Vol.2012-MPS-87 No.1
2012/3/1

A Novel Particle Swarm Optimization based Algorithm for Path Optimization in Embedded Systems

Umair F. Siddiqi,[†1] Yoichi Shiraishi[†1]
and Sadiq M. Sait [†2]

This work presents a fast and memory efficient Particle Swarm Optimization (PSO) based algorithm for solving the multi-objective path optimization problem. The proposed algorithm uses innovative technique for particles' displacement which is based on exploring new sub-paths in the network in-order to improve the particles' positions. The proposed algorithm is implemented using C++ and executed on an ARM based embedded system. Its performance is compared with Non-dominated Sorting Algorithm-II (NSGA-II) and Simulated Annealing (SA). The results show that the proposed algorithm has found Pareto optimal solutions of quality equal to the NSGA-II and better than SA. The maximum number of paths which should be stored in the memory during optimization is about half of the NSGA-II. Therefore, it is suitable for implementation on embedded systems.

1. Introduction

Path Optimization (PO) is a critical operation in many applications. A large number of applications are executed on small size embedded systems which have limited amount of memory and computational speed. Navigation systems of intelligent vehicles is an example application which uses less powerful embedded systems to reduce power consumption. Intelligent vehicles include electric vehicles, hybrid vehicles and internal combustion engine based vehicles. The algorithms which require high memory and/or are highly computational are not suitable for the embedded systems. Therefore, the algorithms which can perform good quality path optimization and require lesser amount of memory and processor speed are desired.

The Path Optimization (PO) generally have two or more optimization objectives and therefore, also called as multi-objective shortest path (MOSP) problem[1]. The MOSP is an NP hard problem[1],[2]. The objectives in the multi-objective optimization problem can be in contradiction to each other and no single solution is said to be optimum. Therefore, the multi-objective optimization algorithms find a set of Pareto optimal solutions. The Evolutionary Computation (EC) algorithms have been predominately used to solve the multi-objective optimization problems. The population based algorithms work on a population of solutions and simultaneously find several solutions. This approach is suitable for multi-objective optimization problems which have several Pareto optimal solutions. The Genetic Algorithm (GA) and Particle Swarm Optimization (PSO)[6] are examples of population based algorithms.

This work presents a Particle Swarm Optimization (PSO) based algorithm for the PO problem. The proposed algorithm reduces the memory requirement to be equal to the size of its Swarm. The proposed algorithm has innovative methods for the velocity calculation and displacement of particles. The performance of the proposed algorithm is compared with Simulated Annealing (SA)[4] and Non-Dominated Sorting Algorithm -II (NSGA-II)[3]. SA works on only one solution and is therefore very memory efficient. NSGA-II is a popular algorithm for solving the multi-objective optimization problems and finding Pareto optimal solutions. The results show that the proposed algorithm has found Pareto optimal solutions of quality better than the SA and equal

†1 Department of Production Science & Technology, Gunma University, Gunma, Japan
†2 King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia

to NSGA-II. Its average computation time is at-most 1.42 times of the NSGA-II and 1.47 times of the SA to execute equal number of iterations. .

This paper is organized as follows: The second section describes the problem of multi-objective path optimization. Third section presents the proposed algorithm. Fourth section presents the simulation results and discussion. The last section contains the conclusion.

2. Problem Description

Let us consider an undirected graph, G = (V, E). The set V contains the vertices or nodes in the graph and set E contains the edges or arcs which join the nodes. Let us consider that the graph contains a total of N_v number of vertices and N_e number of edges. Any edge $e_i \in E$ is represented as $e_i = (n_x, n_y)$, where n_x is the starting node and n_y is the ending node of the edge e_i. e_i is associated with up-to K weights, i.e., $(w_1, w_2, w_3, ..., w_K)$. A path between a source node i.e., n_A and a destination node i.e., n_B $(n_A, n_B \in V)$ is represented as: $P = \{e_0, e_1, ..., e_{n-1}\}$, such that: $P \subset E$, $e_0 = (n_A, n_x)$, $e_{n-1} = (n_y, n_B)$, $n_x, n_y \in V$. Any multi-objective optimization problem can have up-to K objective functions which are represented as: $f_k(P) = \sum_{\forall e_x \in P} e_x.w_k$, for $k = 1$ to K. The goal of the optimization is represented as:
$Minimize(f_1(P), f_2(P), ..., f_K(P))$. The solution of any multi-objective optimization problem is a set of Pareto optimal solutions, i.e. $S_{PO} = \{P_{S1}, P_{S2}, ..\}$, such that any $P_{Si} \in S_{PO}$ is a complete path from the source node (n_A) to the destination node (n_B) and is not dominated by any other solution in the set S_{PO}. A solution dominates another solution if it is better than the other solution in at-least one objective function value and the remaining of its objective function values are equal to or better than the other algorithm.

3. Proposed Algorithm

The PSO algorithm was first proposed by Kennedy and Eberhart in 1995[6]. This section discusses the enhancements in the proposed algorithm in-order to perform memory efficient multi-objective path optimization.

3.1 Representation of Particles

In the proposed algorithm, any particle $P_j \in S$, for j= 0 to $M-1$ is a complete solution and is represented in an m-dimensional search space (R^m) as $P_j = \{p_0, p_1, ..., p_{m-1}\}$. The value of any element $p_i \in P_j$ can be obtained as follows:

$$p_i = \begin{cases} n_A \text{ if i=0} \\ n_x \text{ if } (p_{i-1}, n_x) \in E, n_x \in V \\ null \text{ if } p_{i-1} = \{n_B \text{ or } null\} \end{cases} \quad (1)$$

The first element in any particle P_j contains the source node (n_A), whereas the destination node (n_B) is the last not null element in it. Any two consecutive elements (p_i, p_{i+1}) represent an edge $(e_x \in E)$. The objective functions f_1 to f_K can be applied to individual particles, i.e. $Obj(P_j) = (f_1(P_j), f_2(P_j), ..., f_K(P_j))$. The function $f_k(P_j) = \sum_{e_x \in P_j}(e_x.w_x)$, (where k= 1 to K).

3.2 Initialization

The first step in PSO is the initialization of the particles in the Swarm. In this work, the Swarm is initialized to M particles and each particle is a distinct randomly generated path from the source (n_A) to the destination (n_B) node. Fig. 1 shows an algorithm to generate a random path between the two nodes $(n_A$ & $n_B)$.

3.3 Find *gbest* and *pbest* values

In every iteration the values of *pbest* and *gbest* solutions

```
Input: nodes: $n_A$, & $n_B$, G=(V,E), $N_e$= Number of elements in $E$
Output: Q: A path from $n_A$ to $n_B$ nodes.
1: $W= random(N_e)$
2: Q= Apply Dijkstra's Algorithm $(n_A, n_B)$
3: return Q
```

Fig. 1 Method to find a random path: $form\_path(n_A, n_B)$.

should be updated. The *pbest* value for any particle P_j consists of K components and is represented as: $pbest(P_j(t)) = \{l_{P_j1}, l_{P_j2}, ..., l_{P_jK}\}$. The values of the components can be determined as: $l_{P_jk}(t) = \arg\min_t(f_k(P_j(t)))$, where t is the iteration count, k= 1 to K and $j = 0$ to $M - 1$. The global best i.e., *gbest* has up-to K components and is represented as: $gbest = \{g_1, g_2, ..., g_K\}$. The value of any component $g_k \in gbest$ can be computed as: $g_k = \arg\min_j((l_{P_jk}))$, where j= 0 to $M - 1$ and k= 1 to K.

3.4 Selection of the Pareto Optimal Solutions

In every iteration, up-to $R_z \times M$ (where $R_z \in \{x \in \mathbb{R} | 0 \leq x \leq 1\}$ and M is the Swarm size) number of Pareto optimal particles in the Swarm are marked and the remaining particles are unmarked. The procedure is shown in Fig. 2.

3.5 Method for the Calculation of Velocities

The velocity of the particles should lie in the range $\{x \in \mathbb{Z} | 0 \leq x < V_{max}\}$, where $V_{max} \in \mathbb{Z}^+$ is the maximum velocity. The proposed algorithm stores the objective function values of the local best and global best particles. The difference between any two positions is computed as: Let us consider two positions A and B, the objective function values of A and B are represented as: $Obj(A) = \{a_1, a_2, .., a_K\}$ and $Obj(B) = \{b_1, b_2, .., b_K\}$. First the differences $a_i - b_i$, for i= 1 to K are computed. Then the square root of the sum of the squares of the differences i.e.

```
Input: $S = \{P_0, P_1, ..., P_{M-1}\}$, $R_z \in \{x \in \mathbb{R} | 0 \leq x \leq 1\}$
Output: $S$ in which the Pareto optimal solutions are marked
1: $CntPareto = 0$
2: for i=0 to $M - 1$ do
3:   $S.marked = false$
4: end for
5: for $i = 0$ to $M - 1$ do
6:   cnt=0;
7:   for $j = 0$ to $M - 1$ do
8:     if $P_j \succ P_i$ then
9:       $cnt++$:
10:    end if
11:  end for
12:  if $cnt == 0$ then
13:    $P_i.marked = true$
14:    $CntPareto++$
15:  end if
16: end for
17: while $CntPareto > R_z \times M$ do
18:   $U_{max} = \infty$, $I_U = 0$
19:   for $j = 0$ to $M - 1$ do
20:     $t = \sqrt{\sum_{i=1}^{k} f_i(P_j)^2}$
21:     if $U_{max} > t$ & $P_j.marked == true$ then
22:       $U_{max} = t$, $I_U = j$
23:     end if
24:   end for
25:   $P_{I_U}.marked = false$, $CntPareto--$
26: end while
27: return $S$
```

Fig. 2 Procedure to distinguish the Pareto optimal solutions in the Swarm.

```
Input: $V_{max}$, $c_1$, $c_2$, $r_1$, $r_2$, $w$, $v_{P_j}(t)$, $P_j$, $pbest(P_j) = \{l_{P_j1}, l_{P_j2}, ..., l_{P_jK}\}$, $gbest = \{g_1, g_2, .., g_K\}$
Output: $v_{P_j}(t+1)$
1: for k= 1 to K do
2:     $t_k = f_k(P_j) - l_{P_jk}$
3: end for
4: $L_1 = \sqrt{\sum_{i=1}^{K} t_i^2}$
5: for $k$= 1 to K do
6:     $t_k = f_k(P_j) - g_k$
7: end for
8: $L_2 = \sqrt{\sum_{i=1}^{K} t_i^2}$
9: $v_{P_j}(t+1) = wv_{P_j}(t) + c_1r_1L_1 + c_2r_2L_2$
10: $v_{P_j}(t+1) = v_{P_j}(t+1)\%V_{max}$
11: return $v_{P_j}(t+1)$
```

Fig. 3 Method to find the velocity of particles.

$\sqrt{\sum_{i=1}^{K}(a_i - b_i)^2}$ is computed, which is the difference between the particles A and B. The method proposed for the velocity determination is shown in Fig. 3. The symbol % indicates the modulus operation which is used to keep the velocity value within a range of $[0, v_{max} - 1]$.

3.6 Method for the Displacement of Particles

The velocities are added into the particles' current position to move them to their new positions. The procedure proposed to displace the particles is shown in Fig. 4.

3.7 Calculation of the Memory Requirement

The memory required by the proposed algorithm is primarily consists of the memory which is required to store the paths. Therefore, the memory requirements are determined in terms of the maximum number of solutions which should be stored in the memory at any time. Let us consider that a path requires Δ units of memory. The proposed algorithm stores paths equal to

```
Input: $P_j(t) = \{p_0, p_1, ..., p_{m-1}\}$, $v_{P_j}(t+1)$,
Output: $P_j(t+1)$
1: count =0
2: for $i = 0$ to $m-1$ do
3:     if $p_i$ is not null then
4:         count++;
5:     end if
6: end for
7: for $i$= 0 to $v_{P_j}(t+1) - 1$ do
8:     $r_n$: random number s.t. $r_n \in \{x \in \mathbb{Z} | 0 \le x \le count - 1\}$
9:     $t = form\_path(p_{r_n}, p_{count-1})$
10:    if $P_j.marked == true$ then
11:        if $t \succ P_j$ then
12:            $P_j(t+1) = t$
13:            $Exit$ from the $for$ loop
14:        end if
15:    else if $P_j.marked == false$ then
16:        $cmp = 0$
17:        for $k$=1 to $K$ do
18:            if $f_k(t) < f_k(P_j)$ then
19:                cmp++
20:            end if
21:        end for
22:        if $cmp > 0$ then
23:            $P_j(t+1) = t$
24:            $Exit$ from the $for$ loop
25:        end if
26:    end if
27: end for
28: return $P_j(t+1)$
```

,

Fig. 4 Proposed method to displace the particles.

the Swarm size. The method of particles' displacement also creates a path t. Therefore, it requires $(M+1)\Delta$ units of memory to store the paths. NSGA-II stores paths equal to the twice of the population size[3] and therefore, requires $2N\Delta$ units of memory. If we consider N is equal to M then the ratio between the memory required by the proposed algorithm to the memory required by the NSGA-II becomes $\frac{M+1}{2M}$. The memory required by the proposed algorithm to store the paths is nearly half of the memory required by the NSGA-II to store its paths. The SA maximally stores the current solution and a neighbouring solution and therefore requires 2Δ units of memory.

4. Simulation Results

In simulations, the value of K is considered as 3, and the multi-objective path optimization problem has three objective functions. The algorithms are developed on the PC and then compiled for the ARM9 embedded system by using the C++ cross-compiler for the ARM[10]. The undirected graphs are generated by using a random graph generation tool[7]. The graphs are labelled as SG_0, SG_1, SG_2, and SG_3 and their number of nodes and edges vary between [190, 270] and [670, 1250].The edges have up-to three weights and their values are assigned to random real numbers between $[0, 200]$. An instance of a problem consists of finding a set of Pareto optimal solutions between a source and destination nodes. The source and destination nodes are randomly selected in the road network. In the experiments, twenty problem instances are executed on each graph. The Hypervolumes of the Pareto optimal sets are calculated by using the tool[8] which is proposed by Carlos Fonseca et al. The Wilcoxon Rank Sum test[9] is used to test the null hypothesis, which shows that the hypervolume distributions of any two algorithms are equal. The rank sum tests are applied at significance level of 60% and it returns p and h values. If $h = 0$, which means that the two distribution are same and the two algorithms are equal in terms of the quality of their Pareto optimal solutions. p indicates the probability that an element from the first distribution is equal to the element from the second distribution. The algorithms were executed for 30 iterations. The proposed algorithm is implemented with parameter values as follows: The Swarm size (M) is set to 10. V_{max}, which is the maximum velocity of any particle is set to 3. R_z, which is the ratio between the number of Pareto optimal solutions to the Swarm size to set to 0.50. The SA implementation has the following parameters: $T=$ 100, $\alpha=$ 0.8, β = 0.85 and $M=$ 50. The stopping criteria in the SA is also 30 iterations. The NSGA-II implementation has a population size of 10. The results of the Wilicoxon rank-sum tests are shown in Table 1. The results show that the hypervolume distributions of the proposed algorithm are always equal to the NSGA-II distributions and always better than SA distributions. The average execution times of the algorithms to execute up-to 30 iterations are shown in Table 2. The results show that the average execution time of the proposed algorithm is 1.2 to 1.42 times of the average execution time of NSGA-II and 0.9 to 1.47 times of the average execution time of SA. Therefore, the execution time of the proposed algorithm also remains comparable to the other algorithms.

Table 1 Results of the Wilcoxon rank-sum tests

Graph	$H_0 : Proposed = NSGA-II$		$H_0 : Proposed = SA$	
	p	h	p	h
SG_0	0.67	0	0.19	1
SG_1	0.97	0	0.37	1
SG_2	0.95	0	0.48	1
SG_3	0.90	0	0.50	1
SG_4	1	0	0.33	1
SG_5	1	0	0.44	1

Table 2 Average execution times of the algorithms on different graphs.

Graph	Algorithm	Average execution Time (sec)	$\frac{Proposed}{NSGA-II\ or\ SA}$
SG_0	Proposed	32.48	-
	NSGA-II	22.83	1.42
	SA	33.11	0.98
SG_1	Proposed	42.6570	-
	NSGA-II	31.0435	1.2
	SA	46.7870	1.17
SG_2	Proposed	19.0510	-
	NSGA-II	13.5290	1.4
	SA	19.9740	0.17
SG_3	Proposed	27.4630	-
	NSGA-II	20.0700	1.42
	SA	31.7045	0.97
SG_4	Proposed	20.4545	-
	NSGA-II	13.0190	1.38
	SA	20.1685	0.9
SG_5	Proposed	47.4808	-
	NSGA-II	30.3410	1.22
	SA	38.4595	1.47

5. Conclusion

We have proposed a memory efficient Particle Swarm Optimization (PSO) based algorithm for solving the multi-objective path optimization problem. The proposed algorithm reduces memory requirements by storing the objective function values of the global best and local best positions instead of the particles. Each position is represented by a complete path. The particles change their new positions by modifying the sub-paths in their current positions. The proposed algorithm is executed on an ARM based embedded system. The proposed algorithm is compared with NSGA-II and SA. The comparison results show that the proposed algorithm has found Pareto optimal solutions of quality very close to the NSGA-II and better than SA. The average execution time of the proposed algorithm is also remains within 1.42 times of NSGA-II and 1.47 times of SA. Based on the experimental results, we can conclude that the proposed algorithm is suitable to perform multi-objective path optimization in embedded systems which generally have less powerful processor and limited memory. In the future, the proposed algorithm will be applied to perform path optimization in the navigation systems of electric vehicles.

References

1) Zbigniew TARAPATA, Selected Multicriteria Shortest Path Problems: AN Analysis of Complexity, Models and Adoption of Standard Algorithms, Int. J. Appl. Math. Comput. Sci., Vol. 17, No. 2, (2007) 269-287.
2) M.R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Co., 1997
3) Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan, A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II, IEEE Trans. Evolutionary Computation, vol. 6, No. 2, (2002) 182- 197.
4) S.M. Sait & H. Youssef, Iterative Computer Algorithms with Applications in Engineering, IEEE Computer Society Press, 1999.
5) Johannes M. Bader, Hypervolume-Based Search for Multiobjective Optimization: Theory and Methods, Ph.D. dissertation, Swiss Federal Inst. Technology (ETH) Zurich, Switzerland, 2009.
6) J. Kennedy, R. Eberhart, Particle Swarm Optimization, Proceedings of 1995 IEEE International Conference on Neural Network, pp. 1942-1948 (1995).
7) Fabien Viger, Matthieu Latapy, Efficient and simple generation of random simple connected graphs with prescribed degree sequence, 11th Conference of Computing and Combinatoric (COCOON 2005), (2005) 440-449.
8) Carlos M. Fonseca, Lus Paquete, and Manuel Lpez-Ibez, An improved dimension - sweep algorithm for the hypervolume indicator, 2006 IEEE Congress on Evolutionary Computation (CEC'06), (2006) 1157-1163.
9) Hollander, M., and D. A. Wolfe, Nonparametric Statistical Methods, John Wiley & Sons, Inc., 1999.
10) http://www.embeddedarm.com