An Empirical Evaluation of Martins' Algorithm for the Multi-Objective Shortest Path Problem

Gerrit K. Janssens
Hasselt University, Diepenbeek, Belgium
gerrit.janssens@uhasselt.be
and
Jose Maria Pangilinan
Saint Louis University, Baguio City, Philippines
joey.pangilinan@slu.edu.ph

KEYWORDS

Multi-objective optimization, Martins' algorithm, shortest path problem

ABSTRACT

The Shortest Path Problem is a popular optimization problem in operations research due to its wide range of practical applications. In most cases a single objective is considered, while also the multi-objective case has useful applications. The algorithms by Martins is considered very efficient. This study evaluates this algorithm by comparing it to a brute force algorithm as a first step to develop evolutionary algorithms for the multi-objective case. Experiments confirm the strength of the Martins' algorithm.

1. INTRODUCTION

The Multi-Objective Shortest Path Problem (MSPP) is an extension of the Shortest Path Problem that aims to find efficient (non-dominated or Pareto-optimal) paths from a source vertex to a target vertex with multiple objectives in a single execution. MSPP have applications in many different industries such as telecommunications, traffic routing, resource allocation and rapid response for different purposes like cost cutting and time management. Most studies on the evaluation of the performance of Martins' algorithm for the MSPP are theoretical and mathematical in nature. A few empirical studies show the algorithm's performance in terms of the number of solutions it generates on different test networks. However, recent empirical studies lack necessary comparative illustrations to show its performance against another search algorithm.

This paper presents a comparative performance evaluation of Martins' algorithm against a brute-force algorithm in terms of the number of efficient solutions generated and execution time. The paper first describes a brief background on the MSPP, then gives a summary of several types of algorithms that compute efficient paths for the multi-objective shortest path problem. Based on these algorithms, the performance of Martins algorithm is evaluated against a brute-force algorithm using several test networks. Finally, the results of the experiments are interpreted in terms of solution sets and execution time.

2. BACKGROUND

Multi-objective Shortest Path Problem

Given a directed graph $G = (V, E)$, where V is the set of vertices (nodes) and E the set of edges (arcs) with cardinality $|V| = n$ and $|E| = m$ and a d-dimensional function vector $\boldsymbol{c}:E\rightarrow [\Re^{+}]^{d}$. Each e belonging to E is associated with a cost vector $\boldsymbol{c}(e)$. A source vertex s and a target vertex t are identified. A path p is a sequence of vertices and arcs from s to t. The cost vector $\boldsymbol{C}(p)$ for linear functions of path p is the sum of the cost vectors of its edges, that is $\boldsymbol{C}(p) = \Sigma_{e\in p}\, \boldsymbol{c}(e)$ while $\boldsymbol{C}(p) = \min_{e\in p}\, \boldsymbol{c}(e)$ for maxmin functions. Given two vertices s and t, let $P(s, t)$ denote the set of all s-t paths in G. If all objectives are to be minimized, a path $p\in P(s, t)$ dominates a path $q\in P(s, t)$ iff $C_i(p) \leq C_i(q)$, $i = 1,\ldots,d$ and we write $p \preceq q$. A path p is Pareto-optimal if it is not dominated by any other path and the set of non-dominated solutions (paths) is called the Pareto-optimal set. The objective of the MSPP is to compute the set of non-dominated solutions that is the Pareto-optimal set **P** of $P(s,t)$ with respect to $\boldsymbol{c}$.

The problem of the single-source s, single-target t multi-objective shortest path is to find the set of all paths from s to t in G.

Algorithms for the MSPP

A variety of algorithms and methods such as dynamic programming, label selecting, label correcting, interactive methods, and approximation algorithms to name a few have been implemented and investigated with respect to the MSPP (Ehrgott and Gandibleux, 2000). The problem is known to be NP-complete (Garey and Johnson, 1979). It has been shown that a set of problems exist wherein the number of Pareto-optimal solutions is exponential which implies that any deterministic algorithm that attempts to solve it is also exponential in terms of runtime complexity at least in the worst-case. But some labeling algorithm studies (Gandibleux et al., 2006; Müller-Hannemann and Weihe, 2001) dispute this exponential behavior. They show that the number of efficient paths is not exponential in practice. Other authors avoid the complexity problem by developing new methods that run in polynomial time. For instance, Hansen and Warburton (Müller-Hannemann, 2001) separately developed fully polynomial time approximation schemes (FPTAS) for finding paths that are approximately Pareto-optimal. Interactive procedures (Coutinho-Rodriguez, 1999; Granat, 2003) similarly avoid the problem

of generating the whole set of efficient paths by providing a user-interface that assists the decision-maker to focus only on promising paths and identify better solutions according to preference.

Martins' algorithm (Martins, 1984) is a label setting algorithm that assigns for every vertex in its path permanent labels and temporary labels. The algorithm selects the minimum lexicographic label from all the sets of temporary labels and converts it to a permanent label, and propagates the information contained in this label to all the temporary labels of its successors. The process stops when there are no more temporary labels. Each permanent label corresponds to a unique efficient path. Martins algorithm ensures the computation of the maximal complete set of efficient paths from one vertex to all the other vertices of a network.

Gandibleux et al. (2006) extended the capability of Martins algorithm by modifying its dominance test to ensure the computation of the maximal complete set of efficient paths for min-max problems associated with the multi-objective networks.

Pangilinan and Janssens (2007) explored a Multi-Objective Evolutionary Algorithm as applied to the MSPP and described its behavior in terms of variety of solutions, computational complexity, and optimality of solutions. Results showed that the evolutionary algorithm is capable of finding diverse solutions to the MSPP in polynomial time.

Granat and Guerriero (2003) proposed an interactive procedure for the MSPP based on a reference point labeling algorithm. Their algorithm converts the multi-objective problem into a parametric single-objective problem whereby the efficient paths are found. The algorithm was tested on different grid and random networks and performance was measured based on execution time. They conclude that an interactive method, from their experimental results, is encouraging and does not require the generation of the whole Pareto-optimal set (which avoids the intractability problem). Likewise, Coutinho-Rodrigues et al. (1999) proposed an interactive method that incorporates an efficient *k*-shortest path algorithm in identifying Pareto-optimal paths in a bi-objective shortest path problem. The algorithm was tested against other MSPP algorithms on 39 network instances. They conclude that their *k*-shortest path algorithm performs better than other MSPP algorithms in terms of execution time.

Tsaggouris and Zaroliagis (2006) provided a Fully Polynomial-Time Approximation Scheme (FPTAS) for the determination of an approximate Pareto curve for multi-objective shortest paths that significantly improves especially in the case of more than two objective functions. The study shows that it can be used to provide better approximate solutions to multi-objective constrained efficient paths, multi-objective constrained paths, and non-additive shortest paths.

Paixao and Santos (2007) proposed a ranking algorithm that solves MSPP. This ranking algorithm finds all non-dominated path solution between a source node to destination node based on ranking path procedure by applying a stop ranking condition which allows to determine the entire set of non-dominated paths at the very early stage of the ranking procedure.

Pinto and Pascoal (2010) developed a labeling algorithm that computes multi-objective shortest paths by restricting the set of arcs according to the bottleneck values in order to find the minimal complete set of Pareto-optimal solutions.

3. EXPERIMENTS

The experiments intend to compare the performance between Martins' algorithm and a brute-force search algorithm. The algorithms are evaluated in terms of the number of efficient solutions and computation time for a single source-target multi-objective shortest path problem. The single source-target problem is different from the single source MSPP as it requires the computation of all efficient paths from a single source to a single target only and not from a single source to all other vertices.

Program Implementation

The implementation of Martins algorithm is based on Gandibleux et al. (2006). The algorithms are implemented using the C++ programming language, a desktop computer equipped with an Intel Core 2 Duo 2.56 GHz processor and two Gigabytes of RAM on a 32-bit operating system.

ALGORITHM 1: MARTINS' ALGORITHM

Requires: $G=(V, A)$ and C, the cost matrix for all arcs $(i,j) \in A$

Ensures: *All efficient paths from s to all vertices* $i \in V\backslash\{s\}$

l_i : is the label of vertex i
lt_i : is the entire list of temporary labels of vertex i
lp_i : is the entire list of permanent labels of vertex i
$z^p{}_{q,h}$: is the p^{th} performance of a permanent label of vertex q in position h
Δ : is the dominance relation (if $z \Delta z'$ then z is dominated by z')
perf() : performance operator

Initialization
$lt_i, lp_i \leftarrow \emptyset, \forall i \in V$
$lt_s \leftarrow \{[0,...,0,\perp,\perp]\}$ (the latter two have no meaning in the start vertex)

Iteration
while $(\bigcup_{i\in V} lt_i \neq \emptyset)$ **do**
  Find the minimum lexicographic label in $lt_i, \forall i \in V$
  $l_q \leftarrow \min \text{lex}\{\bigcup_{i\in V} lt_i\}$
  Move the selected label from the 'temporary' to the 'permanent' list $lt_q \leftarrow lt_q \backslash \{l_q\}$; $lp_q \leftarrow lp_q \cup \{l_q\}$
  Store the position of label l_q from list of vertex j
  $h_q \leftarrow \text{location}(lp_q)$
  Label all successors of q
  for all $j \in V \mid (q,j) \in A$ **do**
    Compute l_j, the current label of vertex of j
    $l_j \leftarrow [z^1_{q,h} + c^1(q,j), ..., z^k_{q,h} + c^K(q,h), q, h]$
    Verify that there is no performance of vertex j labels dominating perf(l_j)
    if $(\nexists\, l'_j \in \{lt_j \cup lp_j\} \mid \text{perf}(l_j) \Delta \text{perf}(l'_j))$ **then**
      Store the label l_j of vertex j as 'temporary'
      $lt_j \leftarrow lt_j \cup \{l_j\}$
      Delete all temporary labels of vertex j dominated by l_j
      $lt_j \leftarrow lt_j \backslash \{l'_j \in lt_j \mid \text{perf}(l'_j) \Delta \text{perf}(l_j)\}$
    end if
  end for
end while

The brute-force search algorithm used in the study is implemented as a depth-first search algorithm. The depth-first search starts at the root vertex and explores all possible successors before backtracking. The brute-force search algorithm lists all possible paths that lead to the target vertex. After the listing process, all dominated solutions are eliminated by dominance tests.

ALGORITHM 2: BRUTE-FORCE SEARCH ALGORITHM

BruteForceSearchAlgorithm(G,v) (v is the source vertex)
Stack S : ⟵ ∅; (start with an empty stack)
for each vertex u, set visited[u] := false;
    push S, v;
    while (S is not empty) **do**
        u ⟵ pop S;
        if (not visited[u]) **then**
            visited[u] := true;
            for each unvisited neighbor w of u
                push S, w;
        end if
    end while
end for

for path *paths p in S*
    if (perf(l_j) ∆ perf(l'_j)) **then**
        if perf(l_j) is dominated by perf(l'_j) **then**
            Store the label l_j as an optimal solution
            lt_j ⟵ lt_j ∪ {l_j}
            Delete all temporary labels of vertex j dominated by l_j
            lt_j ⟵ lt_j \ {l'_j ∈ lt_j | perf(l'_j) ∆ perf(l_j)}
        else
    else if perf(l_j) and perf(l'_j) is non-dominated **then**
            Store the label l_j
            lt_j ⟵ lt_j ∪ {l_j}
    end if
end for
END BruteForceSearchAlgorithm()

Test Networks

Forty-five network configurations were randomly chosen from Gandibleux et al. (2006). The configuration is as follows: 15 networks with 50 vertices (5 networks of 5% density, 5 networks of 10% density and 5 networks of 20% density); 15 networks with 100 vertices (5 networks of 5% density, 5 networks of 10% density and 5 networks of 20% density); and 15 networks 200 vertices (5 networks of 5% density, 5 networks of 10% density and 5 networks of 20% density). Each edge of network has two cost values in the range [1, 1000].

For each network configuration, the maximal complete set of the shortest paths from vertex = 1 to a randomly selected vertex was computed. The objective function is a minimize a (2-Sum) problem, i.e. min $\Sigma_{e \in p}$ $\boldsymbol{c}(e)$. The algorithms were run and given a 24 hours duration limit to complete. All network traversals that are still in-process after 24 hours were marked as an incomplete execution.

4. RESULTS AND FINDINGS

Figure 1 and Figure 2 show the efficient paths found for a single source-target problem on five different 50-vertex networks using Martins' algorithm and the brute-force search algorithm respectively. Martins' algorithm was able to compute the maximal complete set of efficient solutions to the 50-vertex networks of different densities whereas the brute-force search algorithm can only compute the maximal complete set of efficient solutions to the 50-vertex networks with 5% density.

Figure 1 shows that the number of efficient solutions generally increases as the density of the test network increases. This is not the case for Network 1 wherein the number of solutions for the 20% test network is lower than the 5% and 10% networks.

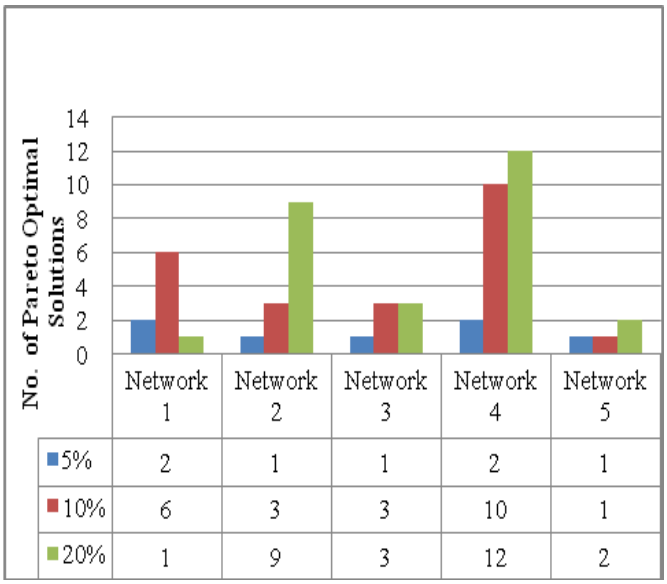


Figure 1 Efficient Solutions using Martins' Algorithm for 50-Vertex Networks

Figure 2 shows the cardinality of the solutions sets of the test networks and it is evident that the brute-force algorithm failed to compute efficient paths when the densities of the networks are 10% and 20%. This means that there were no solutions found in a span of 24 hours.

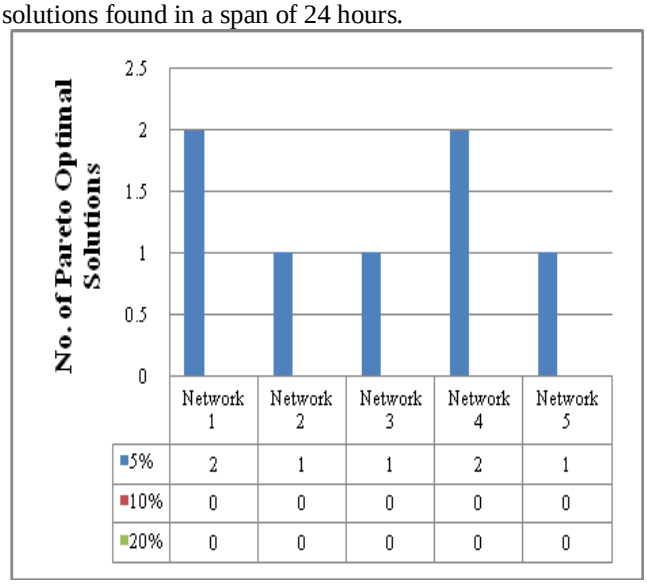


Figure 2. Efficient Solutions of the Brute-force algorithm for 50-Vertex networks

Looking at Figure 1 and 2, the number of efficient solutions and the efficient solutions to the 50-vertex test networks with 5% density found by Martins' algorithm and by the brute-force algorithm are the same. Since the brute-force algorithm in this case computed for the complete set of efficient solutions, means that Martins' algorithm also computed the complete set of efficient solutions for 50-vertex, 5% density test networks.

Figures 3 and 4 show solution sets to the 100-vertex and 200-vertex test networks obtained by Martins' algorithm. No solutions were generated by the brute-force algorithm for these types of networks. Figure 3 presents the cardinality of the solution sets of efficient paths for the 100-vertex networks with 5%, 10%, 20% densities. It is observed that the number of efficient solutions increases as the density of each network increases.

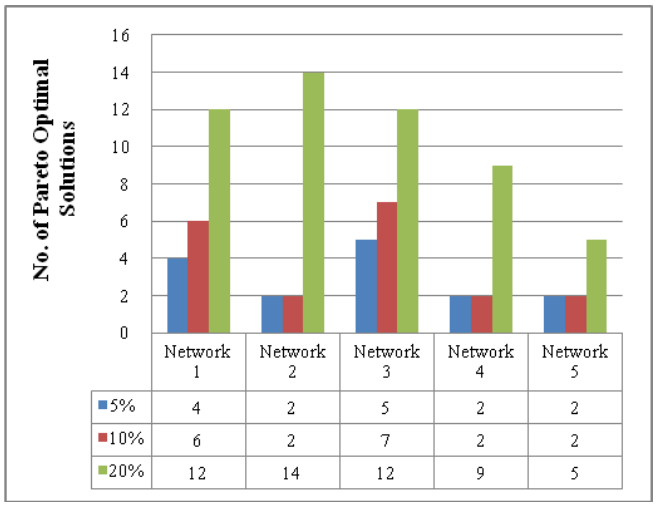


Figure 3 Efficient Solutions using Martins Algorithm for 100-Vertex networks

Figure 4 shows the cardinality of the solution sets of efficient paths for 200-vertex networks with 5%, 10%, 20% densities. Again, it is observed that the number of efficient solutions increases as the density of each network increases.

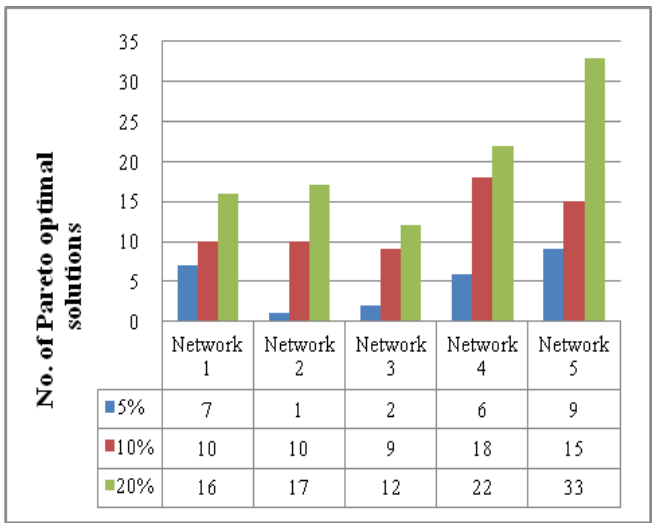


Figure 4 Efficient Solutions using Martins' Algorithm for 200-Vertex networks

Figure 5 and Figure 6 show the execution times of Martins algorithm and the brute-force algorithm for the 50-vertex sample networks respectively. Martins' algorithm computes all efficient paths in less than 4 seconds for all density configurations and execution time ranges from 0.02 to 3.31 seconds whereas the brute-force algorithm computes for efficient paths in the range of 16 to 5,823 seconds. With respect to Network 5 at 5% density, Martins algorithm computes for the solution set in 0.02 seconds while the brute-force algorithm requires 1.6 hours. The variability in the execution time of Martins algorithm is small while the variability in the brute-force algorithm is very large.

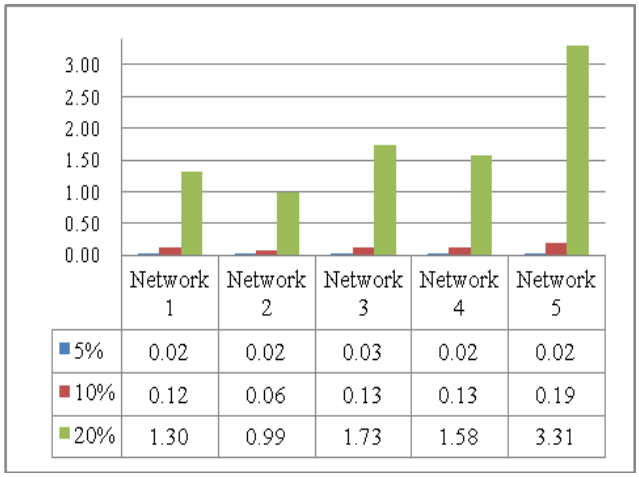


Figure 5 Execution Time in Seconds using Martins' Algorithm for 50-Vertex Networks

The execution times of the brute-force algorithm for the 10% and 20% density configurations cannot be shown since no solutions were found within a 24-hour execution time span. The zeroes shown in Figure 6 are equivalent to undetermined execution times.

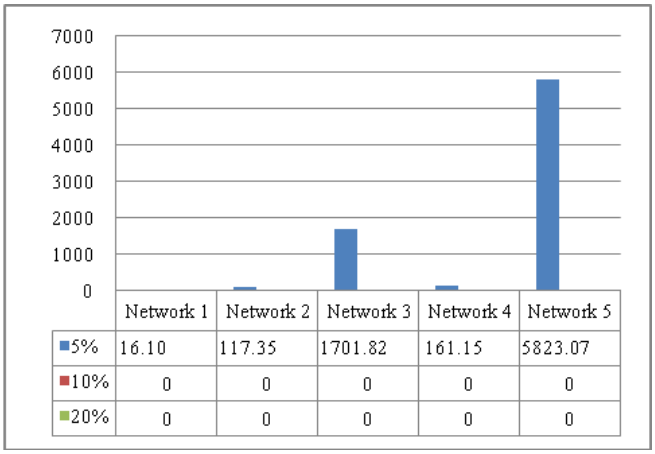
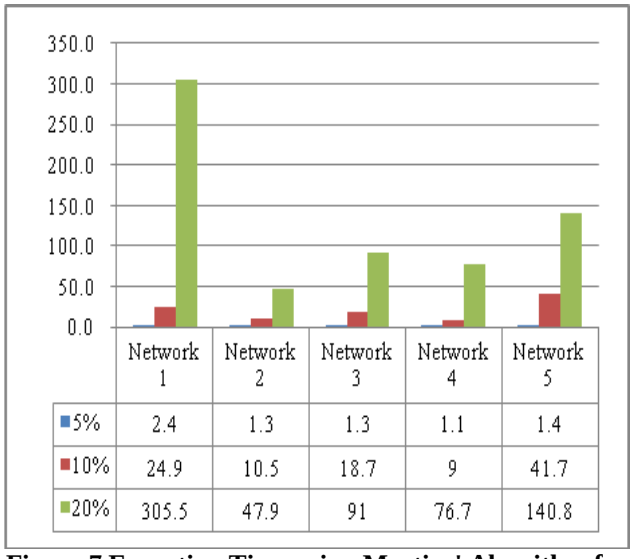


Figure 6 Execution Time of Brute-force algorithm for 50-Vertex Networks

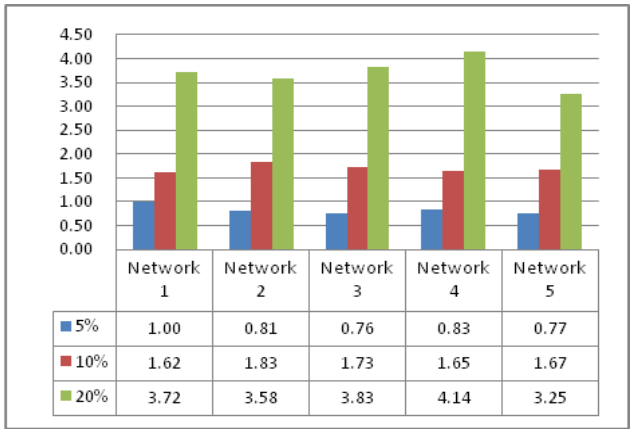
Figure 7 and Figure 8 show the execution times of Martins' algorithm for the 100-vertex and 200-vertex test networks respectively. It can be observed from Figure 7 and Figure 8 that the execution times increase as the network density increases. With respect to the 100-vertex test networks and considering all density configurations, Martins' algorithm computes all efficient paths in 2.4 to 140.8 seconds. With respect to the 200-vertex test networks and all density configurations, Martins' algorithm computes all efficient paths in 1.0 to 3.5 seconds. It is evident that the execution times of the 100-vertex test networks are higher than the execution times of the 200-vertex test networks. This may be due to the test problem i.e. efficient paths are computed

for only one pair of vertices instead of one source vertex to all other vertices in the network .



	Network 1	Network 2	Network 3	Network 4	Network 5
5%	2.4	1.3	1.3	1.1	1.4
10%	24.9	10.5	18.7	9	41.7
20%	305.5	47.9	91	76.7	140.8

Figure 7 Execution Time using Martins' Algorithm for 100-Vertex networks



	Network 1	Network 2	Network 3	Network 4	Network 5
5%	1.00	0.81	0.76	0.83	0.77
10%	1.62	1.83	1.73	1.65	1.67
20%	3.72	3.58	3.83	4.14	3.25

Figure 8 Execution Time of Martins' Algorithm for 200-Vertex networks

CONCLUSIONS

This study presents an empirical evaluation of Martins' algorithm against a brute-force search algorithm for the single source-target multi-objective shortest path problem. The efficient solutions and execution times are the primary concern of the evaluation process. Several network configurations with varying sizes and densities were used to compare the performance of Martins' algorithm and the brute-force search algorithm.

The results of the experiments show that Martins algorithm generates a complete set of efficient solutions to all network configurations whereas the brute-force algorithm only generates solutions for small-sized and low-density networks. Hence comparison of solutions between algorithms is defined by the limitations of the brute-force algorithm. With respect to Martins' algorithm, the size of the solution set increases as size and density increases. In terms of execution times Martins algorithm is fast but its execution time is dependent on the density of the network i.e. it requires longer execution times for higher density networks.

Acknowledgment The authors would like to thank the group of Kevin Sumagit, CS students in 2010 at Saint Louis University, in Baguio, The Philippines for coding the algorithms and running the experiments.

REFERENCES

Coutinho-Rodrigues, J., Climaco, J., and Current, J. 1999. "An interactive bi-objective shortest path approach: searching for unsupported nondominated solutions", *Computers and Operations Research* Vol. 26(8), pages 789-798.

Ehrgott, M. and Gandibleux, X. 2000. "A survey and annotated bibliography of multi-objective combinatorial optimization", *OR Spektrum* Vol. 22(4), pages 425-460.

Gandibleux, X., Beugnies, F., and Randriamasy, S. 2006. "Martins' algorithm revisited for multi-objective shortest path problems with a MaxMin cost function", *4OR: A Quarterly Journal of Operations Research* Vol. 4(1), pages 47-59.

Garey, M.R. and Johnson D.S. 1979. *Computers and intractability: A guide to the theory of NP-completeness*, San Francisco, CA: Freeman.

Granat, J., and Guerriero, F. 2003. "The interactive analysis of the multi-criteria shortest path problem by the reference point method", *European Journal of Operational Research* Vol. 151, pages 103-111.

Martins, E.Q.V. 1984. On a multicriteria shortest path problem. *European Journal of Operational Research* Vol.16, pages 236-245.

Müller-Hannemann, M. and Weihe, K. 2001. "Pareto Shortest Paths is often feasible in practice", In: Brodal, G., Frigioni D., Marchetti-Spaccamela A. (eds.)*: Proc. 5th International Workshop on Algorithm Engineering: WAE 2001. Lecture Notes in Computer Science*, Vol. 2141, pages 185-198.

Pangilinan, J.M., and Janssens, G.K. 2007. "Evolutionary Algorithms for the Multiobjective Shortest Path Problem", *World Academy of Science, Engineering and Technology* Vol. 25, pages 205-210.

Paixao, J.M. and Santos, J.L. 2007. Labelling methods for the general case of the multi-objective shortest path problem – a computational study. Pre-publications of the Department of Mathematics 07-42, Universidade de Coimbra.

Pinto, L. and Pascoal, M. 2010. "On algorithms for the tri-criteria shortest path problem with two bottleneck objective functions", *Computers and Operations Research*, Vol. 37(1), pages 1774-1779.

Tsaggouris, G. and Zaroliagis C. 2005. "Multiobjective optimization: Improved FPTAS for Shortest Paths and Non-linear objectives with Applications", In: *Algorithms and Computation* – ISAAC2006, *Lecture Notes in Computer Science* Vol.4288, Springer-Verlag, pages 389-398.