update memory.
   (h) ITER=ITER+1.
7. Until ITER > Max_Iter.

graz.ac.at/qaplib. These selected problems are representative instances of different problem groups, commonly handled by several researchers, and have reasonable problem sizes for experimental evaluations.

The ACO algorithm used with which the external memory is integrated is MMAS-QAP [16] which is a well-known and provably successful algorithm for the solution of QAP instances. All programs are prepared using Matlab 6.5 programming language and the computing platform is a 1 GHz PC with 256 MB of memory. Each experiment is performed 20 times over 1000 iterations. The relative percentage deviation, RPD, is used to express the accuracy of experimental results. RPD is a measure of the percentage difference between the best known solution and an actual experimental result. RPD is simply calculated by the following formula.

$$RPD = 100 \times (\frac{Cost_{Actual} - Cost_{Best}}{Cost_{Best}}) \quad (2)$$

Details of implementations and the sets of parameter values used are given below.

In the implementation of the proposed ACO strategy, the memory size M and the number of ants in ACO are set equal to the number of locations (facilities) in QAP instances. The tournament size $Q = 7$, and the top $k = 0.05 * M$ solutions are used in updating the memory. The local search algorithm used is 2.5-opt and 25 elite solutions are allowed to be processed within the local search. Maximum lifetime of individuals in the shared-memory is taken as 5 iterations.

In the implementation of the ACO algorithms, the two sets of parameter values taken from well-known publications are used as follows [16]: First set of parameters are used in initializing the partial permutations memory and a highly biased probabilistic selection scheme is followed in the selection of solution components. In this respect, the next element for which $\tau^{\alpha}.\eta^{\beta}$ is maximal is selected with probability $q_0 = 0.9$. Together with the local search used, memory elements are initialized with partially complete permutations extracted from potentially promising solutions. Other parameter values used in this phase are as follows: $\rho = 0.1$, $\alpha = 1$, $\beta = 2$, $\tau_{init} = 1/n$, $\tau_{max} = 3.0$. Parameters used during the normal course of ACO algorithms are $\rho = 0.02$, $\alpha = 1$, $\beta = 2$, $\tau_{max} = 1/(\rho.L_{Best})$, $\tau_{init} = \tau_{max}$, and $\tau_{min} = 1/(2n)$.

Stagnation in ACO is detected when

$$\sum_{\tau_{ij}} min(\tau_{max} - \tau_{ij}, \tau_{ij} - \tau_{min}) \tag{3}$$

is less than 10^{-5}. In this case, ACO is restarted by setting all elements of the pheromone matrix to τ_{max}.

Experimental results for the solved QAP instances are given in Table 1.
From Table 1, it can be observed that the proposed ACO strategy with an external partial permutations memory performs better than the MAX-MIN AS supported with the same local search algorithm, in all problem instances. In addition, the effectiveness of the external memory and the implemented memory management approaches is observed much more clearly for larger problem instances. This is an expected result because partial solutions from potentially promising solutions provide more efficient intensification for more difficult problems. The main drawback of the proposed approach is the long running times. It is a well-known fact that Matlab is a useful programming language to prepare prototypes, however it can be very slow in the execution of loops, which is also the case in the presented implementations. The CPU times with the used hardware platform change from 15 minutes (for the wil50 QAP instance) to 4 hours (for the tai100a QAP instance). However, these CPU time are smaller than those required for MMAS algorithm within the same computing platform, programming language, and algorithm parameters. For example, CPU times for MMAS change from 22 minutes (for the wil50 QAP instance) to 6 hours (for the tai100a QAP instance). Implementations with a faster programming language is currently an ongoing but yet not completed task.

Table 1. Results for Max-Min AS, and the proposed approach on QAP instances

Strategy	Instance	RPD		
		Min	Average	Max
Max-Min AS	wil50	1.831	3.823	4.412
	wil100	3.512	5.711	7.3
	tai100a	8.910	13.36	18.44
	tai100b	10.117	15.19	19.5
	tai35b	4.218	9.3	14.5
	tai40b	4.887	6.1	11.7
	tai50b	5.628	7.6	13.6
	tai60b	7.917	9.5	17.32
	tai80b	13.226	17.5	28.49
	tai100a	21.11	25.6	33.51
	sko100a	17.6	20.3	24.1
	sko100b	19.44	22.7	26.8
MX-MIN AS with a Partial Permutations Memory	wil50	0.98	2.61	3.73
	wil100	2.49	3.26	4.73
	tai100a	4.91	8.32	11.94
	tai100b	7.96	10.27	14.81
	tai35b	3.21	6.73	10.66
	tai40b	3.16	5.25	8.86
	tai50b	3.83	5.91	8.45
	tai60b	4.92	7.53	11.74
	tai80b	7.74	10.16	15.36
	tai100a	16.49	21.71	27.45
	sko100a	14.51	19.11	22.35
	sko100b	16.43	20.21	24.81

5 Conclusions and Future Research Directions

In this paper a novel ACO strategy using an external memory of partial permutations from elite solutions of previous iterations is introduced. The stored partial permutations are used in the construction of solutions in the current iteration to provide further intensification around potentially promising solutions. Using partially constructed solutions also improve time spent in the construction of solutions since part of the solution is already available. Performance of the proposed external memory approach is tested using several instances of a well-known hard combinatorial optimization problems.

From the results of case studies, it can easily be concluded that the proposed ACO strategy performs better than Max-Min AS algorithm in terms of the convergence speed and the solution quality. It can easily be observed that, even though the proposed strategy better than the MAX-MIN AS strategy, its real effectiveness is seen for larger size problems. This is an expected result because information incorporation from previous iterations should be helpful for more difficult problem instances. The proposed ACO strategy is very simple to

implement and it does not bring significantly additional computational or storage cost to the existing ACO algorithms.

Further investigation of the proposed ACO strategy for the solution of other difficult problems, particularly the non-stationary optimization problems, may be considered as a direction for future studies.

References

1. Dorigo, M., Caro, G.D., Gambardella, L.M.: Ant algorithms for distributed discrete optimization, Artificial Life, Vol. 5, (1999), 137-172.
2. Dorigo, M., Caro, G.D.,: The ant colony optimization metaheuristic, In Corne, D., Dorigo, M., Glover, F. (eds.): New ideas in optimization, McGraw-Hill, London, (1999), 11-32.
3. Eggermont, J., Lenaerts, T.: Non-stationary function optimization using evolutionary algorithms with a case-based memory, Technical report, Leiden University Advanced Computer Science (LIACS) Technical Report 2001-11.
4. Ramsey, C.L., Grefenstette, J. J.: Case-based initialization of GAs, in Forest, S., (Editor): Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, CA, (1993), 84-91.
5. Louis, S., Li, G.: Augmenting genetic algorithms with memory to solve traveling salesman problem, Proceedings of the Joint Conference on Information Sciences, Duke University, (1997), 108-111.
6. Simoes, A., Costa, E.: Using genetic algorithms to deal with dynamic environments: comparative study of several approaches based on promoting diversity, in W. B. Langton et al. (eds.): Proceedings of the genetic and evolutionary computation conference GECCO'02, Morgan Kaufmann, New York, (2002), 698.
7. Simoes, A., Costa, E.: Using biological inspiration to deal with dynamic environments, Proceedings of the seventh international conference on soft computing MENDEL'2001, Czech Republic, (2001).
8. Acan, A., Tekol, Y.: Chromosome reuse in genetic algorithms, in Cantu-Paz et al. (eds.): Genetic and Evolutionary Computation Conference GECCO 2003, Springer-Verlag, Chicago, (2003), 695-705.
9. Lewis, J., Hart, E., Ritchie, G.: A comparison of dominance mechanisms and simple mutation on non-stationary problems, in Eiben, A. E., Back, T., Schoenauer, M., Schwefel, H. (Editors): Parallel Problem Solving from Nature- PPSN V, Berlin, (1998), 139-148.
10. Montgomery, J., Randall, M: The accumulated experience ant colony for the traveling salesman problem, International Journal of Computational Intelligence and Applications, World Scientific Publishing Company, Vol. 3, No. 2, (2003), 189-198.
11. Guntsch, M., Middendorf, M.: A population based approach for ACO, in S. Cagnoni et al., (eds.): Applications of Evolutionary Computing - EvoWorkshops2002, Lecture Notes in Computer Science, No:2279, Springer Verlag, (2002), 72-81.
12. Guntsch, M., Middendorf, M.: Applying population based ACO for dynamic optimization problems, in M. Dorigo et al., (eds.): Ant Algorithms - Third International Workshop ANTS2002, Lecture Notes in Computer Science, No:2463, Springer Verlag, (2002), 111-122.
13. A. Acan. An External Memory Implementation in Ant Colony Optimization. In *Ant Colony Optimization and Swarm Intelligence - ANTS2004*, page 73-82, Brussels, September 2004.

14. Stützle, T., Fernandes, S.: New benchmark instances for the QAP and the experimental analysis of algorithms. In: Gottlieb, J., Raidl, G. R. (eds.): Evolutionary Computation in combinatorial Optimization - EvoCOP 2004, Springer-Verlag, (2004), 199-209.
15. Stützle, T., Dorigo, M.: ACO Algorithms for the Quadratic Assignment Problem. In: Corne, D., Dorigo, M., Glover, F. (eds.): New Ideas in Optimization, McGraw-Hill, (1999), 33-50.
16. Stützle, T.: MAX-MIN ant system for the quadratic assignment problem. Technical Report AIDA-97-04, Darmstadt University of Technology, Computer Science Dept., Intellectics Group, (1997).
17. Stützle, T.: Iterated local search for the quadratic assignment problem. Technical Report AIDA-99-03, Darmstadt University of Technology, Computer Science Dept., Intellectics Group, (1999).
18. Maniezzo, V.: Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS Journal on Computing, 11(4), (1999), 358-369.
19. Maniezzo, V., Colorni, A., Dorigo, M.: The ant system applied to the quadratic assignment problem. Technical Report IRIDIA/94-28, Universite Libre de Bruxelles, (1994).