Worm Optimization: A novel optimization algorithm inspired by C. Elegans

Jean-Paul Arnaout
Department of Business Administration
Gulf University for Science and Technology
West Mishref, Kuwait

Abstract

Within the arena of Swarm Intelligence, this research introduces a novel optimization algorithm (*WO*) inspired by the foraging behaviors of *Caenorhabditis elegans (Worms)*. *WO* effectiveness is illustrated on the traveling salesman problem (TSP), a known NP-hard problem, and compared to Simulated Annealing (SA) and Genetic Algorithm (GA) using existing TSP data. The computational results reflected the superiority of *WO* in all tested problems. Furthermore, this superiority improved as problem sizes increased, and *WO* attained the global optimal solution in all tested problems.

Keywords
Worm Optimization, Metaheuristics, Swarm Intelligence

1. Introduction
In recent years, and as optimization problems are becoming more complex and exact solution approaches computationally infeasible, the arena of biologically inspired computing is becoming quite popular. The latter links several subfields together such as social behavior and emergence, relying heavily on the fields of biology, computer science and mathematics with an objective of using computers to model living phenomena and concurrently analyzing life to improve the usage of computers (Bongard 2009). In particular, the discipline of swarm intelligence is receiving a great deal of attention from the operations research community as it has delivered good and efficient solutions to NP-hard problems that would not be possible using traditional optimization approaches. Swarm intelligence concentrates on the cooperative and collective behaviors resulting from the interactions of individuals among themselves and with the environment, including examples such as colonies of ants, bees, and termites, flocks of birds, and schools of fish (Dorigo and Birattari 2007). Following the same body of knowledge, we propose in this research a novel optimization algorithm called Worms Optimization that is based on the behaviors of *Caenorhabditis elegans* (informally known as 'the worm').

Caenorhabditis elegan is a small, soil-dwelling nematode with only 302 neurons (Macosko et al. 2009). Nevertheless, these neurons allow *C. elegans* to achieve several intricate behaviors including finding food (Avery and You 2012), avoiding toxins (Xu and Deng 2012), interchanging between solitary and social foraging styles (Lockery 2009), alternating between “dwelling - food exploiting” and “roaming - food seeking” (Shtonda and Avery 2006), and entering a type of stasis stage (Hu 2007).

Worms Optimization algorithm (*WO*) effectiveness is illustrated by solving the Traveling Salesman Problem (TSP) that deals with finding the shortest closed tour after visiting all the cities once and only once in a given set. The main difficulty in solving TSP lies in the great number of possible tours $(n-1)!/2$ for n cities (Larranaga et al. 1999). Consequently, the TSP is known to be NP-hard; i.e. no algorithm can solve all instances, especially large ones, in a practical computational time. While exact solution approaches have been used for small problems, the literature presents lots of heuristics and metaheuristics that were developed to solve TSP problems approximately. For further reading on this topic, the reader can refer to Whitley et al. 1989, Lin et al. 1993, and Basu 2012.

The rest of this paper is organized as follows. In Section 2, we explain the key behaviors of worms. In Section 3, the Worms Optimization algorithm is introduced. The computational tests are presented in Section 4 and finally we conclude this research in Section 5.

2. C. elegans: Behaviors and Characteristics

In this section, we describe the key behaviors of C. elegans along with the associated neurons. In particular, we will discuss feeding, toxins avoidance, foraging styles, dwelling and roaming, and dauer arrest.

2.1 Feeding

Worms are able to distinguish food based on its quality and will constantly seek out higher quality food (Shtonda and Avery 2006). Furthermore, when a worm is exposed to two different types of bacteria (food), it will select always and with no exceptions the better source.

Having said this, the worm when modeled will follow a greedy rule in selecting solutions, as it always moves to the better solution without considering the global bests.

2.2 Foraging Styles

One of the worm's neurons is the RMG inter/motor neuron, which controls aggregation and related behaviors (Macosko et al. 2009). In particular, high RMG activity indicates a social behavior, while a solitary behavior is observed with low RMG activity. Having said this, social strains are attracted by pheromone secreted by other worms as well as good quality food, while solitary strains are repulsed by this pheromone and dispersed evenly across all types of food.

Depending on the RMG level, a worm could be either "social, attracted to pheromone, and drawn to good solutions (greedy rule)" or "solitary, repulsed by pheromone, and evenly dispersed across all solutions".

2.3 Toxins Avoidance

Zhang et al. (2005) showed that worms can learn to avoid odors associated with infection (bad food) through sensory neurons referred to as ADF. Once a worm gets infected, it will learn from this infection in the sense not to eat the same bad food again.

The artificial worm will be equipped with a tabu like list that stores a certain number of bad solutions.

2.4 Dwelling and Roaming

Shtonda and Avery (2006) reported that an extrapharyngeal interneuron called AIY is critical for a worm to alternate between dwelling (local search) and roaming (global search). In particular, at lower AIY activity levels, the worm switched from "roaming – food seeking" to local search; the authors stated that this could be due to the worms perceiving the food to be of higher quality than it was.

Depending on the AIY level, a worm could switch to local search. Furthermore, if the food is of low quality, the worm will leave; i.e. will stop the local search.

2.5 Dauer Arrest

In harsh conditions, a worm enters a Dauer stage, where the latter describes an alternative developmental stage by going from a reproductive stage into a type of stasis and eventually dying if the environment conditions remain unfavorable. The impacting conditions are many, but the prevalent ones are quality of food and increase in number/concentration of worms. In particular, if the present food quality is low and the concentration level is high; i.e. too many worms for low levels of food, then the worm enters dauer arrest. On the other hand, if the concentration level is low and the food supply is high, then the worms will reproduce.

Depending on the dauer status, the worm will move from a reproductive stage to a declining one, which would end once no more worms exist.

3. Worm Optimization (WO)

In this section, we translate the worm's behaviors discussed in Section 2 to the Worm Optimization algorithm; the latter is illustrated on the TSP problem. First, it is worth indicating here that similarly to other swarm intelligence algorithms, and in order to solve an optimization problem using *WO*, the previous can be represented as a connected graph (nodes and arcs). The worm will move from one node to another in order to create a solution. The parameters

needed in *WO* are as shown in Table 1. The actual values of the parameters are determined using Design of Experiments as shown later in Section 5.

Table 1: Parameters of *WO*

Parameter	Range	Description
Worm	(3 , 60)	worm number: number of worms in the algorithm (at the initialization stage)
MaxWorm	(3 , 60)	worm number: number of worms in the algorithm (at the initialization stage)
RMG	(0.2 , 0.8)	lever between local and global search: RMG = 1 indicates only social behavior and RMG =0 only solitary. The range is not below 0.2 or higher than 0.8 as to not exclude a particular behavior.
AIY	(0.1 , 0.5)	percentage of local search: the higher AIY, the higher the chance of dwelling (local search)
AIYIter	(5 , 60)	local search iterations: indicating the max number of local search iterations.
τ_{ij}	(0.01 , 10)	pheromone: initial amount of pheromone depositied on arc (i,j)
ϕ	(0.01 , 0.3)	production rate: rate of reproduction of worms when they are not in Dauer Stage.
λ	(0.01 , 0.5)	bad solution factor: initially, the arc attractiveness (ADF) for all arcs equals 1; i.e. each arc has an equal probability of being selected. In the case of a bad solution, it's associated arcs will be assigned an ADF = λ within the shown range in order to decrease its selection probability.
ρ	(0.01 , 0.4)	pheromone update: amount by which the pheromone is increased when a good solution is encountered.
MaxIteration	(5000, 15000)	Number of tours: referring the total number of worms (tours) generated in *WO* .

3.1 Feeding Modeling

Initially, a certain amount of pheromone (τ_{ij}) is deposited in each arc in the network. The worms move from a node to another according to a probability; the latter is partially analogous to the one found in Ant Colony Optimization (Arnaout et al. 2010), with the addition of worms specialized attributes. The probability is determined by three factors: pheromone amount (τ), visibility (η), and bad solution factor (*ADF*). The probability of moving from node *i* to *j* for worm *k* can be calculated as shown in (1):

$$P_{ij}^{k} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta} ADF_{ij}}{\sum_{l \in \Psi} \tau_{il}^{\alpha} \eta_{il}^{\beta} ADF_{ij}}, \quad (1)$$

where Ψ represents the set of unvisited nodes and the visibility (η) is determined depending if it is a social or solitary worm (explained in detail in Section 3.3). Two important parameters to direct the search are α and β, which are the exponents in the probability function that determine the importance of the pheromone amount over the visibility amount. Subsequently, when a worm finishes its tour; i.e. already visited all cities (nodes) and reported a solution, if the latter is better than the best solution found so far (*BestCost*), then the worm's tour (associated arcs) pheromone is updated according to Equation (2).

$$\tau_{ij} \leftarrow \tau_{ij} + \rho \qquad if\ arc(i,j)\ is\ used\ by\ worm\ k \quad (2)$$

3.2 Foraging Styles Modeling

After *WO* initialization, and according to *RMG*, each worm is labeled as social or solitary. If the worm is social, it will be attracted to pheromone and will move according to the greedy rule; if the worm is solitary, it is repelled by pheromone and will move with equal probabilities to possible cities. The pseudo code is summarized below:

Step 1: Generate random variable (*rv*) from U(0,1)
Step 2: If $(rv \leq RMG)$, then worm is social:
  Step 2.1: for (i = 1, …, N; j = 1, …, N), $\eta_{ij} = {}^{1}/_{d(i,j)}$,
  Step 2.2: Equation (1) becomes: $P_{ij}^{k} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta} ADF_{ij}}{\sum_{l \in \Psi} \tau_{il}^{\alpha} \eta_{il}^{\beta} ADF_{ij}}$.

Step 3: ElseIf($rv > RMG$), then worm is solitary:

Step 3.1: for ($i = 1, \ldots, N; j = 1, \ldots, N$), $\eta_{ij} = 1/N$, //$\eta_{ij} = 1/N$ to ensure equal probabilities//

Step 3.2: Equation (1) becomes: $P_{ij}^{k} = \dfrac{\left(1/\tau_{ij}\right)^{\alpha} \eta_{ij}^{\beta} ADF_{ij}}{\sum_{l \in \Psi} \left(1/\tau_{il}\right)^{\alpha} \eta_{il}^{\beta} ADF_{ij}}$.

where $d(i,j)$ refers to the Euclidean distance between cities i and j.

3.3 Toxins Avoidance – ADF Modeling

After a worm finishes its tour, its solution is assessed to determine if it should be added to the list of bad solutions or not. In this stage, we need to define an ADF list ($ADFList_{k,N+1}$) to store the inferior solutions. This list can store up to k solutions, where $k = \lceil \sqrt{Worm} \rceil$; i.e. k is always changing depending on the dauer status (reproductive versus declining population). N is needed for every row to store the ordering of the N cities (nodes) in the solution and an additional cell to store the solution cost (*WormCost*). Furthermore, we define the array $ADF_{N,N}$ to be linked with ADFList in the sense that for every solution stored in the list, its associated arcs are updated following $ADF_{i,j} = \lambda$, where λ is the bad solution factor. The pseudo code for the ADF modeling, which is applied for every worm, is shown below:

Step 1: Sort *ADFList* in the ascending order according to *WormCost* //$ADFList_{1,N+1}$ stores the worst solution//

Step 2: Assess current worm's solution (*WormCost*) //$ADFList_{k,N+1}$ stores the best worst solution//

Step 2.1: If ($WormCost > ADFList_{1,N+1}$), then add worm's tour to *ADFList*,

Step 2.2: ElseIf ($WormCost > ADFList_{k,N+1}$), replace row k in the list with the current worm's tour,

Step 2.3: Else, $ADFList_{k,i} = 0$, for $i = 1, \ldots, N+1$. //Remove the best worst solution//

Step 3: for ($l = 1, \ldots, k$; $i = 1, \ldots N$),

Step 3.1: If ($ADFList_{l,i} <> 0$), then $ADF_{l,i} = \lambda$. //Reduce *ADF* from 1 to λ//

Step 4: for ($i = 1, \ldots, N$; $j = 1, \ldots N$), Do: $\sum ADF = \sum ADF + ADF_{ij}$,

Step 4.1: If ($\sum ADF == N * \lambda$), then: //Randomly release a starting arc//

Step 4.1.1: choose a random city c,

Step 4.1.2: $ADF_{i,c} = 1$.

Note that in Step 2.3, and in case the solution generated was not worse than the last tour stored in *ADFList* (which represents the best bad solution), then the latter is released from *ADFList* back into the search space. In Step 4, if $\sum ADF == N * \lambda$, this means that all candidate starting arcs in a tour have been added to the *ADFList*; we release a random arc in order to continue the search.

3.4 Dwelling and Roaming – AIY Modeling

When a worm generates a tour and its cost (*WormCost*), it has a probability of conducting local search. In this case, two cities from the worm solution are chosen at random and swapped. If this leads to better solution, then the latter is retained; otherwise, the local search continues until either a better solution is found or maximum iterations (*AIYIter*) is reached. In summary, after each worm completes its tour, the following is checked:

Step 1: Generate random variable (rv) from U(0,1).

Step 2: If ($rv \leq AIY$), Do Local Search:

Step 2.1: for ($i = 1, \ldots,$ *AIYIter),* Do:

Step 2.1.1: choose 2 different cities (randomly) from the worm solution (*city1, city2*),

Step 2.1.2: swap *city1* and *city2* in the worm solution,

Step 2.1.3: if *WormCost* improved, then update solution; else, go to Step 2.

3.5 Dauer Arrest Modeling

Depending on the search space's solutions quality and worm concentration levels, a worm will enter Dauer. The food quality is assessed after each worm finishes its tour using Equation (3). As can be seen, when the *WormCost* isn't less than *BestCost* (i.e. solution not improving), $FQ = 1$.

$$Food\ Quality\ (FQ) = min\left\{1, \left(1 + \frac{WormCost - BestCost}{BestCost}\right)\right\} \quad (3)$$

As for the concentration of worms in the search space, it is better assessed by the number of tours generated (i.e. worms); for each worm, the number of iterations is updated until it reaches its maximum (*MaxIteration*), which indicates high concentration of worms. Equation (4) is used to normalize the concentration to a (0,1) scale, where *WCt* = 1 indicates that *WO* reached the maximum number of iterations.

$$Worms\ Concentration\ (WCt) = \frac{Current\ Iteration}{MaxIteration} \quad (4)$$

Following each worm's tour, the Dauer status (*DauerStatus*) is assessed as follows:
Step 1: Compute *FQ* and *WCt* according to Equations (3) and (4).
Step 2: Compute Dauer level: $DauerStatus = \frac{FQ+WCt}{2}$.
Step 3: Assess if worms are in reproductive or declining stage:
Step 3.1: If $(DauerStatus < 1)$, then $Worm = \min\{MaxWorm, \lceil Worm * (1+\phi)\rceil\}$,
Step 3.2: If $(DauerStatus == 1)$, then $Worm = \lfloor Worm * (1-\phi)\rfloor$.
Step 4: If $(Worm == 0)$, Stop *WO* and report *BestCost*.

3.6 Worm Optimization (WO) Modeling Summary

According to the above, *WO* pseudo code can be summarized as follows:

Step 1: Initialize *WO* Parameters and populate τ_{ij} and ADF_{ij} with the specified amounts

Step 2: While $((Worm \neq 0)$, Do:
Step 2.1: *Iteration* = *Iteration* + 1;
Step 2.2: Solve for a tour according to Section 3.2 (Social versus Solitary)
Step 2.3: Find *WormCost* associated with Step 2.2
Step 2.4: Execute the local search approach (according to *AIY*) described in Section 3.4
Step 2.5: Update the pheromone according to Equation (2)
Step 2.6: Execute the bad solutions approach described in Section 3.3 (*ADFList* and *ADF*)
Step 2.7: Update *Worm* number according to the Dauer approach in Section 3.5

Step 3: Output the best tour and its cost, Stop *WO*.

4. Computational Tests

The proposed *WO* was implemented in Microsoft Visual C++ 6.0 running on Windows XP with a Pentium 4 processor. Design of Experiments (DoE) was utilized to determine the appropriate values for the *WO* parameters that will minimize the tour cost. Numerous publications provide a good review of DoE (e.g., Fisher 1960; Taguchi 1993; NIST/SEMATECH 2006).
The factors considered in this experiment with their levels of low and high are presented in Table 1 in addition to α and β from Equation (1) with the associated range (1,3). The values of the parameter levels were selected based on many runs under different settings. To reduce the number of runs but reach sound conclusions, D-Optimal Design was utilized, which has been shown to be an effective design (NIST/SEMATECH 2006). JMP 10.0 from SAS was used to generate a D-Optimal design, with 112 experiments. The factors along with their interactions were analysed using regression, ANOVA, and factors' effect tests. Three-factor interactions and higher were not considered as they typically have weak effect (Ross 1996). Based on a 95% Confidence Interval, a relatively large t-Stat, and a small *p*-value (less than 0.05), the following parameter values were determined to provide the best performance for *WO*: $Worm = 40, MaxWorm = 7438, RMG = 0.55, AIY = 0.36, AIYIter = 38, \tau_{ij} = 0.01, \phi = 0.3, \lambda = 0.01,$ $\rho = 0.01, \alpha = 1.5, \beta = 2.5, and\ BestIter = 125.$

To test the performance of *WO*, it was compared to Genetic Algorithm (GA) and Simulated Annealing (SA). The GA results were obtained from Whitley et al. (1989) for Oliver30, Eil50, Eil75, and Bersini et al. (1995) for KroA100. SA results were reported in Lin et al. (1993). The problems selected are known TSP data and can be found in TSPLIB. The results are shown in Table 2, and it can be seen that *WO* outperformed GA and SA in all problem sizes. Furthermore, *WO* attained the optimal solution in all test problems. Even though *WO* outperformed SA and GA in all problem instances, and as the latter's solutions appeared to be close to *WO* (Table 2), the

percentage deviation (δ) of the algorithms (SA and GA) from *WO* was used as a measure of performance for each problem. That is:

$$\delta = \frac{TourCost_{Algorithm} - TourCost_{WO}}{TourCost_{WO}} \times 100\% \quad (5)$$

Table 2: Comparison of *WO* with *GA* and *SA*

Problem Name	WO	GA	SA	Optimum
Oliver30 (30-city problem)	420	421	424	420
Eil50 (50-city problem)	425	428	443	425
Eil75 (75-city problem)	535	545	580	535
KroA100 (100-city problem)	21282	21761	N/A	21282

Figure 1 shows the values of δ for all problems. It can be seen that SA performed the worst, followed by GA. Furthermore, both algorithms' deviation from WO increased with problem sizes; in particular, SA was not able to reach a solution within a feasible computational time for KroA100.

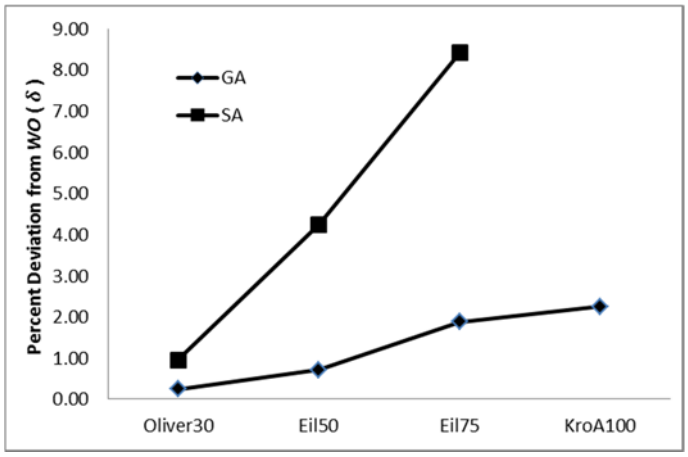

Figure 1: Deviation of SA and GA from *WO* ($\boldsymbol{\delta}$)

5. Conclusions and Future Research

In this paper, we have introduced a novel optimization algorithm, *WO*, which is inspired by the foraging behaviors of C. elegans (worms). *WO* performance was illustrated on the Traveling Salesmen Problem, a known NP-hard problem. The algorithm was compared to simulated annealing and genetic algorithm, and *WO* outperformed the algorithms in all problems, as well as attaining the optimal solution in all cases.

While the conducted preliminary experiments show very promising results, more testing is needed on larger problems. In particular, in future research, we will compare *WO* to more advanced metaheuristics such as Ant Colony and Bee Optimization under larger problems. Furthermore, the computational times need to be assessed to determine the practicality of *WO*.

References

Arnaout, J-P., Rabadi, G., and Musa, R., A two-stage Ant Colony Optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times, Journal of Intelligent Manufacturing, vol. 21, no. 6, pp. 693 – 701, 2010.

Avery, L., and You, Y.J., *C. elegans* feeding (May 21, 2012), *WormBook*, ed. The *C. elegans* Research Community, WormBook, Available: http://www.wormbook.org/chapters/www_feeding/feeding.pdf, May 21, 2012.

Basu, S., Tabu Search Implementation on Traveling Salesman Problem and Its Variations: A Literature Survey, *American Journal of Operations Research*, vol. 02, nb. 02, p.163, 2012.

Bersini, H., Oury, C., and Dorigo, M., Hybridization of Genetic Algorithms. *Tech. Rep. No. IRIDIA 95-22*, IRIDIA, Université Libre de Bruxelles, Belgium, 1995.

Bongard, J., Biologically Inspired Computing, *Computer*, vol. 42, no. 4, pp. 95-98, 2009.

Dorigo, M., and Birattari, M., Swarm intelligence, *Scholarpedia*, vol. 2, no. 9, pp. 1462, 2007.

Fisher, R.A., The Design of Experiments, New York: Hafner Publishing Company, 1960.

Hu, P.J., Dauer, WormBook, ed. The C. elegans Research Community, WormBook, Available: http://www.wormbook.org/chapters/www_dauer/dauer.pdf, August 08, 2007.

Larranaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I., and Dizdarevic, S., Genetic Algorithms for the Traveling Salesman Problem: A Review of Representations and Operators, *Artificial Intelligence Review*, vol. 13, pp. 129 – 170, 1999.

Lin, F.–T., Kao, C.–Y., and Hsu, C.–C., Applying the genetic approach to simulated annealing in solving some NP-hard problems. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, pp. 1752–1767, 1993.

Lockery, S., A social hub for worms, *Nature*, vol. 458, pp. 1124 – 1125, 2009.

Macosko, E., Pokala, N., Feinberg, E., Chalasani, S., Butcher, R., Clardy, J., and Bargmann, C., A hub-and-spoke circuit drives pheromone attraction and social behavior in C. elegans, *Nature*, vol. 458, no. 30, pp. 1171 – 1176, 2009.

NIST/SEMATECH, e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/, Available: 25 June 2008.

Ross, P., *Taguchi Techniques for Quality Engineering*, New York: McGraw Hill, 1996.

Shtonda, B.B., and Avery, L., Dietary choice behavior in Caenorhabditis elegans, *The Journal of Experimental Biology*, vol. 209, pp. 89 – 102, 2006.

Taguchi, G., *Taguchi Methods: Design of Experiments*, Michigan: American Supplier Institute, Inc, 1993.

TSPLIB, Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

Whitley, D., Starkweather, T., and Fuquay, D., Scheduling problems and travelling salesman: the genetic edge recombination operator, in: *Proceedings of the Third International Conference on Genetic Algorithms*, J.D. Schaffer (ed.) (Morgan Kaufmann, San Mateo, CA) pp. 133–140, 1989.

Xu, J-X., and Deng, X., Complex chemotaxis behaviors of C. elegans with speed regulation achieved by dynamic neural networks, *Proceedings of the IEEE World Congress on Computational Intelligence*, Brisbane, Australia, June 10 – 15, 2012.

Zhang, Y., Lu, H., and Bargmann, C., Pathogenic bacteria induce aversive olfactory learning in Caenorhabditis elegans, *Nature*, vol. 438, pp. 179 – 184, 2005.

Biography

Jean-Paul Arnaout is an Associate Professor of Management at the Business Administration Department at Gulf University for Science and Technology. He received his PhD and M.S. from the Department of Engineering Management and Systems engineering at Old Dominion University, Norfolk, Virginia in 2006 and 2003 respectively. He received his bachelor's degree in Mechanical Engineering from the University of Balamand, Lebanon. Dr. Arnaout developed several simulation and optimization models in several areas including but not limited to port operations, supply chain, agriculture, and healthcare. His Research interests include Optimization Techniques, Modeling and Simulation, and Scheduling and Rescheduling. He can be reached at arnaout.j@gust.edu.kw.