2-1-2014

Ant Colony Optimization approaches for the Sequential Ordering Problem

Ahmed Mohamed Alaa El-din Ezzar

THE AMERICAN UNIVERSITY IN CAIRO

School of Sciences and Engineering

Ant Colony Optimization Approaches for the Sequential Ordering Problem

A thesis submitted to
Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
Master of Science

by Ahmed Ezzat
under the supervision of Dr. Ashraf Abdelbar
July 2013

Abstract

We present two algorithms within the framework of the Ant Colony Optimization (ACO) metaheuristic. The first algorithm seeks to increase the exploration bias of Gambardella et al.'s (2012) Enhanced Ant Colony System (EACS) model, a model which heavily increases the exploitation bias of the already highly exploitative ACS model in order to gain the benefit of increased speed. Our algorithm aims to strike a balance between these two models. The second is also an extension of EACS, based on Jayadeva et al.'s (2013) EigenAnt algorithm. EigenAnt aims to avoid the problem of stagnation found in ACO algorithms by, among other unique properties, utilizing a selective rather than global pheromone evaporation model, and by discarding heuristics in the solution construction phase. A performance comparison between our two models, the legacy ACS model, and the EACS model is presented. The Sequential Ordering Problem (SOP), one of the main problems used to demonstrate EACS, and one still actively studied to this day, was utilized to conduct the comparison.

Thesis Supervisor: Ashraf Abdelbar

Acknowledgments

I would like to express my gratitude and appreciation to my supervisor Dr. Ashraf Abdelbar for his outstanding support. I would also like to extend my gratitude to Dr. Jayadeva, my colleagues, and the department for their cooperation.

Contents

1 Overview and Motivation **9**

2 Background **11**

2.1 Ant Colony Optimization 11

2.2 Ant System 14

2.3 Ant Colony System 16

2.4 Sequential Ordering Problem 18

3 Literature Review **19**

3.1 EigenAnt 19

3.1.1 Analysis of EigenAnt 21

3.2 Hybrid Ant System and the SOP–3–exchange local optimization procedure 24

3.3 Enhanced Ant Colony System 31

4 Proposed algorithms **33**

4.1 Probabilistic EACS 33

4.2 EigenAnt-Based Ant System 34

5 Experimental Results **36**

5.1 Experimental Setup 36

5.1.1 Problem Instances 36

5.1.2 Algorithm Parameters 36

5.2 Empirical Data 37

5.2.1 Results 37
5.2.2 Statistical Significance 50
5.3 Experiments on the exploitation parameter q_0 52

6 Analysis **56**

7 Future work **58**

List of Figures

2-1 Skeleton pseudo–code of the ACO metaheurisitic 13

3-1 EigenAnt algorithm 20
3-2 Illustration of the 2–*exchange* operation 26
3-3 Illustration of the *path–preserving*–3–*exchange* operation 26
3-4 Local search pseudo-code 29
3-4 Local search pseudo-code 30

5-1 Performance Comparison - Cost 100 - Precedence 1 40
5-2 Performance Comparison - Cost 100 - Precedence 15 40
5-3 Performance Comparison - Cost 100 - Precedence 30 41
5-4 Performance Comparison - Cost 100 - Precedence 60 41
5-5 Performance Comparison - Cost 1000 - Precedence 1 41
5-6 Performance Comparison - Cost 1000 - Precedence 15 42
5-7 Performance Comparison - Cost 1000 - Precedence 30 42
5-8 Performance Comparison - Cost 1000 - Precedence 60 42
5-9 Performance over time - Cost 100 - Precedence 1 43
5-10 Performance over time - Cost 100 - Precedence 15 43
5-11 Performance over time - Cost 100 - Precedence 30 44
5-12 Performance over time - Cost 100 - Precedence 60 44
5-13 Performance over time - Cost 1000 - Precedence 1 44
5-14 Performance over time - Cost 1000 - Precedence 15 45
5-15 Performance over time - Cost 1000 - Precedence 30 45
5-16 Performance over time - Cost 1000 - Precedence 60 45

5-17 Percentage Improvement Over Time: PEACS vs. EACS 50
5-18 Percentage Improvement Over Time: EAAS vs. EACS 50

List of Tables

5.1 Experimental Results 38
5.2 Percentage improvement of PEACS over EACS 46
5.3 Percentage improvement of EAAS over EACS 48
5.4 Results of Wilcoxon Signed-ranks Tests 53
5.5 Results of q_0 experiment 53
5.6 Results of Wilcoxon Signed-ranks Tests for q_0 experiment 55

Chapter 1

Overview and Motivation

Ant Colony Optimization (ACO) [17] is a relatively new algorithm family. More accurately, it is a meta–heuristic, an algorithmic framework that can be adapted to various problems. The first ACO algorithm, Ant System [16], was proposed by Marco Dorigo in 1992. The traveling salesman problem (TSP) was used to demonstrate its performance. While the results did not compare to the state-of-the-art algorithms, the Ant System would later stimulate further research, and many ACO variants were devised with the aim of solving difficult problems, specifically combinatorial optimization (CO) problems, with the best balance between speed and accuracy possible.

Combinatorial optimization problems are ones that require finding the best solution to the problem among ones that satisfy a group of pre-defined constraints. A solution is considered "best" if it is the one with the highest (or lowest, depending on the problem) fitness value among the group of feasible solutions. Well known CO problems include TSP, scheduling problems (Job-shop, group-shop, etc...), vehicle routing, assignment problems (quadratic, generalized, etc...), set problems (set partition, set covering, maximum independent set, etc...).

One problem that has been focused upon for quite some time by various researchers is the sequential ordering problem (SOP). Formally, it is the problem of finding a minimum weight Hamiltonian path on a directed graph with weights on the arcs and the nodes, subject to precedence constraints among nodes. A precedence constraint defined between two nodes x and y would mean that x has to come before y in the

final solution. Another way of describing would be asymmetric TSP (ATSP) with restrictions on the returned tour enforcing that certain nodes be visited before others.

The problem of ordering jobs according to the cost of performing one before the other and to restrictions dictating the couple-wise relative order of those jobs (i.e.: production planning) can be modeled as an instance of SOP. The problem of determining in what order to visit clients given similar descriptions of costs and restrictions (i.e.: vehicle routing) can also modeled as an SOP instance.

Research into this problem spans decades, and is still ongoing. Gambardella et. al's Enhanced Ant Colony System (EACS) [21] was published as recently as 2010, and at the time was the state-of-the-art method for solving SOP, if not till now. A diverse range of methodologies were used to solve it as well, ranging from genetic algorithms to particle swarm optimization.

Since the original Ant System (AS), many ACO algorithms emerged, including the MAX-MIN Ant System (MMAS) [35], and the Best-Worst Ant System (BWAS) [13], among others. A new ACO algorithm was introduced by Jayadeva et. al in 2013 called the EigenAnt algorithm [1]. It has a number of unique properties. First, by default, it only utilizes a single ant. Second, the ant chooses between entire paths, rather than edges. Third, the choice is based only the amount of pheromone on the paths. No heuristic value is associated with the path. Finally, only pheromone on the selected path is updated. No other pheromone trail is affected.

Given how relevant research on SOP still is today, we attempted to contribute to this field, and in this thesis we propose two algorithms. The first, probabilistic EACS (PEACS) [3], is a modified variation of EACS that aims to improve its performance by increasing its bias towards exploration. The second, EigenAnt Ant System (EAAS) [2], is a new algorithm based on EACS and the EigenAnt algorithm. We provide comparisons between our new algorithms, EACS and the legacy ACS.

Chapter 2

Background

2.1 Ant Colony Optimization

Combinatorial optimization problems involve the assignment of values to discrete variables that provide the optimal solution quality with respect to a given objective function. The search may also be subject to a set of constraints. Examples of this type of problem include minimum-cost delivery planning, job assignment, network data packet routing, and the traveling salesman problem, among others.

Ant Colony Optimization [17] is a metaheuristic devised by Marco Dorigo in 1992 [16] to tackle this category of problems. It is based on the behaviour of real-life ants. When ants leave their nest to search for food, they experiment with the multiple paths available to be traversed in order to reach it. While doing so, the ants deposit pheromone. The pheromone helps the ants favor one path over another whenever multiple directions present themselves. Pheromone deposit, along with its evaporation over time, helps to identify shorter paths to reach the food source.

For example, assume we have two paths of unequal length, and that two ants begin to traverse the two paths simultaneously. An ant taking the shorter path will reach the source first. On deciding which path to take back, it will notice that the pheromone trail on the path it took is stronger, especially considering that the other ant still has not reached the source yet. Consequently, it will more likely take the short path back to the nest, depositing more pheromone on the way. On reaching the nest,

it will have made the trail on the short path even stronger. Note also that the other ant will probably take the short path on the way back as well, further enforcing the trail on the path. As time progresses, all ants will lean quite heavily towards taking the short path. What we described is an example of indirect local communication between agents in the form of pheromone trails left by ants to influence the choices of the other ants. This type of communication is called Stigmergy.

As aforementioned, ACO is a metaheuristic, a general-purpose heuristic method encapsulating a problem-specific heuristic to direct it towards promising reasons of the solution space of a problem instance [17]. In other words, it is a generic algorithmic framework which can be adapted to various optimization problems with relatively little effort. Other examples of metaheuristics include simulated annealing, tabu search, and evolutionary computation.

Metaheuristics can be said to fall under the category of approximate algorithms, ones that trade optimality for solution calculation efficiency. On the flip side are exact algorithms, ones that guarantee to reach an optimal solution, but at worst require an exponential amount of time for the majority of combinatorial optimization problems.

The ACO metaheuristic consists of three main procedures: **ConstructAntsSolutions**, **UpdatePheromones**, and **DaemonActions**. In **ConstructAntsSolutions**, members of the ant colony concurrently construct solutions in incremental fashion. Each of them selects the next component of their solution according to the pheromone trail levels and heuristic information. Solutions are evaluated during or after construction to calculate the modifications to be made to the pheromone levels in **UpdatePheromones**.

UpdatePheromones handles the modification of pheromone trails on edges. Naturally, this is a critical part of ACO, as pheromone deposits try to ensure that edges that were part of good solutions will be reused, while pheromone evaporation helps in letting the colony forget unused edges. The act of forgetting unused edges helps to direct the algorithm away from unpromising regions of the search space.

The last procedure, **DaemonActions**, deals with centralized, problem specific actions. These include applying a local optimization method to some or all the ants'

solutions, and selecting a subset of the ants to deposit additional pheromone. The following demonstrates ACO in pseudo-code form:

```
Set parameters & initialize pheromone trails
while (end–condition = false) do
    ConstructAntSolutions
    LocalSearch
    UpdatePheromone
end while
```

Figure 2-1: Skeleton pseudo–code of the ACO metaheurisitic

It is important to note that the three procedures are not necessarily executed in sequential order. An implementation of ACO may opt to interleave operation between them while utilizing synchronization mechanisms to coordinate their actions. In the Ant Colony System, for example, pheromone trails are updated during solution construction rather than after their completion. A combinatorial optimization problem (S, f, Ω) can be formulated as follows:

1. S is the set of candidate solutions
2. f is the objective function used to calculate the quality of a solution $s \in S$
3. Ω is the set of constraints. For a dynamic problem, one where constraints may change over time, $\Omega(t)$ would be the set of constraints at time t
4. A finite set $C = c_1, c_2, ..., c_n$ of components is given, where n is the number of components
5. A set of all possible states X where $x \in X = \langle c_i, c_j, c_k, ... \rangle$ is a sequence of finite length defined over the elements of C
6. The set of feasible states X^*. A state is considered feasible if it is not impossible to reach a feasible solution from this state as dictated by Ω. The feasibility check here is weak as it does not fully guarantee that the state will lead to a feasible solution

7. The set of S^* of optimal solutions

8. The cost $g(s,t)$ of a solution s at time t. Usually, $g(s,t) = f(s,t)$

9. If needed, $J(x,t)$ is the cost of a state x at time t.

Using this formulation, a set of artificial ants can construct solutions by performing randomized traversals of a completely connected graph $G(C,L)$, with the set of components C being the nodes, and L being the set of connections fully connecting the components. Constraints may be enforced during the tour construction or after reaching a candidate solution.

2.2 Ant System

Perhaps the most well-known example of a CO problem is the travelling salesman problem. Indeed, it was the problem utilized by Dorigo to demonstrate Ant System, the first ACO algorithm [16]. We now show how to apply ACO to TSP.

In TSP, we attempt to find the smallest tour that starts from a city, visits every other city exactly once, and returns to the first city. By default, the distances between cities are direction independent, whereas in asymmetric TSP they are not.

The construction graph that will be used by the ant colony will be similar to the TSP graph, with cities being the components, and the connections being the arcs. For constraints, we only have one, that being that each city has to be visited once. As such, at each construction step, an ant will only be able to select from the list of cities that have not been visited yet. On selecting a city to visit, an ant will utilize the pheromone trails and the heuristic information, which in the case of TSP edges is the inverse of the length of the edge connecting the current city to the city being evaluated.

With the problem now represented in a format solvable by an ACO algorithm, we now proceed to elaborate on the details of the Ant System. At first, all pheromone trails are set to an initial value $= m/C^{nn}$, where m is the number of ants in the colony, and C^{nn} is the cost of an arbitrary solution found by a tour construction procedure.

In Dorigo's case, a nearest-neighbour heuristic is used. The value of the initial trail is quite important as setting it too low would bias the system towards the first tours it finds, while setting it too high would force the system to go through a lot of iterations so that enough of the trails evaporate to allow the system converge towards a solution space.

During solution construction, all ants build their tours concurrently. At each construction step, they pick the next city to visit according to a probabilistic choice rule. Formally, an ant k standing at city i at time t will select the city j from its neighbourhood of unvisited cities N_i^k with the following probability:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta} \tag{2.1}$$

$\tau_{ij}(t)$ represents the amount of pheromone on the edge (i, j), while η_{ij} represents the edge's heuristic value. In TSP, $\eta_{ij} = 1/d_{ij}$, where d_{ij} is the edge length. The parameters α and β represent the relative importance of the pheromone trails and heuristic information in guiding the ant's choice. While there is no restriction on the choices for values for α aside from ranging from zero to one, assigning a value greater than one to β has been found to quickly lead to stagnation, a situation where ants repeatedly give the same solutions.

When all ants finish constructing their solutions, the pheromone trails on the edges are updated. First, all edges are reduced by a constant factor according to the following rule:

$$\tau_{ij}(t) = (1 - \rho)\tau_{ij}(t) \tag{2.2}$$

ρ is the pheromone evaporation rate, and it takes on positive values from 0 to 1. This controls how quickly AS forgets previous decisions, especially bad ones. After evaporation, each ant will deposit pheromone on the edges present in its solution as follows:

$$\tau_{ij}(t) = \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}(t)^k \tag{2.3}$$

$\Delta\tau_{ij}(t)^k$ is the amount of pheromone to be deposited by ant k on the edge (i, j), and it is equal to $1/C^k$, where C^k is the length of the tour generated by ant k. The better the solution, the more pheromone is deposited on the edge, giving it a higher chance of selection in the next iteration, in the hope that it would lead to an even better solution.

Other ACO algorithms such as Elitist Ant System, Rank Based Ant System and the MAX-MIN Ant System [35] differ in the way pheromone trails are updated. The differences include allowing only a subgroup of the ants to update the trails as well as changes in the actual update rule itself. ACO algorithms such as the Ant Colony System [15], Approximate Non-deterministic Tree Search, and the Hyper-Cube Framework for ACO on the other hand include new ideas not present in AS.

2.3 Ant Colony System

In ACS, ants choose the next node to visit using a pseudo-random rule, rather than a fully random one. With a probability of q_0, an ant will select the most attractive as dictated by the pheromone trails and heuristic information. Otherwise, an ant will utilize the same probabilistic choice rule defined above. The selection rule is defined as follows:

$$j = \begin{cases} \arg\max\{Q(i,l)\} & q \leq q_0 \\ J & q > q_0 \end{cases} \tag{2.4}$$

J is a random variable whose value is calculated according to the probabilistic choice rule defined above, while q is random variable uniformly distributed from [0,1]. Q is an edge quality function defined as follows:

$$Q(i,j) \leftarrow [\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta} \tag{2.5}$$

The value of q_0 affects how exploitative the ACS algorithm is. Increasing it will force ants to depend more on previous history, while a lower value helps in letting the system be more explorative. A more explorative system however runs the risk of generating worse solutions and perhaps becoming unable to find better solutions. Another difference in ACS from AS is how pheromone trails are updated. There are two main differences. First, pheromone trails are updated during tour construction using the following rule:

$$\tau_{ij}(t) = (1 - \eta)\tau_{ij}(t) + t_0 \tag{2.6}$$

τ_0 is the initial of the pheromone trails. The actual value is problem specific. Against TSP and the Sequential Ordering Problem, a value of $1/(n * C^{nn})$ was used by Dorigo and Gambardella, respectively [16, 20]. Applying the rule allows the remaining ants to consider other options for the node choice in their solutions, thereby increasing exploration and avoiding stagnation. Second, at the end of the construction phase, only the best solution found so far by the algorithm is used to update the pheromone trails. This is done using the following rule:

$$\tau_{ij}(t) = (1 - \rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}^{bs} \tag{2.7}$$

Here, $\tau_{ij}^{bs} = 1/C^{bs}$, where C^{bs} is the cost of the best solution found so far. Only the edges that are part of the best solution are updated. Their corresponding trails, as a result of the evaporation parameter ρ, becomes a weight average between their old value and the amount of pheromone deposited.

An interesting result of applying the two rules together is that an implicit range is enforced on the pheromone trails. Their values can only fall between t_0 and $1/C^{bs}$. Such a limitation is present in another ACO algorithm, MMAS, although in that algorithm the limitation is more explicit. After pheromone update, all trails are checked to make sure they are still in range, and are modified to the nearest limit if they are not.

2.4 Sequential Ordering Problem

The Sequential Ordering Problem is a generalization of the Asymmetric Travelling Salesman Problem. It can be used to represent real–world problems such as production planning, vehicle routing and transportation problems in flexible manufacturing systems [9, 19, 31–33]. Various methodologies have been investigated for this problem, including genetic algorithms [12, 33], a parallelized roll–out algorithm [22], and the hybrid ant system, an ACS algorithm coupled with a special local search procedure [20]. Montemanni *et al.* built upon the hybrid ant system by incorporating a heuristic manipulation technique [27–29], and later Anghinolfi *et al.* investigated a discrete particle swarm optimization method [7, 8].

An instance of the SOP consists of a set of nodes V, a start node $v_s \in V$, a finish node $v_f \in V$, a set of weighted directed edges E where v_s is not allowed to have incoming edges and v_f is not allowed to have outgoing edges, and a set of precedence constraints. A precedence constraint is set between two nodes v_1 and v_2 if it is required that v_1 be visited before v_2. The objective is to construct a minimum–cost path that starts from v_s, passes through all nodes, and ends at v_f without violating any constraint.

In applying ACO to SOP, τ would be a two–dimensional $|V| \times |V|$ array; the entry τ_{ij} represents the extent to which the collective wisdom of the colony is inclined towards the edge e_{ij}. A common choice for the η_{ij} for the TSP is the reciprocal of the distance d associated with the edge e_{ij}, $\eta_{ij} = 1/d(e_{ij})$. Since SOP can be considered to be a generalization of the ATSP, we can use this measure as the heuristic desirability of edges in SOP [20, 21].

Chapter 3

Literature Review

3.1 EigenAnt

A recently proposed ACO algorithm is the EigenAnt algorithm [1]. It attempts to solve the problem of stagnation in ACO in general, or, in its authors words, ACO's lack of plasticity. Over time, should an ACO algorithm fail to improve on a solution quickly enough, it becomes increasingly difficult and eventually almost impossible to for ants to construct better solutions. Even if an ant does find a better solution, a lot of pheromone will have built up by then on the edges of the previous solution, and as such ants will tend to use its edges a lot more.

In EigenAnt, a single ant is charged with the task finding the shortest of paths available to take from a source to a destination. During every iteration, it randomly chooses a path and accordingly the pheromone trail on that path is updated. The following equation is used to update the pheromone on a path i that has been chosen by the ant at iteration t:

$$\tau_i^{t+1} \leftarrow (1-\alpha)\tau_i^t + \beta d_i p_i^t \tag{3.1}$$

where α is the evaporation rate, β is a weighting parameter, and $d_i = 1/f(L_i)$, where f is a monotonically decreasing function of L_i, the length of path i. In addition, p_i^t is the probability of taking a path i in iteration t, and is equal to $C_i^t / \sum_{i=1}^{n} C_i^t$, the

amount of pheromone on path i at iteration t divided by the total sum of all trails on all paths at iteration t. After an ant constructs a path, the pheromone trail on that path is updated.

As we can see, two main characteristics differentiate EigenAnt from most if not all ACO algorithms. First, path selection does not take path length into account at all. Only pheromone trail information plays any part in this decision. Second, only the pheromone trail on the selected path is updated. No other trail is affected. So rather than employing an evaporative pheromone update model, EigenAnt utilizes selective removal.

In pseudo-code, the EigenAnt algorithm is as follows:

procedure EIGENANT(L)

Initialize trip counter vector $J = (J_i) = 0$.

Initialize total ant trip counter $t = 1$. Choose $J_{max} \in \mathbb{N}$, the total number of ant trips

Initialize path i with pheromone concentration: $C_i^t = (1/n), i = 1, ..., n$

Initialize probability of choosing the ith path: $p_t^i = (1/n), i = 1, ..., n$

Initialize the weights d_i as $1/L_i$, where Li is the length of the ith path

Choose the pheromone removal parameter α and the pheromone deposition parameter β

while $(\sum_i J_i \leq J_{max})$ **do**

Choose a path randomly in accordance with the distribution of probabilities $p_j^t, j = 1, ..., n$

if the ith path is chosen **then**

Update the trip counter of the ith path: $J_i \leftarrow J_i + 1$

Update pheromone only on path $i : C_i^{t+1} \leftarrow (1 - \alpha)C_i^t + \beta d_i p_i^t$

No changes in pheromone on paths $j \neq i : C_j^{t+1} \leftarrow C_j^t$

end if

Update path choice probabilities: $p_i^{t+1} \leftarrow C_i^{t+1} / \sum_{j=1}^n C_j^{t+1}, i = 1, ..., n$

Update total ant trip counter: $t \leftarrow t + 1$

end while

Return normalized pheromone concentrations matrix (p_i versus trip t) and final trip counter vector J

end procedure

Figure 3-1: EigenAnt algorithm

At the end of the algorithm execution, two entities are returned. The first is a pheromone concentration matrix indicating at each iteration the amount of pheromone present on each path. The second is a counter array indicating how many each path has been taken.

Jayadeva et. al have shown that given all path lengths prior to running the algorithm, the ant will over time tend to use the shortest path a lot more often, and pheromone trails on the shortest path will be much more than the rest. This will occur even if the pheromone trails are initialized to favour longer paths. Also, should the length of the paths change during the algorithm's execution, it will counteract this and shift its focus towards the shortest path. Moreover, the algorithm quite handsomely handles the case of new paths being discovered. The ant will shift focus to a new path only if it is shorter. In summary, the EigenAnt algorithm keeps its focus on the shortest path, and manages to sustain that focus regardless of false initial information or dynamically changing graph conditions.

3.1.1 Analysis of EigenAnt

In this section, we present an analysis of the equilibrium state of the EigenAnt algorithm. When a path i is chosen in iteration t, the amount of pheremone on it is updated as follows:

$$\Delta C_i^t = -\alpha C_i^t + \beta d_i p_i^t \tag{3.2}$$

As such, the expected change on path i at iteration t would be

$$\Delta C_i^t = p_i^t(-\alpha C_i^t + \beta d_i p_i^t) \tag{3.3}$$

Letting $p_i^t = C_i^t/V^t$, where V^t is the sum of the concentrations at time t, we get

$$\langle \Delta C_i^t \rangle = \frac{C_i^t}{V^t}(-\alpha C_i^t + \beta d_i \frac{C_i^t}{V^t}) \tag{3.4}$$

and hence

$$\langle \Delta \mathbf{C}^t \rangle = \frac{1}{V^t}\,\mathrm{diag}(\mathbf{C}^t)(-\alpha \mathbf{C}^t + \beta D \mathbf{C}^t) \tag{3.5}$$

$\langle \Delta \mathbf{C}^t \rangle \in \mathbb{R}$ is a vector representing the expected change in pheromone concentration in iteration t, diag($\mathbf{C}^t$) is a diagonal matrix having entries $C_1^t, ..., C_n^t$ on its principal diagonal, and D is a diagonal matrix having entries $d_1, ..., d_n$ on its principal diagonal.

In a stable state, the expected change in the total pheromone concentration is zero. In other words, $\langle \Delta C_i^t \rangle = 0$ for $i = 1, 2, 3, ..., n$. In compact form, this would be

$$\langle \Delta \mathbf{C}^t \rangle = 0. \tag{3.6}$$

A pheromone concentration vector that satisfies (3.6) is an equilibrium point. To find such points, let us first assume that

$$L_1 < L_2 < L_3 < ... < L_n \tag{3.7}$$

and consequently

$$d_1 > d_2 > d_3 > ... > d_n \tag{3.8}$$

The above equation can be written as

$$\alpha \mathbf{C} = \frac{\beta}{V} D\mathbf{C} \tag{3.9}$$

Writing $V = \mathbb{1}^T \mathbf{C}$, where $\mathbb{1} \in \mathbb{R}$ is the vector with all n components equal to 1 we get

$$\alpha \mathbf{C} = \frac{\beta}{\mathbb{1}^T \mathbf{C}} D\mathbf{C} \tag{3.10}$$

and consequently

$$D\mathbf{C} = \frac{\alpha}{\beta} (\mathbb{1}^T \mathbf{C}) \mathbf{C}. \tag{3.11}$$

Furthermore, we let $\lambda = \frac{\alpha}{\beta}(\mathbb{1}^T \mathbf{C})$ to get

$$D\mathbf{C} = \lambda \mathbf{C}, \tag{3.12}$$

which is reminiscent of an eigenvalue-eigenvector equation, with λ being the eigenvalue and $\mathbf{C}$ the eigenvector. Note though that λ is a function of $\mathbf{C}$.

Since D is a diagonal matrix, its eigenvectors are the canonical vectors $\mathbf{e}_i$, corresponding to the eigenvalues d_i. Hence, all solutions of (3.12) will be of the form $\mathbf{C} = \mu_i \mathbf{e}_i$. In other words, for some $i \in \{1, 2, 3, ..., n\}$, $C_i = \mu_i, C_j = 0, \forall j \neq i$. By substituting $\mathbb{1}^t \mu_i \mathbf{e}_i = \mu_i$ into (3.11), we get

$$D\mu_i \mathbf{e}_i = \frac{\alpha}{\beta}(\mu_i)\mu_i \mathbf{e}_i, \tag{3.13}$$

and finally

$$\mu_i = \frac{\beta}{\alpha} d_i. \tag{3.14}$$

Given that $i \in \{1, 2, 3, ..., n\}$, we have n solutions of the form $\frac{\beta}{\alpha} d_i \mathbf{e}_i$. Now that we have the equilibrium points, we proceed to show through perturbation analysis that only the point $\mu_1 \mathbf{e}_1$ representing the shortest path is stable.

By multiplying (3.1.1) by V^2, we get

$$\langle V^2 \Delta C \rangle = \operatorname{diag} C(-\alpha V C + \beta D C) \tag{3.15}$$

Starting from the equilibrium point $\mathbf{C} = \mu_i \mathbf{e}_i$, the pheromone concentration vector after perturbation would be

$$\mathbf{C} + \epsilon = \mu_i \mathbf{e}_i + \epsilon = (\epsilon_1, \epsilon_2, ..., \mu_i + \epsilon_i, \epsilon_{i+1}, ..., \epsilon_n)^T. \tag{3.16}$$

Pheromone on any path is always non-negative, so

$$\epsilon_j \geq 0, \forall j \neq i; \epsilon_i \geq \mu_i, \mathbb{1}^T \epsilon > -\mu_i. \tag{3.17}$$

As a result, we have

$$V = \mathbb{1}^T(\mathbf{C} + \epsilon) = \frac{\beta}{\alpha} d_i + \mathbb{1}^T \epsilon. \tag{3.18}$$

By substituting (3.16) and (3.18) in (3.15) and dropping higher-order terms ($O(\epsilon^3)$),

the value of the component of $\langle V^2 \Delta \mathbf{C} \rangle^T$ in the direction of $\mathbf{e}_j$ can be calculated as

$$\langle V^2 \Delta \mathbf{C} \rangle^T \mathbf{e}_j = \begin{cases} \epsilon_j^2 \beta (d_i - d_i,) & j \neq i \\ -(\mu_i + \epsilon_i)^2 (\mathbb{1}^T \epsilon), & j = i. \end{cases} \tag{3.19}$$

Consider the equilibria $\mu_i e_i, i \neq 1$. For all $j < i$, $(d_j - d_i) > 0$ and hence $\langle V^2 \Delta \mathbf{C} \rangle^T \mathbf{e}_j > 0$, leading to a movement away from equilibrium. The converse is true for cases where $j > i$. Overall however, this means that these equilibria are unstable.

As for $\mu_1 e_1$, we see that $(d_j - d_i) < 0$ for all j, and as such all perturbations in the directions $\mathbf{e}_j$ are negative. For perturbations in the direction $\mathbf{e}_i$, we find that $\langle V^2 \Delta \mathbf{C} \rangle^T \mathbf{e}_j > 0$ if $\mathbb{1}^T \epsilon < 0$ and vice-versa, indicating that the equilibrium $\mu_1 e_1$ is stable. Since $\mathbb{1}^T \epsilon$ represents the perturbation in the total pheromone concentration, this also indicates that the system as a whole acts to preserve the total amount of pheromone. Given that L_1 is the shortest length, we conclude that the pheromone trails remain concentrated on the shortest path, and that the total pheromone remains conserved. Since path choice is probabilistic, adding a new shortest path will cause the system to converge to it over time as well [1].

Note here how $\mu_1 e_1 = (\frac{\beta}{\alpha} d_1)\mathbf{e}_1$, an eigenvector, is associated with the the dominant eigenvalue, which is the largest element of $\frac{\beta}{\alpha} D$. That, and it being the only asymptotically stable equilibrium point of (3.12), inspired the authors with the name EigenAnt.

3.2 Hybrid Ant System and the SOP–3–exchange local optimization procedure

The hybrid ant system (HAS–SOP) [20] is an ACS system that uses the SOP–3–exchange as its local search procedure. SOP–3–exchange is an edge–exchange procedure. In such a procedure, a given tour is iteratively improved by continually replacing a set of k edges with another, provided that the swap improves the tour quality, represented in SOP's case by its length. This operation is called a *k–exchange*. The

procedure repeats this operation till no improving *k–exchange* can be found. At this point the final solution is said to be *k–optimal*. Verifying that a solution is *k–optimal* requires $O(n^k)$ time, assuming no more than a constant amount of time is exerted to check the feasibility and improvement of an exchange.

Increasing the value of k leads to better quality solutions, but due to computational constraints k is generally restricted to a maximum value of 3. 2–opt, 3–opt and the Lin–Kernighan–heuristic are examples of edge–exchange procedures [24, 25]. 2–opt uses 2-*exchanges*, 3–opt uses 3–*exchanges*, and the Lin–Kernighan method allows the cardinality of the swapped edge sets to vary.

When a 2–*exchange* is performed on a tour, part of the tour must be inverted. In TSP, this is not a problem because all edges have the same cost in both directions. This is not the case in ATSP, and as such whenever we perform a 2–*exchange* we will not be able to calculate the effect of the exchange in constant time, as we would need to perform a linear cost calculation of the cost of the inverted part of the tour. By using 3–*exchanges* instead, we will always be able to find a new set of edges to replace the old ones that preserves the directions of the tour segments affected in the swap, allowing us to calculate the improvement in constant time.

As an example, suppose we have a tour $\langle x_0, x_1, ..., x_n \rangle$. If we remove the edges (x_k, x_{k+1}) and (x_l, x_{l+1}), our only option would be to reverse the tour section from x_{k+1} to x_l, and then add the edges (x_k, x_l) and (x_{k+1}, x_{l+1}), as illustrated in Figure (3-2). We would now need to calculate the cost of the tour section after reversal, as well as to check that no precedence constraints were violated in the case of SOP. If we instead opt to remove an extra third edge (x_m, x_{m+1}), we will have two tour sections $\langle x_{k+1}, ..., x_l \rangle$ and $\langle x_{l+1}, ..., x_m \rangle$. By adding the edges (x_k, x_{l+1}), (x_m, x_{k+1}) and (x_l, x_{m+1}), we will effectively swap their positions without altering the order of their elements, as illustrated in Figure (3-3). This is called a *path–preserving–*3–*exchange* (*pp–*3–*exchange*). An $O(1)$ check of the edges' costs is all that is needed to calculate the improvement gained by the exchange.

After performing a 3–*exchange* on a tour, we would still need to check that no precedence constraints have been violated. This requires a computational effort of

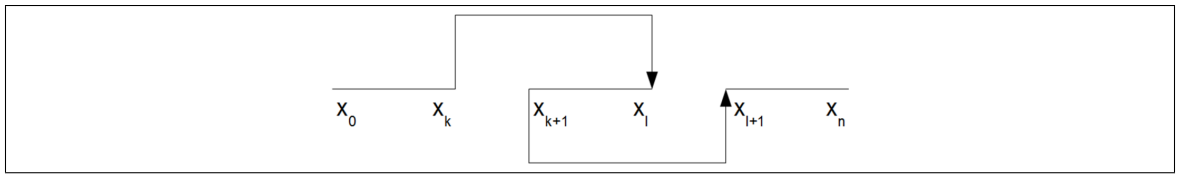

Figure 3-2: Illustration of the 2–*exchange* operation

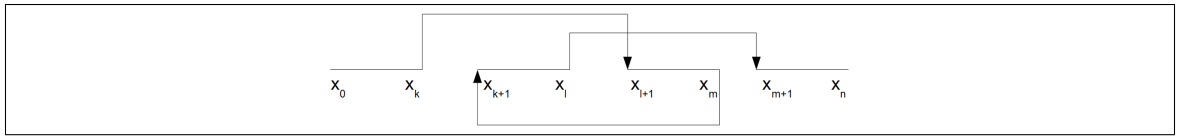

Figure 3-3: Illustration of the *path–preserving*–3–*exchange* operation

$O(n^2)$, as for each node we would need to check that none of its successor nodes have already been visited. To overcome this, a lexicographic search strategy [23] is employed, allowing the exploration of only feasible solutions.

Suppose we have a tour and three indices h, i and j, where $h < i < j$. The three indices indicate the three edges to be swapped out in an exchange, $(h, h+1), (i, i+1)$ and $(j, j+1)$. Removing the three edges gives rise to two isolated sections of the tour, path–left $= \langle h+1, ..., i \rangle$ and path–right $= \langle i+1, ..., j \rangle$. The exchange is feasible if all elements in path–right can appear in the tour before all elements in path–left. Assuming it is feasible, we can create a new exchange by adding an extra element to path–right. Since the original exchange was already feasible, we only need to check the new element against the current members of path–left to verify the feasibility of the new exchange. If we hit an element that causes a violation, i is incremented and j is set to $i+1$, expanding path–left by one element and resetting path–right to $\langle j \rangle$, and still only needing to check the now single element in path–right against those in path–left. When all possible combinations of path–left and path–right where path–left starts with h have been explored, we repeat the process with a new value for h. At this point, we have minimized the feasibility verification effort to $O(n)$.

In a *forward–lexicographic–path–preserving*–3–*exchange* (*f–lpp*–3–*exchange*) procedure, we start with the indices h, i, and j set to 0, 1 and 2 respectively. As such, the two paths are set to $\langle 1 \rangle$ and $\langle 2 \rangle$. If this exchange is feasible and beneficial, we can perform it immediately or postpone it to be compared against other potential exchanges, depending on parameters passed to the procedure. After that, path–right

is expanded by one element, which is checked against the elements of path–right for feasibility. When a violation is found, path–left is expanded by one element, and path–right is reset to include the element immediately after. In essence, we loop over all possible values of h, i and j, where $h < i < j$ and swapping the two paths $\langle h+1, ..., i\rangle$ and $\langle i+1, ..., j\rangle$ is feasible. More importantly we explore these solutions with only a linear cost required for feasibility checking as noted above.

A *backward–lpp–*3*–exchange* is the mirror image of the forward version where $j < i < h$, path–left $= \langle j, ..., i\rangle$ and path–right $= \langle i+1, ..., h\rangle$. In this case, path–left is iteratively expanded by decrementing j, and when it can be expanded no further due to a constraint violation, path–right is expanded by one element and path–left is reset. In the SOP–3–exchange procedure, both directions are performed for each node h.

To reduce the cost of the feasibility check down to a constant level, a labelling mechanism is utilized. For each node v in the tour, we keep track of the value mark(v). Whenever an element is added to path–left, we set mark(v) where $v \in$ successors(i) to the value of a global variable *count_h*, which is initially set to zero. By doing so, when a new node is considered for path–right, we immediately know if it is feasible to add by checking whether the mark value is equal to *count_h*, hence performing the check in constant time. When a new value of h is used, the value of *count_h* is incremented, immediately invalidating the mark values of all nodes, and relieving us from us from the effort of resetting the marks.

At this point, the cost of finding one profitable exchange using the lexicographic search and the labelling techniques is $O(n^3)$. To allow the local search to repeatedly improve the tour, rather than just return a one–time improvement, this would need to be reduced as new solutions are found. Two ways of achieving this are the heuristic selection of nodes to explore and the premature halting of a search when certain conditions have been met.

By default, SOP–3–exchange explores all possible values of i and j for a selected value of h. An alternative to this is the OR–*exchange* [30]. It dictates that given an h, we should only explore the three closest values of h. This means the value of i will

be restricted to $h+1, h+2, h+3$ for the forward direction and $h-1, h-2, h-3$ for the backward direction. A parameter in the SOP–3–exchange procedure called WalkingCriterion dictates whether the OR–*exchange* or 3–*exchange* walking criterion is used.

The *don't–look bit* [10] and the *don't–push stack* [20] are two data structures that can be used to minimize the number of nodes explored (i.e. the possible values of h). The *don't–look bit* data structure associates each node with a bit indicating if it should not be explored. Initially all bits are set to zero. When a node is explored, its bit is set to one. If a profitable exchange is found, the bits of the six pivot nodes ($h, h+1, i, i+1, j$, and $j+1$) are reset to zero after performing the exchange. The search then attempts to use the first available h that has an unmarked bit. This is repeated until all bits are set to one.

The *don't–push stack* is a stack with a special push operation. In the beginning of the search it contains all nodes. The topmost element in the stack represents the next value of h to explore. When a profitable exchange is found the six pivot nodes that are not already present in the stack are pushed onto it in reverse order, so that h is retained as the topmost node. The exchange is performed after the push operation. We then proceed to attempt to improve the newly found solution given the remaining values of h in the stack. This allows the search to remain in the neighbourhood of the last improvement, rather than re–attempting all possible values of h again. The structure also gives us the added characteristic of a non–sequential search, which is aided by the fact that the labelling procedure works independently of the value of h. The search continues until the stack becomes empty.

During the exploration of a node h, we essentially have three loops, one for each of h, i and j. Stopping a loop after finding an improvement rather than waiting for all three to finish helps decrease the exploration effort. The SOP–3 procedure has a parameter ExchangeFirstCriterion which dictates at which of these loops a search should stop if an improvement was found. For example, setting ExchangeFirstCriterion to j will stop the search as soon as an improvement is found, setting it to i will test all values of j before checking if we have found a solution, and setting it to h will

force the search to try all values of i and j for a given h. Not stopping at any of the loops, and as such essentially exploring all possible 3–*exchanges* applicable to a tour, has been found to consume too much computation time.

In our experimental results in this paper, we set ExchangeFirstCriterion = i, and WalkingCriterion = 3–*exchange*. A pseudo–code of our implementation is presented in Figure (3-4).

```
procedure SOP3Exchange(A valid sequence (0...n), G)
    count_h ← 0
    solutionfound ← true
    while solutionfound do
        solutionfound ← false
        h ← PopStack()
        i ← h + 1
        count_h ← count_h + 1
        while i < n and not solutionfound do
            j ← i+1
            feasible ← true
            bestGain ← 0
            for k ∈ succcessors[i] do
                mark[k] ← count_h
            end for
            while j < n and feasible do
                feasible ← mark[j] ≠ count_h
                gain ← ComputeGain(h, i, j, G)
                if feasible and gain > bestgain then
                    solutionfound ← true
                    bh ← h
                    bi ← i
                    bj ← j
                    bestgain ← gain
                end if
                j ← j + 1
            end while
            i ← i + 1
        end while
```

Figure 3-4: Local search pseudo-code

```
        if solutionfound then
            PerformExchange(bh, bi, bj, G)
            PushToStack(bh, bi, bj)
            Restart–While–Loop
        end if
        i ← h–1
        count_h ← count_h + 1
        while i > 0 and not solutionfound do
            j ← i – 1
            for k ∈ predecessors[i] do
                mark[k] ← count_h
            end for
            while j > 0 and feasible do
                feasible ← mark[j] ≠ count_h
                gain ← ComputeGain(h, i, j, G)
                bestGain ← 0
                if feasible and gain > bestgain then
                    solutionfound ← true
                    bh ← h
                    bi ← i
                    bj ← j
                    bestgain ← gain
                end if
                j ← j – 1
            end while
            i ← i – 1
        end while
        if solutionfound then
            PerformExchange(bj, bi, bh, G)
            PushToStack(bh, bi, bj)
        end if
    end while
end procedure
```

Figure 3-4: Local search pseudo-code

3.3 Enhanced Ant Colony System

Gambardella *et al.*'s Enhanced Ant Colony System [21] differs from conventional ACS in two ways. In the **ConstructAntSolutions** step, when an ant k is choosing a node $j \in D(S_k)$ to add to its current partial tour S_k, it performs the following. Let $i = \lambda(S_k)$ denote the last node visited in the partial tour S_k, let S^* denote the global best–so–far tour, and let j be the node that follows i in S^*.

1. With probability q_0:

   (a) If the node j has not yet been visited in S_k and the edge (i, j) does not violate the precedence constraints, then the partial tour S_k is extended with node j. Call this **Case 1**.

   (b) Otherwise, the deterministic selection approach described in Section II is applied according to Equation (3). Call this **Case 2**.

2. With probability $(1 - q_0)$, the stochastic selection approach described in Section II is applied according to Equation (2). Call this **Case 3**.

In other words, with probability q_0, EACS first attempts to take the next edge in the best–so–far tour S^* if that edge is feasible. This makes EACS much more exploitative than ACS because the best–so–far tour has a much greater influence on tour construction than in ACS. Also, due to the usually high value of q_0, it greatly increases the speed of solution construction. The rest of the behaviour of EACS is similar to ACS. If the next edge in S^* is not feasible, then EACS applies the deterministic approach as in conventional ACS. Finally, with probability $(1 - q_0)$, EACS applies the stochastic selection approach, as in conventional ACS and as in most other ACO models [17]. The other difference between EACS and ACS, in the context of the SOP problem, is in the **LocalSearch** step:

1. EACS applies local search to a constructed tour only if its cost is within 20% of the cost of S^*.

2. The *don't–push stack* (see Section 2.2) is initialized to contain only the nodes that are out of sequence with the best solution.

3. For reference purposes, we shall call this Enhanced SOP–3 (ESOP–3) local search.

Chapter 4

Proposed algorithms

4.1 Probabilistic EACS

Our first approach is a modified version of EACS that aims to be less exploitative. For future reference we shall call it probabilistic EACS (PEACS) [3]. In EACS, an ant would select the next to visit in one of the following manners:

1. With probability q_0:
   (a) The ant tries to select the node j which follows i in S^*, if that node j has not been visited yet and does not violate precedence constraints. (**Case 1**)
   (b) If that node j is not feasible, then most attractive node is chosen. (**Case 2**)
2. With probability $(1-q_0)$, the stochastic choice is applied according to Equation (2). (**Case 3**)

In our variation, we alter Case 2 by letting the ant select j according to the stochastic choice as in Case 3. This is different from decreasing the value of q_0, which while at first glance might achieve the same effect of giving ants a larger freedom of choice, would cause the global–best–suggestion to be ignored even when it is available more frequently, potentially causing a decrease in the quality of generated solutions.

Our variation allows the global–best suggestions to be used as much as possible while giving ants a freedom of choice only when those suggestions are infeasible.

There were some assumptions that we had to take in our implementation. The don't–push stack is initialized such that the topmost node is the rightmost node in the group of nodes selected to be in it. Additionally, when an exchange is performed we push the participating nodes such that the leftmost node is at the top and the rightmost is at the bottom. This means that in the forward case, h will be at the top of the stack after the exchange, while in the backward case it will be j. Furthermore, we assumed that "initializing the don't–push–stack with out–of–sequence elements" means the following: if element i in the current tour being improved and the best–so–far tour are different, then i would be considered out of sequence and added to the don't–push–stack. Finally, in our implementation, when an ant is deciding which node to choose, all feasible nodes are considered viable regardless of their distance from the current node, rather than using a candidate list.

4.2 EigenAnt-Based Ant System

Our second approach is based on the EigenAnt algorithm, the EigenAnt Ant System (EAAS) [2]. It is characterized by the following:

1. Pheromone levels are initially set to the distance of the corresponding edges.

2. During solution construction, ants choose the next node to visit using the approach used in EACS , with one important difference. Only the pheromone trail information is taken into consideration. No heuristic information is stored or calculated for edges. This in particular is quite a departure from most, if not all, ACO algorithms.

3. Pheromone trails are modified after solution construction.

4. After solution construction, solutions that are within 20% of the best solution found so far pass though the local optimization procedure (note that in our

experiments, we only used one ant, but it is possible to use more). The best solution in the current iteration is used to update the pheromone trails

In EigenAnt, a path i selected by the ant has its pheromone trail updated in the following manner:

$$\tau_i(t+1) = (1-\alpha)\tau_i(t) + \beta d_i p_i^t \tag{4.1}$$

In our variation, we update the pheromone trail on an edge (i, j) on the selected path as follows:

$$\tau_{ij}(t+1) = (1-\alpha)\tau_{ij}(t) + \beta d_i p_{ij}^t \tag{4.2}$$

The main factor here is p_{ij}^t, the probability to visit node j when the ant is standing on node i. As aforementioned, when paths are considered, it is calculated as the ratio of the pheromone on the path to the sum of all trails on all paths. We decided to use the same calculation for edges, with one change. We take into the consideration the fact that during solution construction, edges become infeasible, either due to nodes being visited or to precedence constraints not yet being satisfied. As a result, edge number k in a tour will be compared against $n-k$ other edges to calculate its of probability of selection.

Chapter 5

Experimental Results

5.1 Experimental Setup

5.1.1 Problem Instances

We evaluated our systems relative to Gambardella *et al.*'s EACS, and relative to standard ACS, using the SOP instances at the SOPLIB2006 library [22]. This is the same set of instances used in the experimental results of Gambardella *et al.* [17].

The SOPLIB naming convention is as follows: the name of each instance takes the format R.n.c.r where n is the number of nodes, c is the maximum cost of a valid edge, and r is the probability that an edge will be an invalid edge, representing a precedence constraint. The SOPLIB consists of 48 instances based on the following set of values of these parameters: $n \in \{200, 300, 400, 500, 600, 700\}$, $c \in \{100, 1000\}$ and $r \in \{1, 15, 30, 60\}$.

5.1.2 Algorithm Parameters

Based on [20] and [21], we implemented four systems on top of Stützle's public–domain ACOTSP implementation [34]:

1. A standard ACS system, with only one modification: local search is only applied to a solution if its cost is within 20% of the best–so–far solution

2. Our implementation of Gambardella *et al.*'s EACS, as Gambardella *et al.*'s own implementation is not publicly available

3. Our implementation of our modified less-exploitative EACS, PEACS

4. Our implementation based on the EigenAnt algorithm, EAAS

All of the implementations are publicly available at `https://eacssop.codeplex.com/`.

We followed the parameter settings used by Gambardella *et al.* [21]: $\rho = \eta = 0.1$, m (Number of ants) = 10, ExchangeFirstCriterion = i, WalkingCriterion = 3–*exchange*, and $q_0 = 1 - (s/|V|)$, where $s = 5$ and $|V|$ is the number of nodes (s was previously set to 10 in [20]). For ACS, we set $\alpha = \beta = 1$; for EACS and our modified EACS, we set $\alpha = \beta = 0.5$. The parameter τ_0 is set to $1/(n \cdot c(S^{init}))$, where $c(S^{init})$ is the cost of the tour found by the algorithm using only heuristic information.

For EAAS, we set m = 1, $\alpha = 0.5$, and $\beta = 1$. We set the pheromone trail on an edge (i, j) to be equal to zero if it is infeasible (i.e.: it is a precedence constraint), a small constant if the edge length is zero, and to its length otherwise. The parameters for SOP-3-exchange and q_0 are the same as in PEACS. All experiments were carried out on a computer with an Intel Core 2 Duo P7450 2.13 GHz CPU and 4 GB's of RAM.

5.2 Empirical Data

5.2.1 Results

We ran each configuration 30 times for 10 minutes of CPU time. We recorded the mean and standard deviation of the solution cost for each configuration, for each problem instance. These results are shown in Table (5.1). We note that PEACS tied with standard EACS in 7 out of the 48 instances (3 of these 7 ties were "easy" instances for which EACS also tied ACS), and had a lower average solution cost than EACS in 33 out of the remaining 41 problem instances.

With EAAS, the results were quite divisive. For problems with precedence probabilities 1 and 60 (i.e.: Either almost no or a lot of precedence constraints), EAAS performed poorly. For the other two categories, it was better across the board, achieving a better average on all problem instances.

Table 5.1: Experimental Results

Instance	ACS		EACS		PEACS		EAAS	
	mean	stdev	mean	stdev	mean	stdev	mean	stdev
R.200.100.1	76	3.3	73	3	74	2.5	108.67	7.02
R.200.100.15	1957	29.1	1890	33.4	1836	25.7	1899.57	37.74
R.200.100.30	4236	13.3	4229	3.2	4223	6.6	4225.97	5.59
R.200.100.60	71749	0	71749	0	71749	0	72332.1	502.68
R.200.1000.1	1529	34.7	1459	19.2	1460	20.1	1705.97	57.69
R.200.1000.15	22255	397.2	21503	356.9	20865	179.2	21024.87	282.5
R.200.1000.30	41456	87.4	41223	39.3	41196	0	41210.37	29.97
R.200.1000.60	71556	0	71556	0	71556	0	71718.3	210.04
R.300.100.1	54	3.3	44	3.4	44	4.8	104.53	9
R.300.100.15	3589	127.5	3317	37.6	3231	30.3	3240.9	27.74
R.300.100.30	6192	33.9	6132	10.8	6120	0	6126.1	10.95
R.300.100.60	9726	0	9726	0	9726	0	9754.37	23.04
R.300.1000.1	1505	41.1	1450	27.4	1446	30.5	1847.93	73.21
R.300.1000.15	33353	835.2	31108	731.3	30062	343.6	30131.7	539.7
R.300.1000.30	54734	400.6	54397	113.2	54183	32.3	54270.43	282.07
R.300.1000.60	109633	0	109590	0	109549	58.5	110247.7	547.66
R.400.100.1	48	6.3	34	4	34	3.8	102.63	12.39
R.400.100.15	4622	103.7	4324	76.9	4031	36	4046.57	39.16
R.400.100.30	8420	60.4	8230	36.7	8184	23.1	8189.87	14.39
R.400.100.60	15235	14	15228	0	15228	0	15337.23	64.55
R.400.1000.1	1695	146.8	1533	34.6	1518	36.4	2016.1	66.05
R.400.1000.15	44962	1010.4	42671	629.5	40228	413.8	40413.87	519.5
R.400.1000.30	86149	341.5	85739	262.9	85324	119.1	85561.07	231.17
R.400.1000.60	140852	65.5	140932	117.5	140878	103.9	141367.3	405.14

R.500.100.1	45	7.3	23	4.7	25	4	100.27	15.57
R.500.100.15	6485	208.9	5892	97.5	5509	48.9	5527.8	55.65
R.500.100.30	10161	90.2	9765	27.4	9674	10.2	9702.07	19.19
R.500.100.60	18295	6.7	18267	1.9	18264	3.4	18367.7	76.04
R.500.1000.1	1669	41.9	1547	29.4	1560	35.8	2117.37	90.44
R.500.1000.15	59860	2140.7	54705	835.5	51541	373.5	51724.97	386.71
R.500.1000.30	102288	1154.2	99692	254.6	99142	90.2	99259.33	170.28
R.500.1000.60	178424	240.3	178212	0	178212	0	179226.8	683.7
R.600.100.1	47	7.1	16	4.7	18	3.8	101.87	13.14
R.600.100.15	7144	141.7	6442	101.2	5840	63.7	5848.53	61.73
R.600.100.30	13005	127.7	12569	34.3	12474	10.4	12517.8	23.9
R.600.100.60	23381	38.7	23326	0	23317	14.8	23460.6	69.48
R.600.1000.1	1761	41.4	1647	32.4	1611	31.7	2309.33	105.65
R.600.1000.15	67048	1920.6	62188	895.6	57881	488.1	58559.2	649.06
R.600.1000.30	133417	2967.7	128613	356.3	127154	261.8	127266.6	318.42
R.600.1000.60	216018	451.4	214701	80.8	214748	108.3	215980.3	603.91
R.700.100.1	43	8.7	12	3.1	15	3.8	87.3	14
R.700.100.15	8693	185.6	8021	111.5	7411	48.2	7494.3	68.23
R.700.100.30	15352	183.2	14613	31.5	14530	8.1	14583.03	21.73
R.700.100.60	24229	16.3	24124	22.4	24144	21.2	24301.1	94.64
R.700.1000.1	1744	55.2	1614	46.4	1587	41.7	2366.6	128.56
R.700.1000.15	82659	3264.5	74680	1240.8	68490	622.3	69421.73	751.84
R.700.1000.30	141764	1586.5	135869	442.1	134711	137.5	134862.7	308.84
R.700.1000.60	248852	649	245781	131.8	245655	119.4	247587	759.9

The following charts illustrate the relative performance of the configurations. For each instance, the average solution quality over 30 runs is plotted against the corresponding configuration. Each chart represents a subset of the problems with a particular cost and precedence probability.

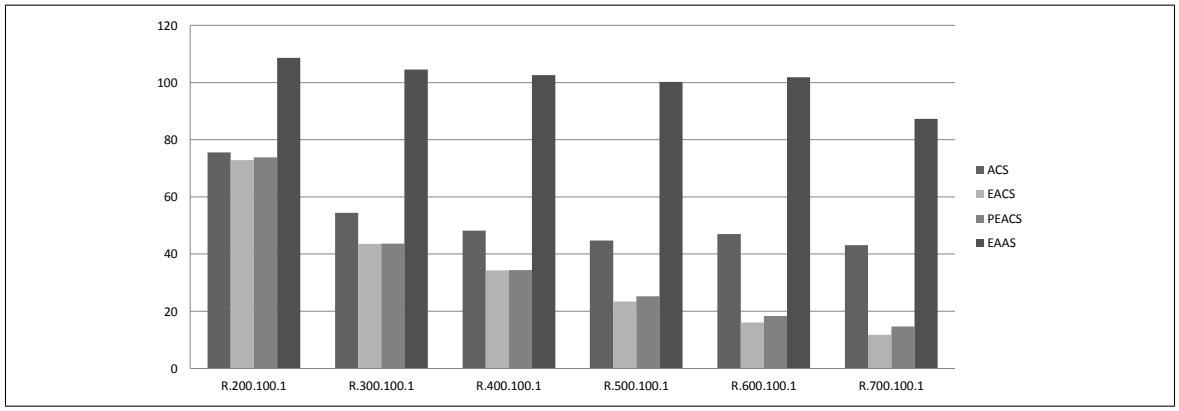

Figure 5-1: Performance Comparison - Cost 100 - Precedence 1

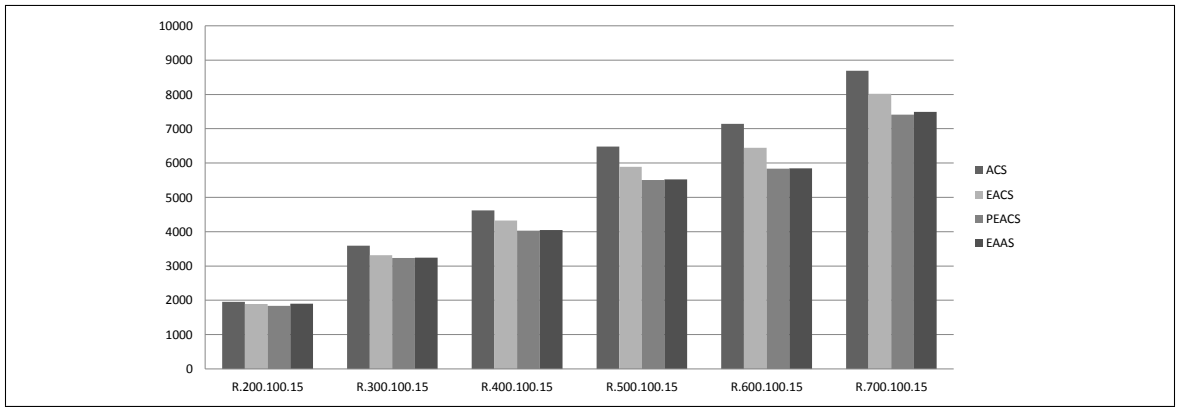

Figure 5-2: Performance Comparison - Cost 100 - Precedence 15

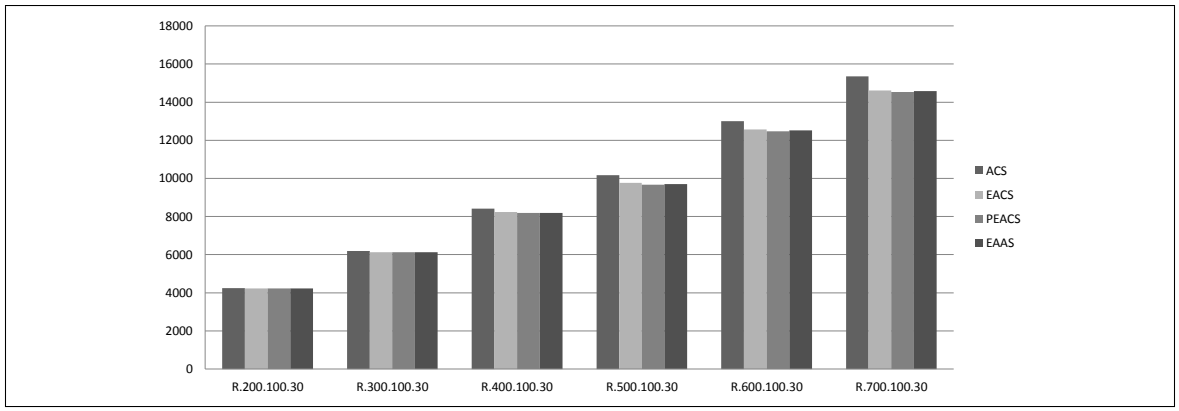

Figure 5-3: Performance Comparison - Cost 100 - Precedence 30

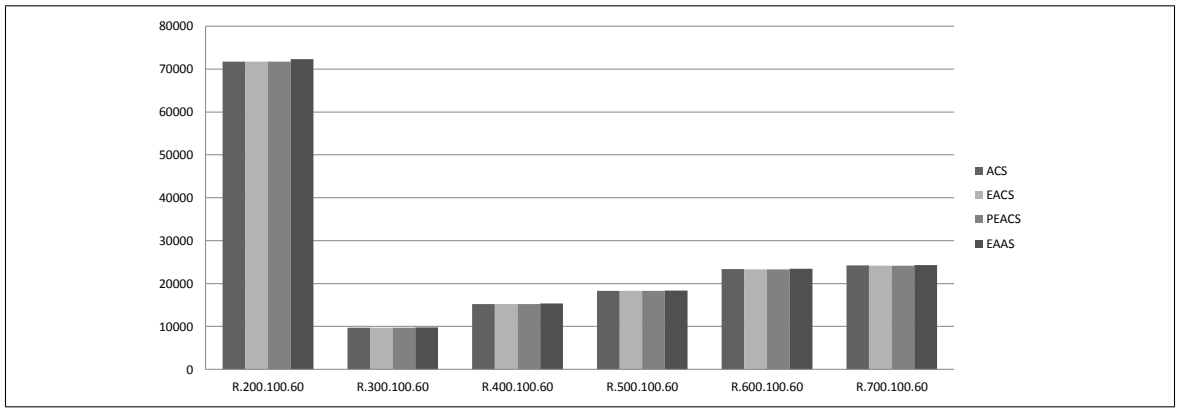

Figure 5-4: Performance Comparison - Cost 100 - Precedence 60

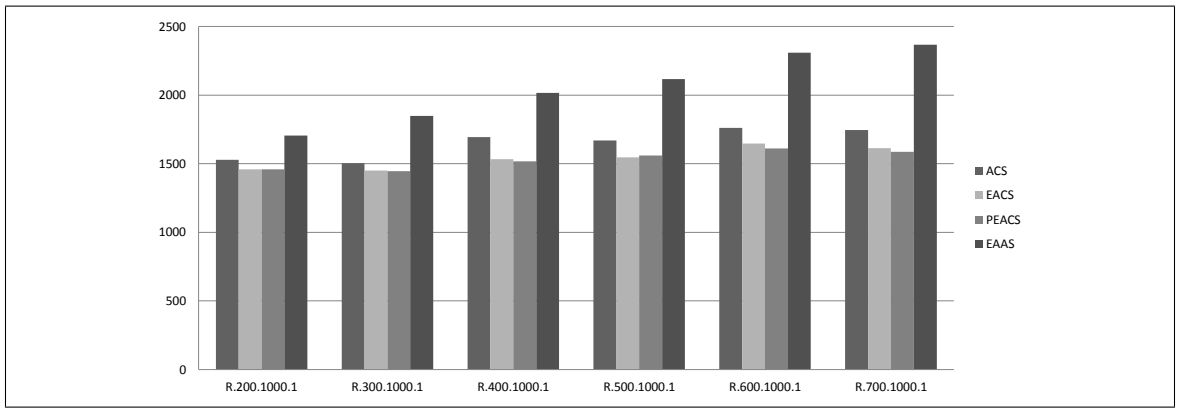

Figure 5-5: Performance Comparison - Cost 1000 - Precedence 1

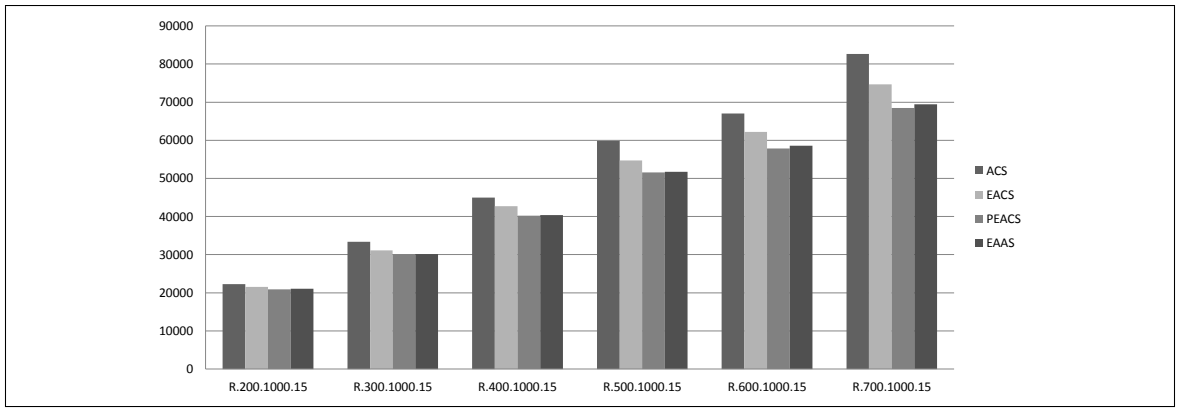

Figure 5-6: Performance Comparison - Cost 1000 - Precedence 15

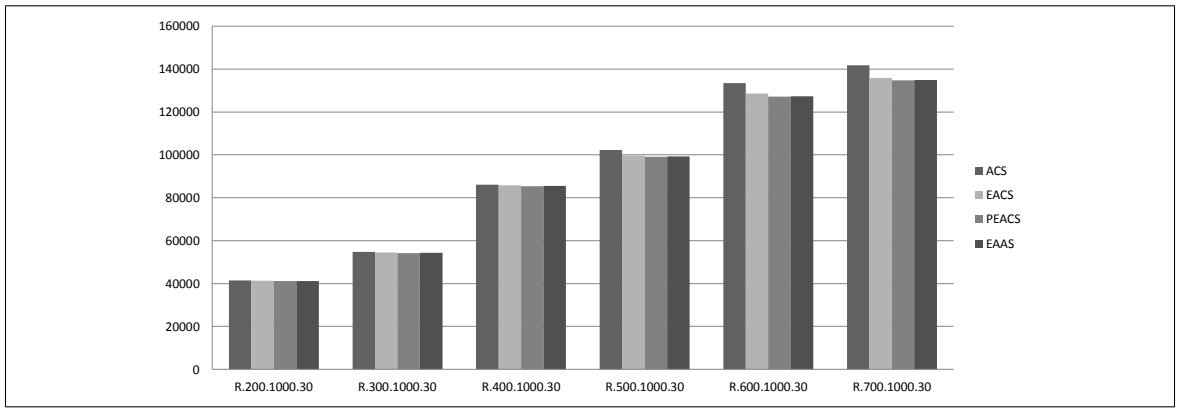

Figure 5-7: Performance Comparison - Cost 1000 - Precedence 30

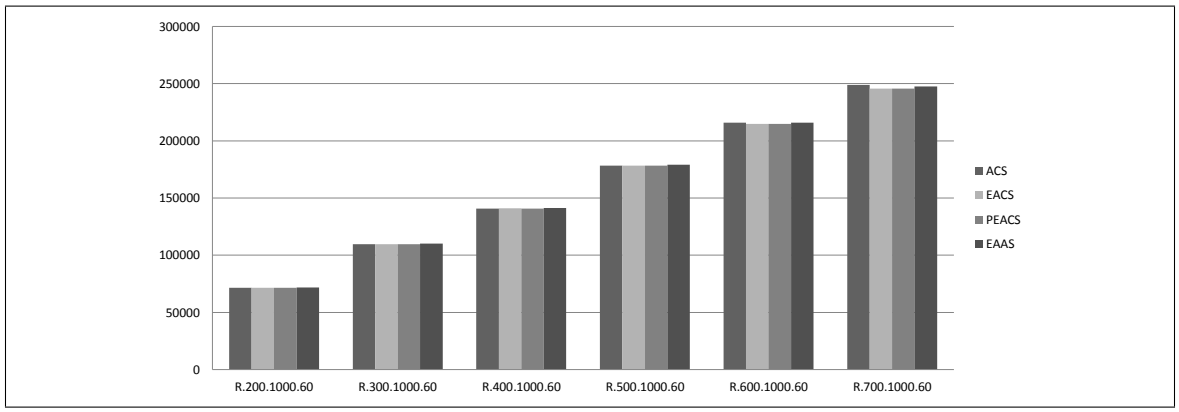

Figure 5-8: Performance Comparison - Cost 1000 - Precedence 60

The following charts illustrate the performance of the configurations over time for a selection of the instances. Again, each chart represents a subset of the problems with a particular cost and precedence probability. The average percentage difference against the best average solution quality is plotted for each algorithm at 1 minute intervals.

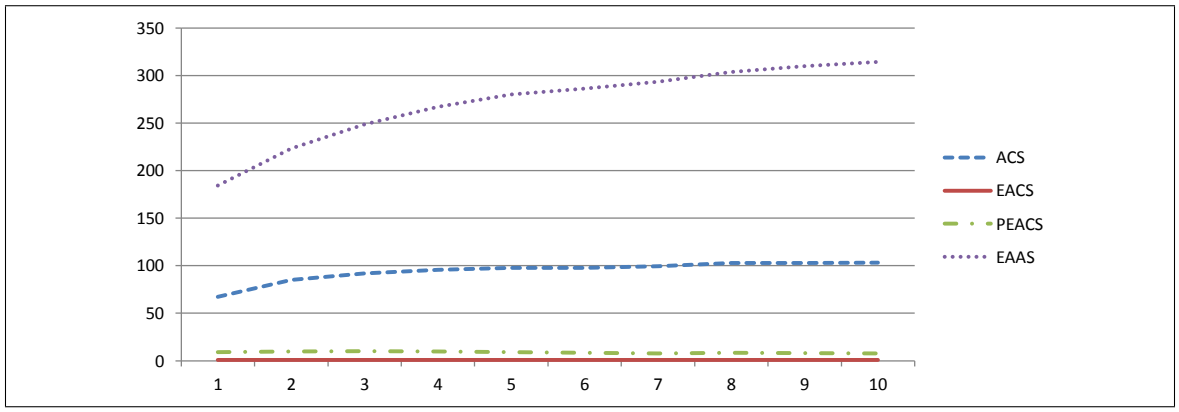

Figure 5-9: Performance over time - Cost 100 - Precedence 1

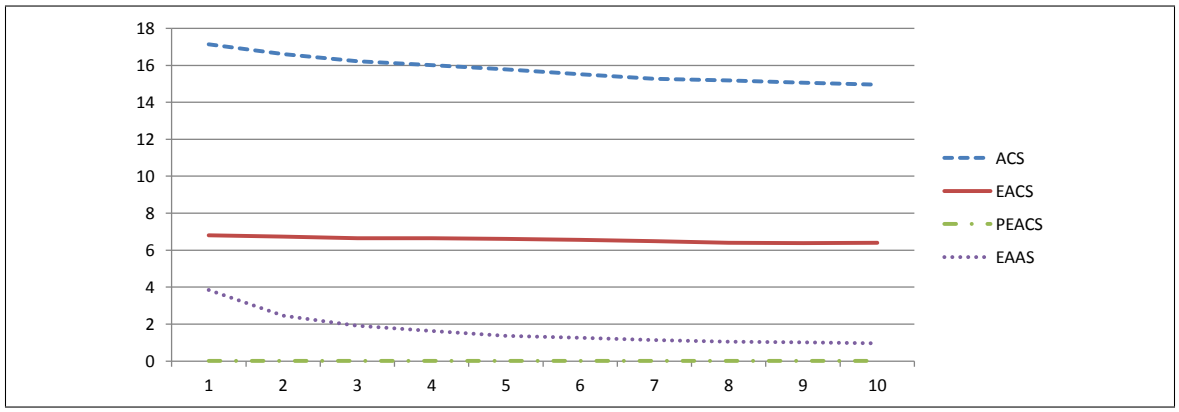

Figure 5-10: Performance over time - Cost 100 - Precedence 15

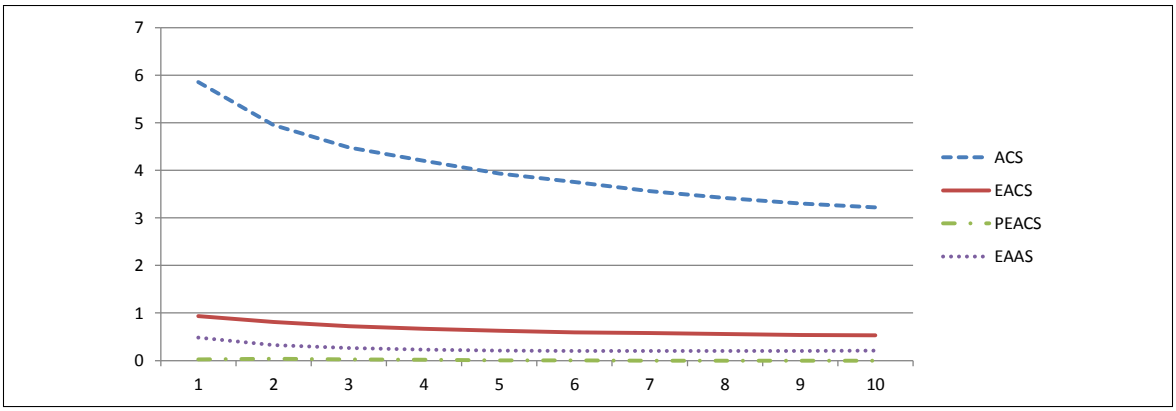

Figure 5-11: Performance over time - Cost 100 - Precedence 30

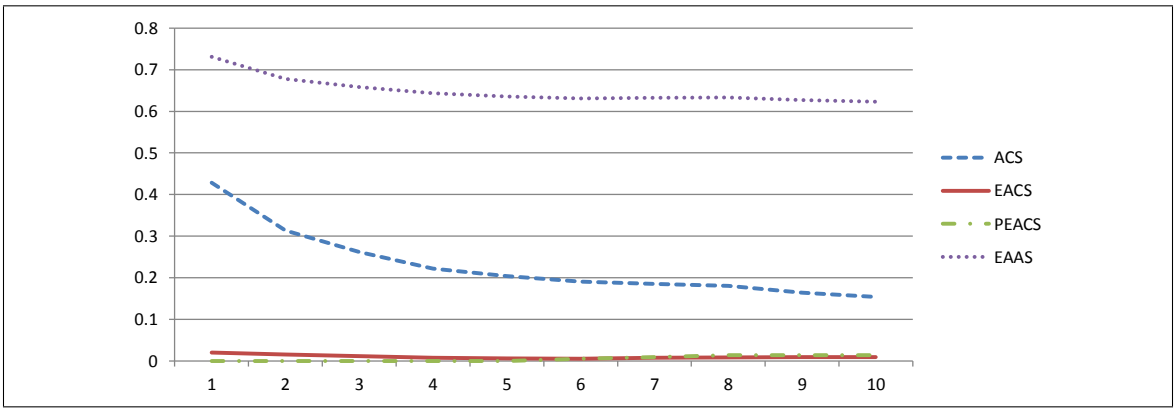

Figure 5-12: Performance over time - Cost 100 - Precedence 60

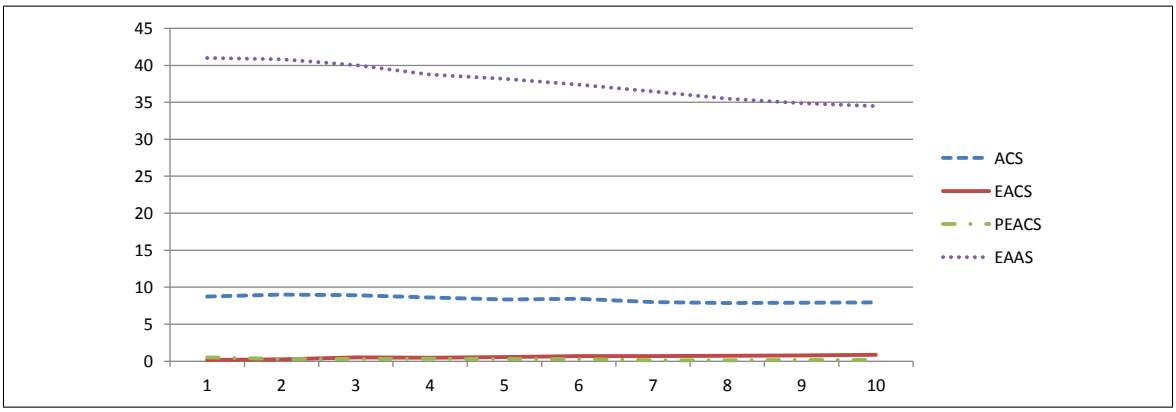

Figure 5-13: Performance over time - Cost 1000 - Precedence 1

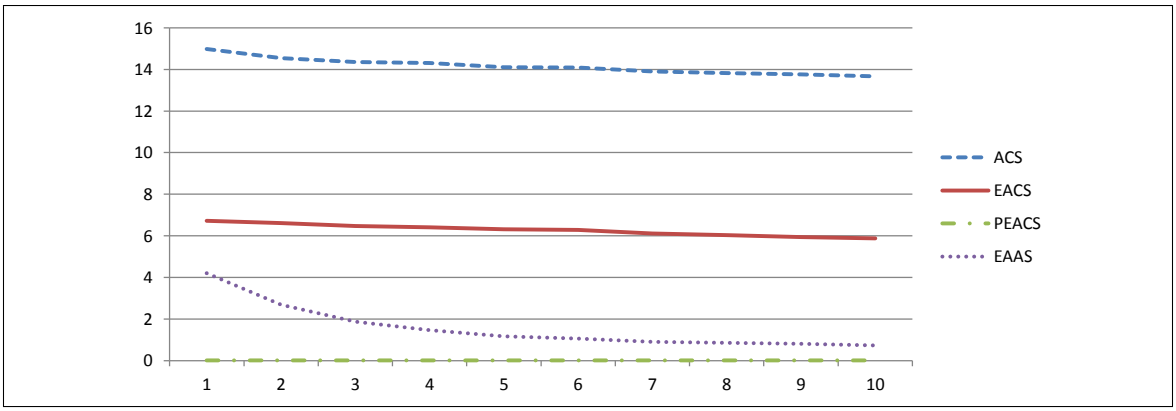

Figure 5-14: Performance over time - Cost 1000 - Precedence 15

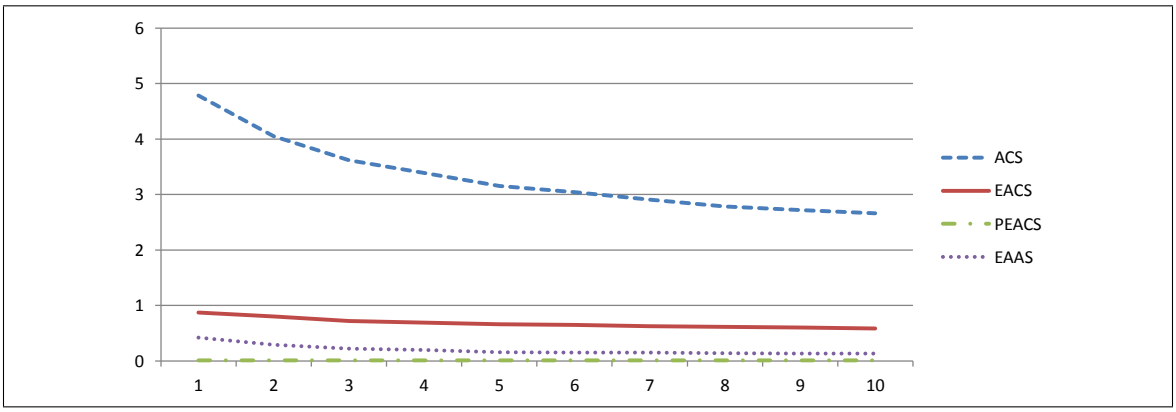

Figure 5-15: Performance over time - Cost 1000 - Precedence 30

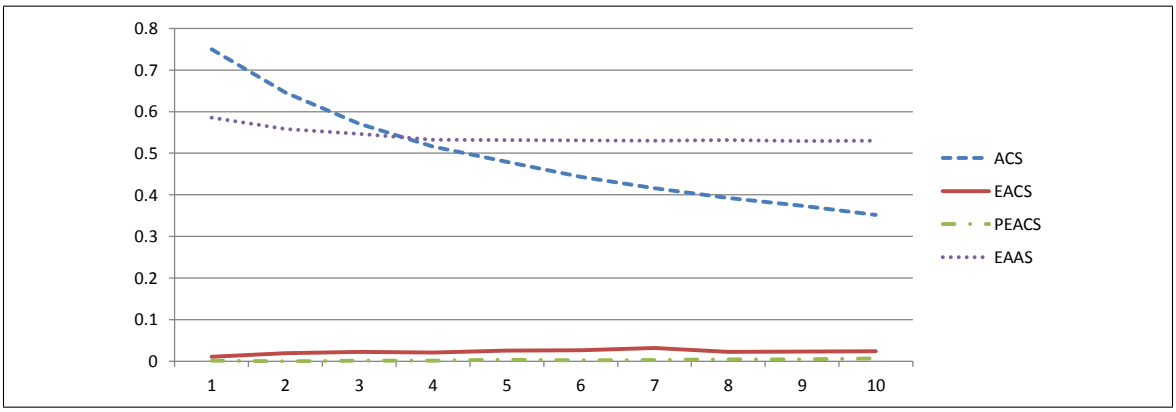

Figure 5-16: Performance over time - Cost 1000 - Precedence 60

We see that problems with a precedence probability of 15 and 30 are where our new configurations provide the best improvement. PEACS and EAAS consistently outperform EACS (and consequently ACS) throughout their operation. While the margin is considerably smaller, PEACS consistently outperforms EAAS as well.

Another measure we conducted is to calculate the percentage improvement of PEACS and EAAS against EACS per instance at one minute intervals. The results are shown in table (5.2) and (5.3). They confirm the results obtained from the temporal analysis.

Table 5.2: Percentage improvement of PEACS over EACS

Instance	1 min.	2 min.	3 min.	4 min.	5 min.	6 min.	7 min.	8 min.	9 min.	10 min.
R.200.100.1	-0.43	-0.83	-1.36	-2.16	-2.09	-1.74	-1.75	-1.26	-1.22	-1.26
R.200.100.15	4.07	3.55	3.25	3.19	3.14	3.02	2.89	2.89	2.96	2.93
R.200.100.30	0.09	0.04	0.08	0.09	0.09	0.10	0.14	0.13	0.14	0.15
R.200.100.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
R.200.1000.1	-0.70	-0.24	-0.27	-0.23	0.10	0.20	0.08	0.06	0.07	-0.06
R.200.1000.15	4.01	3.66	3.49	3.41	3.40	3.39	3.24	3.17	3.09	3.06
R.200.1000.30	0.19	0.16	0.13	0.10	0.08	0.08	0.07	0.07	0.07	0.07
R.200.1000.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
R.300.100.1	-4.75	-5.41	-4.62	-3.79	-2.48	-2.74	-2.24	-2.17	-1.21	-0.15
R.300.100.15	3.65	3.12	2.99	3.03	2.89	2.77	2.66	2.61	2.61	2.63
R.300.100.30	0.85	0.57	0.49	0.41	0.37	0.29	0.26	0.23	0.22	0.20
R.300.100.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
R.300.1000.1	-1.16	-0.87	-0.57	-0.68	-0.34	-0.27	-0.05	0.12	0.01	0.25
R.300.1000.15	3.88	4.08	3.83	3.83	3.68	3.66	3.39	3.52	3.47	3.48
R.300.1000.30	0.79	0.65	0.58	0.51	0.50	0.49	0.46	0.45	0.43	0.39
R.300.1000.60	0.01	0.02	0.03	0.02	0.03	0.03	0.04	0.04	0.04	0.04
R.400.100.1	-7.43	-7.36	-5.89	-4.72	-4.31	-2.89	-4.09	-2.70	-0.57	-0.39
R.400.100.15	7.78	8.18	7.96	7.96	7.78	7.57	7.47	7.18	7.19	7.27
R.400.100.30	0.78	0.63	0.62	0.61	0.60	0.61	0.59	0.59	0.53	0.57
R.400.100.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
R.400.1000.1	-0.06	1.02	1.03	0.61	0.78	0.40	0.33	0.66	0.76	1.03
R.400.1000.15	7.36	7.02	7.00	6.95	6.76	6.59	6.42	6.27	6.16	6.07
R.400.1000.30	0.68	0.58	0.57	0.58	0.53	0.53	0.52	0.52	0.50	0.49

R.400.1000.60	0.02	0.05	0.06	0.06	0.07	0.08	0.10	0.03	0.04	0.04
R.500.100.1	-17.02	-10.75	-10.18	-13.17	-11.18	-9.86	-9.23	-7.73	-7.78	-7.28
R.500.100.15	7.89	7.67	7.49	7.31	7.31	7.33	7.24	7.05	7.01	6.96
R.500.100.30	1.53	1.24	1.07	1.05	1.03	1.03	0.98	0.96	0.94	0.95
R.500.100.60	0.06	0.03	0.02	0.01	0.01	0.01	0.01	0.02	0.02	0.02
R.500.1000.1	-1.19	-0.51	-0.99	-1.09	-1.45	-1.33	-0.75	-0.71	-0.83	-0.87
R.500.1000.15	6.90	6.91	6.64	6.58	6.56	6.53	6.44	6.26	6.13	6.14
R.500.1000.30	1.02	0.89	0.73	0.70	0.65	0.61	0.60	0.55	0.56	0.55
R.500.1000.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
R.600.100.1	-13.40	-12.66	-11.21	-7.63	-3.79	-3.61	-4.97	-10.16	-11.55	-11.66
R.600.100.15	9.93	9.88	9.95	10.08	10.23	10.33	10.26	10.37	10.30	10.32
R.600.100.30	1.10	1.10	1.03	0.93	0.88	0.86	0.83	0.80	0.78	0.76
R.600.100.60	0.00	0.00	0.00	0.00	0.01	0.02	0.03	0.03	0.04	0.04
R.600.1000.1	0.65	0.09	1.09	1.29	1.38	1.68	1.81	1.90	2.17	2.21
R.600.1000.15	8.34	8.06	8.16	8.27	8.01	7.99	7.94	7.80	7.61	7.44
R.600.1000.30	1.48	1.45	1.33	1.27	1.25	1.25	1.19	1.17	1.15	1.15
R.600.1000.60	-0.01	0.00	-0.01	-0.01	-0.02	-0.01	-0.02	-0.02	-0.02	-0.02
R.700.100.1	-5.77	-15.27	-19.21	-19.94	-21.79	-21.60	-18.07	-19.87	-19.87	-19.36
R.700.100.15	7.46	7.99	8.23	8.31	8.29	8.33	8.36	8.27	8.20	8.24
R.700.100.30	1.14	1.06	0.90	0.81	0.74	0.67	0.65	0.63	0.62	0.57
R.700.100.60	0.06	0.06	0.05	0.04	0.01	-0.03	-0.05	-0.08	-0.08	-0.08
R.700.1000.1	0.31	0.44	0.92	0.81	1.11	1.93	1.75	1.54	1.50	1.67
R.700.1000.15	9.81	9.91	9.74	9.44	9.47	9.51	9.22	9.19	9.14	9.04
R.700.1000.30	1.09	1.06	0.98	0.95	0.94	0.91	0.90	0.91	0.89	0.86
R.700.1000.60	0.04	0.04	0.04	0.04	0.05	0.05	0.05	0.05	0.05	0.05

Table 5.3: Percentage improvement of EAAS over EACS

Instance	1 min.	2 min.	3 min.	4 min.	5 min.	6 min.	7 min.	8 min.	9 min.	10 min.
R.200.100.1	-40.72	-38.75	-37.31	-36.69	-35.62	-33.32	-34.25	-33.85	-33.39	-32.94
R.200.100.15	-0.52	-0.20	-0.30	-0.40	-0.30	-3.31	-0.43	-0.55	-0.56	-0.53
R.200.100.30	0.12	0.09	0.10	0.10	0.10	0.00	0.10	0.08	0.08	0.07
R.200.100.60	-0.86	-0.84	-0.84	-0.84	-0.84	-0.84	-0.83	-0.83	-0.81	-0.81
R.200.1000.1	-21.97	-19.73	-18.55	-17.78	-16.92	-16.55	-15.95	-15.25	-14.73	-14.48
R.200.1000.15	2.79	2.61	2.60	2.50	2.52	-0.72	2.55	2.44	2.32	2.28
R.200.1000.30	0.12	0.08	0.08	0.06	0.04	-0.04	0.04	0.03	0.03	0.03
R.200.1000.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
R.300.100.1	-62.08	-61.69	-60.84	-60.34	-59.66	-58.60	-59.56	-59.00	-58.69	-58.29
R.300.100.15	1.81	1.98	2.11	2.27	2.38	-0.49	2.16	2.23	2.30	2.33
R.300.100.30	0.51	0.40	0.35	0.28	0.24	-0.12	0.14	0.12	0.12	0.10
R.300.100.60	-0.35	-0.35	-0.35	-0.34	-0.31	-0.29	-0.29	-0.29	-0.29	-0.29
R.300.1000.1	-28.95	-26.42	-24.55	-23.94	-23.20	-22.99	-22.49	-22.07	-21.95	-21.54
R.300.1000.15	3.01	3.22	3.54	3.61	3.52	-0.35	3.04	3.15	3.14	3.24
R.300.1000.30	0.58	0.44	0.38	0.32	0.32	-0.17	0.30	0.29	0.27	0.23
R.300.1000.60	-0.61	-0.58	-0.58	-0.58	-0.58	-0.61	-0.58	-0.59	-0.60	-0.60
R.400.100.1	-68.18	-68.62	-67.83	-67.73	-67.14	-65.57	-66.66	-66.50	-66.51	-66.58
R.400.100.15	5.91	7.02	7.04	7.39	7.22	-0.43	7.05	6.86	6.81	6.85
R.400.100.30	0.88	0.80	0.75	0.71	0.63	0.01	0.59	0.56	0.50	0.49
R.400.100.60	-0.78	-0.75	-0.75	-0.75	-0.75	-0.74	-0.73	-0.72	-0.72	-0.71
R.400.1000.1	-31.92	-30.64	-28.61	-27.31	-26.56	-26.29	-25.44	-24.46	-24.18	-23.95
R.400.1000.15	4.97	5.89	5.95	6.01	6.00	-0.70	5.78	5.67	5.62	5.58
R.400.1000.30	0.23	0.17	0.15	0.18	0.20	-0.35	0.17	0.20	0.20	0.21
R.400.1000.60	-0.31	-0.28	-0.27	-0.26	-0.26	-0.35	-0.25	-0.31	-0.31	-0.31
R.500.100.1	-69.56	-72.82	-75.02	-76.92	-77.25	-74.65	-77.39	-76.86	-76.73	-76.70
R.500.100.15	5.28	6.06	6.18	6.21	6.38	-0.86	6.60	6.57	6.51	6.59
R.500.100.30	0.86	0.77	0.67	0.72	0.73	-0.29	0.70	0.69	0.67	0.65
R.500.100.60	-0.62	-0.59	-0.58	-0.58	-0.58	-0.57	-0.56	-0.56	-0.55	-0.55
R.500.1000.1	-32.50	-32.55	-33.11	-31.52	-30.94	-29.13	-29.53	-28.36	-27.27	-26.94
R.500.1000.15	2.88	4.69	5.06	5.42	5.68	-0.72	5.89	5.76	5.69	5.76
R.500.1000.30	0.63	0.61	0.54	0.52	0.51	-0.13	0.46	0.42	0.42	0.44
R.500.1000.60	-0.64	-0.61	-0.59	-0.58	-0.58	-0.58	-0.57	-0.57	-0.57	-0.57
R.600.100.1	-69.97	-76.11	-78.96	-80.05	-80.94	-80.84	-82.32	-83.50	-83.86	-84.13

R.600.100.15	4.18	7.10	8.38	9.05	9.68	-0.31	9.96	10.07	10.12	10.16
R.600.100.30	0.41	0.53	0.55	0.50	0.48	-0.39	0.44	0.42	0.41	0.41
R.600.100.60	-0.75	-0.68	-0.66	-0.63	-0.62	-0.62	-0.59	-0.58	-0.58	-0.57
R.600.1000.1	-29.10	-31.13	-30.67	-31.12	-31.35	-31.79	-29.84	-29.47	-29.12	-28.70
R.600.1000.15	0.17	2.92	4.39	5.48	5.96	-1.68	6.36	6.30	6.19	6.20
R.600.1000.30	0.93	1.21	1.20	1.16	1.18	-0.07	1.12	1.12	1.10	1.06
R.600.1000.60	-0.76	-0.68	-0.67	-0.61	-0.61	-0.59	-0.60	-0.60	-0.59	-0.59
R.700.100.1	-66.38	-75.69	-79.66	-81.80	-83.59	-80.06	-84.71	-85.50	-86.10	-86.48
R.700.100.15	0.52	3.15	4.54	5.26	5.84	-2.01	6.50	6.77	6.86	7.03
R.700.100.30	-0.08	0.32	0.33	0.30	0.31	-0.41	0.25	0.24	0.23	0.21
R.700.100.60	-0.87	-0.73	-0.67	-0.64	-0.66	-0.66	-0.73	-0.74	-0.74	-0.73
R.700.1000.1	-28.41	-30.84	-31.83	-31.89	-32.00	-32.88	-31.70	-31.87	-32.03	-31.78
R.700.1000.15	0.93	3.67	5.51	6.21	6.81	-2.08	7.35	7.44	7.48	7.57
R.700.1000.30	0.20	0.53	0.65	0.70	0.75	-0.13	0.78	0.78	0.76	0.75
R.700.1000.60	-0.87	-0.83	-0.80	-0.78	-0.77	-0.80	-0.75	-0.74	-0.74	-0.73

In the following chart, we illustrate the percentage improvement provided by PEACS vs EACS at each one minute interval.

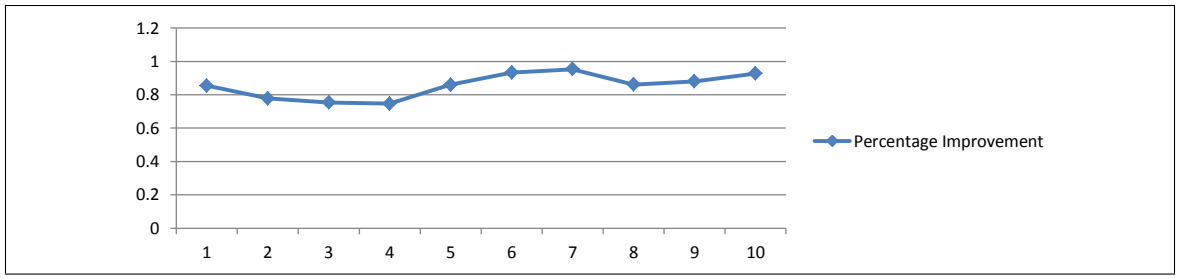

Figure 5-17: Percentage Improvement Over Time: PEACS vs. EACS

In the following chart, we illustrate the percentage improvement provided by EAAS vs EACS at each one minute interval for problems with precedence probability 15 & 30.

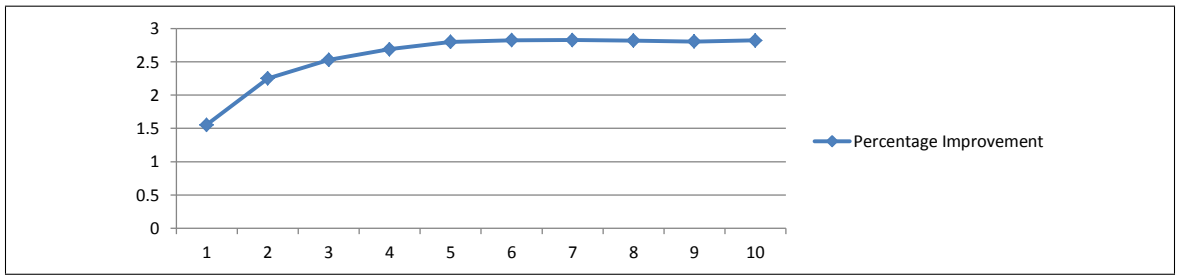

Figure 5-18: Percentage Improvement Over Time: EAAS vs. EACS

5.2.2 Statistical Significance

To assess the performance difference between each pair of methods, we needed to utilize a statistical measure. One way to do this is to take each problem and determine if one algorithm was better than the other "with a statistically significant difference", and if so, count it as a significant win for the winner. Counting the number of significant wins would give us an idea of what the better algorithm is, especially if one of them obtains either a significant win, or at worst a statistically insignificant loss. Tests of this manner include the two–sample Student t–test and the Wilcoxon rank–sum test.

Alternatively, we can take the average result returned for each problem instance by an algorithm, and match each result against its counterpart from another algorithm.

These pairs are then passed to a statistical test to asses whether one algorithm is better "with a statistically significant difference". Such tests include the one–sample Student t–test and the Wilcoxon signed–ranks test.

We opted to use the Wilcoxon signed–ranks test. One reason for this is that it is not dependent on the distribution of the differences between pair, unlike the Student t–test which favours normal distributions. The other is based on the following argument by Demšar [10]:

> Some authors prefer to count only the significant wins and losses, where the significance is determined using a statistical test on each data set.... The reasoning behind this practice is that "some wins and losses are random and these should not count." This would be a valid argument if statistical tests could distinguish between the random and non–random differences. However, statistical test only measure the improbability of the obtained experimental result if the null hypothesis was correct, which is not even the (im)probability of the null–hypothesis.
>
> For the sake of argument, suppose that we compared two algorithms on one thousand different data sets. In each and every case, algorithm A was better than algorithm B, but the difference was never significant. It is true that for each single case the difference between the two algorithms can be attributed to a random chance, but how likely is it that one algorithm was just lucky in all 1000 out of 1000 independent experiments?
>
> Contrary to the popular belief, counting only significant wins and losses therefore does not make the tests more but rather less reliable, since it draws an arbitrary threshold of $p < 0.05$ between what counts and what does not.

Two algorithms are compared in the Wilcoxon signed–ranks test as follows:

1. For each problem instance, calculate the difference between the average solution cost returned by the algorithms.

2. Discard differences with a value of zero.

3. Denote the number of remaining difference as N.

4. Rank the differences according to their absolute values in ascending order. Assign differences with same value the average of their ranks.

5. Denote the summation of the ranks representing positive differences as $W+$.

6. Denote the summation of the ranks representing negative differences as $W-$.

7. Denote $W = min(W+, W-)$.

8. Calculate the following:

$$z = \frac{W - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(N+2)}} \tag{5.1}$$

9. Calculate the probability value p corresponding to the value of z in a standard normal distribution (We can do this since N is large enough; for values of $N < 15$, we need to resort to critical value tables). A value of $p < 0.05$ indicates that the difference in performance between the two algorithms is statistically significant (the value 0.05 is a bit arbitrary; sometimes 0.1 is used).

The results of the Wilcoxon signed–ranks tests are shown in Table (5.4). In all but one case, p is several orders of magnitude less than 0.05 and we can conclude that there is a statistically significant difference for them. EACS performs significantly better than ACS. EAAS performs better than EACS if we take only the instances with precedence probability 15 & 30 into consideration. PEACS is the best algorithm, performing significantly than all other configurations, regardless of whether we take all instances or only precedence probability 15 & 30 into consideration.

5.3 Experiments on the exploitation parameter q_0

The parameter q_0 controls how often ACS opts to choose the next node deterministically, rather than a make a probabilistic choice biased by the knowledge gathered

Table 5.4: Results of Wilcoxon Signed-ranks Tests

Hypothesis	$W+$	$W-$	N	z	p-value
EACS vs. ACS	16	1019	45	-5.66	**2.00E-08**
PEACS vs. ACS	7	1028	45	-5.76	**8.00E-09**
PEACS vs. EACS	58.5	802.5	41	-4.82	**1.00E-06**
EAAS vs. ACS	376	800	48	-2.17	**3.01E-02**
EAAS vs. EACS	581	595	48	-0.07	9.47E-01
EAAS vs. EACS (1 & 60)	300	0	24	-4.28	**1.94E-05**
EAAS vs. EACS (15 & 30)	3	297	24	-4.2	**2.84E-05**
EAAS vs. PEACS	1176	0	48	-6.03	**1.69E-09**
EAAS vs. PEACS (1 & 60)	300	0	24	-4.28	**1.94E-05**
EAAS vs. PEACS (15 & 30)	300	0	24	-4.28	**1.94E-05**

by the system so far. Our modification in PEACS aimed to make ACS less exploitative. Another way to achieve the same goal is to decrease the value of q_0. Note that we ran all configurations for only 5 times per instance with the exception of the base EACS implementation, which ran for 30 times per instance. The results are presented in table (5.5).

Table 5.5: Results of q_0 experiment

Instance	EACS	PEACS	EACS w/ $s = 4$	EACS w/ $s = 10$	EACS w/ $q_0 = 5/6$
R.200.100.1	72.87	73.8	71.4	76.4	73.6
R.200.100.15	1889.53	1835.7	1923.6	1919.8	1842
R.200.100.30	4229.03	4222.5	4223.8	4229	4229
R.200.100.60	71749	71749	71749	71749	71749
R.200.1000.1	1458.9	1459.8	1470.2	1480.2	1459.4
R.200.1000.15	21503.37	20864.9	21500.4	21347.8	21113.6
R.200.1000.30	41223.33	41196	41196	41196	41196
R.200.1000.60	71556	71556	71556	71556	71556

R.300.100.1	43.6	43.67	43.4	44.4	46.8
R.300.100.15	3316.5	3231.43	3334.2	3306.2	3257.8
R.300.100.30	6132.17	6120	6133	6128.4	6120
R.300.100.60	9726	9726	9726	9726	9726
R.300.1000.1	1449.93	1446.27	1463.6	1477.2	1495.4
R.300.1000.15	31108.43	30062	31040.6	31262	30070
R.300.1000.30	54396.73	54183.27	54340.4	54247.8	54269.8
R.300.1000.60	109590	109549.23	109590	109566.2	109590
R.400.100.1	34.3	34.43	31	34.6	34.2
R.400.100.15	4323.67	4030.5	4247.6	4224.8	4070.8
R.400.100.30	8230.27	8183.7	8207.6	8191	8181
R.400.100.60	15228	15228	15228	15228	15228
R.400.1000.1	1533.2	1517.63	1518.6	1547.6	1670.8
R.400.1000.15	42670.97	40228.4	42145.4	41859.8	40644.2
R.400.1000.30	85738.77	85323.67	85426.6	85589.4	85340.6
R.400.1000.60	140931.5	140877.6	140816	140816	140908.4
R.500.100.1	23.37	25.2	22.2	19.4	29.2
R.500.100.15	5892.1	5508.67	5840.6	5862.6	5555.2
R.500.100.30	9765.43	9673.8	9770.8	9751.6	9710.8
R.500.100.60	18267.2	18263.8	18264.6	18251.8	18261.6
R.500.1000.1	1546.87	1560.47	1554.2	1548.6	1884
R.500.1000.15	54705.2	51541.13	55205.4	55104.2	52148.8
R.500.1000.30	99692.3	99142.17	99499.6	99495.4	99209.6
R.500.1000.60	178212	178212	178212	178212	178212
R.600.100.1	16.07	18.3	12.8	13.6	77.6
R.600.100.15	6442.47	5839.7	6437.2	6387.2	5975.8
R.600.100.30	12569.37	12474.43	12553	12524.6	12493.6
R.600.100.60	23326	23317.2	23299.6	23293	23299.6
R.600.1000.1	1646.57	1611	1629.8	1633	2193.8
R.600.1000.15	62188.23	57880.77	63181.6	62993.6	59006.6
R.600.1000.30	128613.23	127153.73	128206	127788	127346.6
R.600.1000.60	214700.93	214747.97	214652.2	214608	214652.2
R.700.100.1	11.77	14.63	8.6	11.4	72

R.700.100.15	8021.33	7410.5	8087.6	7956	7594.4
R.700.100.30	14613.23	14530.4	14627	14615.6	14559.4
R.700.100.60	24124	24143.6	24136	24104.8	24122.8
R.700.1000.1	1614.23	1587.27	1619.2	1639	2421
R.700.1000.15	74680.07	68490.43	74602	74874.2	70176.4
R.700.1000.30	135868.53	134710.87	135680.8	135236.2	134877.2
R.700.1000.60	245780.63	245655.33	245665.8	245634	245634

The results show that changing the value of s from the original value of 5 to either 4 or 10 seems to improve the results considerably for a subset of the instances. The improvement in those problems was just enough to show up in our statisical test as significant, but keep in mind that the modified EACS configurations were only run for 5 times per instance, so we cannot claim this to be conclusive. An interesting outcome came from fixing q_0 at the considerably lower value of 5/6 (approximately 83%). This improved the average solution quality of EACS against most of the instances, but caused a massive decrease of quality against others. Even at its best, though, it was still considerably outdone by PEACS. This is verified by the Wilcoxon test results in table (5.6).

Table 5.6: Results of Wilcoxon Signed-ranks Tests for q_0 experiment

Hypothesis	$W+$	$W-$	N	z	p-value
EACS vs. EACS w/ $s = 4$	638	265	42	-2.33	**2.00E-02**
EACS vs. EACS w/ $s = 10$	676	270	43	-2.45	**1.45E-02**
EACS vs. EACS w/ $q_0 = 5/6$	707	196	42	-3.19	**1.43E-03**
PEACS vs. EACS w/ $q_0 = 5/6$	84	777	41	-4.49	**7.34E-06**

Chapter 6

Analysis

We have shown in the previous chapter how both of our contributions show a significant improvement over the current state of the art method of solving the sequential ordering problem, EACS, in a considerable number of instances. We provide a little discussion as to why.

With regards to PEACS, we hypothesized that this could be a result of affecting one or more of the following:

1. Increased diversity, or more formally an increase in the average edge distance between solutions generated in a single iteration

2. An increase of the rate of "good" solutions generated. A solution is considered if it falls under the 20% difference threshold

These two factors can be affected by manipulating the value of q_0. In general, q_0 was found in our experiments to be proportional to diversity, and inversely proportional to the generation rate of good solutions.

We were unable to find a pattern that is consistent across all instances. Exhibiting a greater solution diversity and good solution rate did not guarantee better solution quality for the same time interval. Exhibiting a higher value in only one of them did not result in consistently better or worse solution quality either. At this point, we can only speculate that under certain edge desirability matrix conditions, PEACS,

with its lower exploitation bias, has a much easier time at finding improvements than standard EACS.

As for EAAS, we can say in short is that its philosophy is that the desirability of an edge should depend entirely on the quality of end solutions it was a part of. This technique so far seems to help it against certain conditions more than others. Of course, our implementation is merely a quick attempt at incorporating the EigenAnt idea against a full-fledged problem, rather than a simple abstract shortest path problem. Due to this, our implementation suffers from the fact that updating the pheromone trail on an edge is actually equivalent to updating the pheromone trails on multiple paths, rather than just one. This is probably the reason why it performs poorly against some of the instances, and to why it still loses to PEACS in all of them.

Chapter 7

Future work

Future work on the algorithms we presented would include further analysis into the reasons they give such positive results. In particular, we will try to investigate into why problems with precedence 15 & 30 greatly benefit from our configurations and not the rest. Possible metrics to analyze would be the pheromone distribution and the edge distance between solutions.

Another thing to look into with regards to PEACS is how much does evaporation affect solution quality. Specifically, we can try to eliminate local evaporation completely and measure the performance. Of course, in this case it would probably stagnate quickly, but some small experiments we conducted have shown that the system generates some very good solutions before that. Experimenting with q_0 is another track we can undertake.

EAAS provides room for exploration as well. Our configuration was one made very quickly using our EACS implementation. It currently suffers from the fact that the actual paths intersect, and hence do not have entirely independent pheromone levels. Finding a way to separate them should lead to an interesting effect on solution quality. Merging PEACS and EAAS together is of course another track we can take.

Bibliography

[1] A. Bhaya, Jayadeva, R. Kothari, S. Chandra, S. Shah, "Ants find the shortest path: a mathematical proof", *Swarm Intelligence*: 1-20

[2] A. Ezzat, A.M. Abdelbar, "A Bare-Bones ACO Algorithm that Performs Competitively on the Sequential Order Problem", *IEEE Transactions on Evolutionary Computation*, under review

[3] A. Ezzat, A.M. Abdelbar, "A Less Exploitative Variation of the Enhanced Ant Colony System Applied to SOP", *IEEE Congress on Evolutionary Computation*, *to appear*, 2013

[4] A.M. Abdelbar, "Stubborn ants," *Proceedings IEEE Swarm Intelligence Symposium* (SIS-08), pp. 1-5, 2008.

[5] A.M. Abdelbar, "Is there a computational advantage to representing evaporation rate in ant colony optimization as a gaussian random variable?," *Proceedings Fourteenth International Conference on Genetic and Evolutionary Computation Conference* (GECCO-12), pp. 1-8. 2012.

[6] A.M. Abdelbar, and D.C. Wunsch, "Improving the performance of MAX-MIN ant system on the TSP using stubborn ants," *Proceedings Fourteenth International Conference on Genetic and Evolutionary Computation* (GECCO-12) Conference Companion, pp. 1395-1396, 2012.

[7] D. Anghinolfi, R. Montemanni, M. Paolucci, and L.M. Gambardella, "A particle swarm optimization approach for the sequential ordering problem," *Proceedings VIII Metaheuristic International Conference* (MIC 2009), 2009.

[8] D. Anghinolfi, R. Montemanni, M. Paolucci, and L.M. Gambardella, "A hybrid particle swarm optimization approach for the sequential ordering problem," *Computers and Operations Research*, Vol. 38, No. 7, pp. 1076–1085, 2011.

[9] N. Ascheuer, "Hamiltonian path problems in the on-line optimization of flexible manufacturing systems," PhD Thesis, Technische Universitat Berlin, 1995.

[10] J. J. Bentley, "Fast algorithms for geometric traveling salesman problems," *ORSA Journal on computing* 4.4 (1992): 387-411.

[11] B. Bullnheimer, R.F. Hartl, and C. Strauss, "An improved ant system algorithm for the vehicle routing problem," *Annals of Operations Research*, Vol. 89, pp. 25–38, 1999.

[12] S. Chen, and S. Smith, "Commonality and genetic algorithms," Technical Report CMURI-TR-96-27, The Robotic Institute, Carnagie Mellon University, 1996

[13] O. Cordón, I. F. de Viana, and F. Herrera, "Analysis of the best-worst ant system and its variants on the TSP," *Mathware and Soft Computing*, Vol. 9, No. 2-3, pp. 177-192, 2002.

[14] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Machine Learning Research*, Vol. 7, pp. 1-30, 2006.

[15] M. Dorigo, and L.M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp. 35–66, 1997

[16] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperative agents," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 26, No. 1, pp. 29-41, 1996.

[17] M. Dorigo, and T. Stützle, *Ant Colony Optimization*, MIT Press, Cambridge, 2004.

[18] M. Dorigo, and T. Stützle, "Ant colony optimization: overview and recent advances," In: M. Gendreau, and Y. Potvin, eds., *Handbook of Metaheuristics*, 2nd edition, Springer-Verlag, New York, pp. 227–263, 2010.

[19] L.F. Escudero, "An inexact algorithm for the sequential ordering problem," *European Journal of Operational Research*, Vol. 37, pp. 232–253, 1988.

[20] L.M. Gambardella, and M. Dorigo, "An ant colony system hybridized with a new local search for the sequential ordering problem," *INFORMS Journal on Computing*, Vol.12, No. 3, pp. 237–255, 2000.

[21] L.M. Gambardella, R. Montemanni, and D. Weyland, "Coupling ant colony systems with strong local searches," *European Journal of Operational Research*, Vol. 220, No. 3, pp. 831–843, 2012.

[22] F. Guerriero, and M. Mancini, "A cooperative parallel rollout algorithm for the sequential ordering problem," *Parallel Computing*, Vol. 29, No. 5, pp. 663–677, 2003.

[23] G. Kindervater, and M. Savelsbergh, "Vehicle routing: handling edge exchanges," In: E.H.L. Aarts, and J.K. Lenstra, eds., *Local Search in Combinatorial Optimization*, Wiley, Chichester, pp. 337–360, 1997.

[24] S. Lin, "Computer solutions of the traveling salesman problem," *Bell Systems Technical Journal*, Vol. 44, pp. 2245–2269, 1965.

[25] S. Lin, and B.W. Kernighan, "An effective heuristic algorithm for the traveling–salesman problem," *Operations Research*, Vol. 21, pp. 498–516, 1973.

[26] R. Montemanni, SOPLIB2006 Problem Instance Library. URL `http://www.idsia.ch/~roberto/SOPLIB06.zip`

[27] R. Montemanni, D.H. Smith, and L.M. Gambardella, "Ant colony systems for large sequential ordering problems," *Proceedings IEEE Swarm Intelligence Symposium* (SIS-07), pp. 60-67, 2007

[28] R. Montemanni, D.H. Smith, and L.M. Gambardella, "A heuristic manipulation technique for the sequential ordering problem," *Computers and Operations Research*, Vol. 35, No. 12, pp. 3931–3944, 2008.

[29] R. Montemanni, D.H. Smith, A.E. Rizzoli, and L.M. Gambardella, "Sequential ordering problems for crane scheduling in port terminals," *International Journal of Simulation and Process Modelling*, Vol. 5, No. 4, pp. 348–361, 2009.

[30] I. Or, "Traveling salesman–type combinatorial problems and their relation to the logistics of regional blood banking," PhD Thesis, Northwestern University, 1976.

[31] W. Pullyblank, and M. Timlin, "Precedence constrained routing and helicopter scheduling: heuristic design," Technical Report RC17154 (#76032), IBM T.J. Watson Research Center, 1991

[32] M.W.P. Savelsbergh, "An efficient implementation of local search algorithms for constrained routing problems," *European Journal of Operational Research*, Vol. 47, pp. 75–85, 1990.

[33] D.I. Seo, and B.R. Moon, "A hybrid genetic algorithm based on complete graph representation for the sequential ordering problem," *Proceedings International Conference on Genetic and Evolutionary Computation* (GECCO-03), pp. 69–680, 2003.

[34] T. Stützle, ACOTSP: a software package for various ant colony optimization algorithms applied to the symmetric traveling salesman problem, URL `http://www.aco-metaheuristic.org/aco-code/`

[35] T. Stützle, and H. Hoos, "MAX–MIN ant system," *Future Generation Computer Systems*, Vol. 16, No. 8, pp. 889–914, 2000.