Parallel Algorithms for Multicriteria Shortest Path Problems

David L. Sonnier
Lyon College, dsonnier@lyon.edu

Follow this and additional works at: http://scholarworks.uark.edu/jaas
Part of the Numerical Analysis and Computation Commons

Recommended Citation
Available at: http://scholarworks.uark.edu/jaas/vol60/iss1/20
Parallel Algorithms for Multicriteria Shortest Path Problems

DAVID L. SONNIER

Lyon College, P.O. Box 2317, Batesville, AR 72503

Correspondence: dsonnier@lyon.edu

Abstract.—This paper presents two strategies for solving multicriteria shortest path problems with more than two criteria. Given an undirected graph with n vertices, m edges, and a set of K weights associated with each edge, we define a path as a sequence of edges from vertex s to vertex t. We want to find the Pareto-optimal set of paths from s to t. The solutions proposed herein are based on cluster computing using the Message-Passing Interface (MPI) extensions to the C programming language. We solve problems with 3 and 4 criteria, using up to 8 processors in parallel and using solutions based on two strategies. The first strategy obtains an approximation of the Pareto-optimal set by solving for supported solutions in bi-criteria sub-problems using a weighted-sum approach, then merging the solutions. The second strategy applies the weighted-sum algorithm directly to the tri-criteria and quad-criteria problems to find the Pareto-optimal set of supported solutions, with each processor using a range of weights.

Key words.—Multicriteria shortest path problems, Message-Passing Interface (MPI), C programming language, bi-criteria sub-problems, weighted-sum algorithm, tri-criteria and quad-criteria problems.

Introduction

The “Shortest Path Problem” has been studied extensively in recent years and numerous algorithmic solutions are available. In its unicriterion version, Dijkstra’s algorithm provides a ready and efficient solution; in the bi-criteria case it has been studied extensively as well, and numerous algorithms have been proposed and tested. Documented research dealing with problem instances involving more than two criteria is also available but not to the same extent as in the bi-criteria case, and research dealing with parallel algorithmic solutions is rare. This paper presents two algorithms for solving multicriteria shortest path problems with more than two criteria. The solutions proposed herein are based on cluster computing using the Message-Passing Interface (MPI) extensions to the C programming language. The proposed algorithms were tested using graphs of various sizes and up to 4 criteria, and performance results are shown.

Multicriteria Shortest Path Problem and Background

Given G = (V,E) is an undirected graph with a set of |V| = n vertices, and |E| = m is a set of edges, and a set of K weights is associated with each edge. We denote by $c_{ij}^k$ the cost of edge $(i,j)$ due to weight function $k$. A path is defined as a sequence of $n$ arcs, $(s, i_1), (i_1, i_2), \ldots (i_{n-1}, t)$ from vertex $s$ to vertex $t$. We want to find the Pareto-optimal set of paths from node $s$ to node $t$ of $G$. If $K = 2$, the problem is the bi-criteria shortest path problem, which is defined as follows:

$$\min f_i(x) = \sum_{(i,j) \in E} c_{ij}^i$$
$$f_2(x) = \sum_{(i,j) \in E} c_{ij}^j$$

s.t.

$$\sum_{(i,j) \in E} x_j - \sum_{(i,j) \in E} x_i = \begin{cases} 1 & \text{if } i = s \\ 0 & \text{if } i \neq s, t \\ -1 & \text{if } i = t \end{cases}$$

$x_j$ binary

The nondominated set of solutions is defined as the set having the following properties. Assume we have a vector of $K \geq 2$ objective functions:

$$f(x) = [f_1(x), f_2(x), \ldots, f_K(x)]$$

(2)

to be minimized over a set of criteria. The decision variable is

$$x = [x_1, x_2, \ldots, x_n]$$

(3)

where the entire set of solutions is $X$. We say that $x^*$ is nondominated or Pareto-optimal if there is no $x \in X$ such that

$$f_i(x) \leq f_i(x^*), \quad i = 1..K,$$

and $f_j(x) < f_j(x^*)$ for some $j \in \{1..K\}$.

(4)

For the bi-criteria problem, a variety of solution techniques based on minimized weighted-sum methods have been presented, such as Mote et al. (1991), Henig (1985), Coutinho-Rodrigues, et al. (1999), and Ehrgott (2000). We define weight $\lambda$ such
that $0 \leq \lambda \leq 1$ and we can reduce the bi-criteria problem to a unicriterior problem by using the following objective function:

$$\lambda f_1 + (1-\lambda) f_2$$

(5)

Finding all supported Pareto-optimal solutions requires solving multiple iterations of the problem using this objective function along with various weights. Various techniques for selecting the weights have been presented and tested. One approach is the parametric analysis. Using this approach, we generate a sequence of weights, $\{\lambda_i\}$, starting with $\lambda_i = 1$, with each weight resulting in a distinct supported Pareto-optimal solution. (Henig 1985). We begin with the lexicographic shortest path associated with $\lambda_i = 1$, then let

$$\alpha = \min -\frac{(c_y^f + f_1(p_j) - f_1(p_i))}{(c_y^f + f_2(p_j) - f_2(p_i))}$$

(6)

where $f_i(p)$ is the cost of path $j$ for criteria $k$, and the minimum is taken over all arcs in the graph such that the denominator is negative. This ratio is the depreciation in the first objective to the improvement in the second. Based on this value of $\alpha$, we compute the next iteration based on the value $\lambda_i = 1/(1+\alpha)$. We continue in this manner until there are no negative results in the denominator, at which point the process terminates, having resulted in $q$ Pareto-optimal solutions. The complexity of calculating $\alpha$ is $O(mn^3)$.

Another method, also described in Henig (1985), is to generate the sequence starting with $\lambda_i = 0$ and $\lambda_q = 1$, solve the two lexicographic problems based on these values, resulting in paths $p_i$ and $p_q$. Using the ratio

$$\alpha = \frac{f_1(p_q) - f_1(p_i)}{f_2(p_q) - f_2(p_i)},$$

(7)

we set $\lambda_i = 1/(1+\alpha)$ and $\lambda_q = 1 - \lambda_i$. As long as $\lambda_i f_1 + \lambda_q f_2$ results in new solutions, we continue the search of the nondominated front between the recently discovered solutions.

A third, more naive method, but one which works well in parallel computing, is to generate a sequence of $\{\lambda_i\}$, starting with $\lambda_i = 0$ and incrementing $\lambda$ by some small $\alpha$ until $\lambda_i = 1$. Each value in the sequence is applied to (5). This method is easily expandable to values of $K > 2$ by employing the following objective function:

$$\lambda_1 f_1 + \lambda_2 f_2 + \ldots + \lambda_K f_K$$

(8)

where the weights are input from file or generated automatically as to give the desired distribution.

The problem with the first two methods described above is twofold. First, they are not easily expanded to higher values of $K$. Another problem, and one shared by all of the above approaches, is that they find only the supported Pareto-optimal solutions, defined as those solutions which lie on the convex hull of the feasible region. It has been shown in Ehrgott (2000) that the number of solutions to an MCSP may be exponential, but computational experience shows that this is not always the case. Existing solutions may be neglected due to the fact that they are not on the convex hull. The existence of such solutions can be seen in the following example. Assume that there are only Pareto-optimal supported solutions. We construct an instance of the Bi-criteria Shortest Path Problem (BSPP) in which we seek the Pareto-optimal set of paths from vertex 1 to vertex 5. (See Fig. 1) The solutions, all Pareto-optimal, are shown in Table 1. We construct another instance of the problem by inserting edge (1, 5), with a cost of (3, 6, 8). Now, in addition to the previous solutions, Path 1-5, which we will denote by $p_5$, with cost vector (3, 6, 8), is a Pareto-optimal solution. Note that no value of $\alpha$ exists such that $p_5$ is the shortest path using the weighted objective function $(1-\lambda) f_1 + \lambda f_2$. Such a solution, often referred to as an “unsupported nondominated solution,” cannot be found using any weighted-sum method. (See Fig. 2). However, unsupported solutions can be found by pairing a weighted-sum method with a second method designed to search for unsupported solutions, as described in Coutinho-Rodrigues (1999), for example. A complete solution set, including both supported and unsupported solutions, can be found by using labeling algorithms, as in Martins (1984), Mote et al. (1991), and Brumbaugh-Smith and Shier (1989). Procedures for finding solutions for problems with $K > 2$ have been presented in Martins (1984), Corley and Moon (1985), and Ehrgott (2000); a summary of research in this area is found in Ehrgott and Gandibleux (2002). An approximation of a solution set is defined as a solution set obtained by using a heuristic algorithm. It is not guaranteed to be complete, but it provides a reasonable nondominated set of solutions from which to choose.

Table 1. Pareto-optimal Paths From Vertex 1 to Vertex 5 in Fig. 1.

<table>
<thead>
<tr>
<th>Path</th>
<th>Cost Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$; 1-2-3-4-5</td>
<td>(5,5)</td>
</tr>
<tr>
<td>$p_2$; 1-3-4-5</td>
<td>(4,6)</td>
</tr>
<tr>
<td>$p_3$; 1-2-3-5</td>
<td>(3,7)</td>
</tr>
<tr>
<td>$p_4$; 1-3-5</td>
<td>(2,8)</td>
</tr>
</tbody>
</table>

Fig. 1. An Undirected Graph
to the multicriteria shortest path problem can be exponential, approximation methods are typically used.

This paper extends previous research by presenting a parallel algorithm for finding an approximation set for problems with \( K = 3 \) or \( K = 4 \) based on solving sub-problems involving only two criteria. We compare the performance of this algorithm against a straightforward, parallel, weighted-sum implementation. The multiplicative version of the problem is the same as that presented in (1), but with the following objective function:

\[
\min \sum_{(x,y) \in E} c_{x}^{k} \quad \text{for} \quad k = 1, \ldots, K \tag{9}
\]

Archival of solutions, which is of special importance in cluster computing, is discussed in Knowles and Corne (2004).

### Methods

**Decomposition of a K-criteria Problem into Bi-criteria Problems.** Brumbaugh-Smith and Shier (1989) define the Merge of any two nondominated sets \( A \) and \( B \) as the set of nondominated vectors in the union of sets \( A \) and \( B \):

\[
\text{Merge}(A, B) = A \cup B - \{x \in A \cup B \mid x^* \leq x \text{ for some } x^* \neq x, x^* \in A \cup B\} \tag{10}
\]

We define a bi-objective sub-problem of a \( K \)-objective problem as a sub-problem obtained by considering only two of the \( K \) criteria of the original problem. Given \( K \geq 2 \) objective functions, an approximation of the nondominated set of solutions can be determined from the merge of the solutions of all \( \binom{K}{2} \) bi-objective sub-problems. A \( K \)-optimal solution is a solution which is optimal for all \( K \) objective functions.

A \( K \)-optimal solution discovered in one of the bi-objective sub-problems is discovered for the larger \( K \)-criteria problem. Assume that \( x^* \) is a \( K \)-optimal nondominated solution of the \( K \)-objective problem that is discovered in one of the \( \binom{K}{2} \) bi-objective sub-problems. Then the following is true for some pair of criteria, \( i \) and \( j \):

\[
\neg \exists x \text{ s.t. either } f_i(x) \leq f_i(x^*) \text{ and } f_j(x) < f_j(x^*) \tag{11}
\]

\[
\text{or } f_i(x) \leq f_i(x^*) \text{ and } f_j(x) < f_j(x^*)
\]

since \( x^* \) is nondominated in the sub-problem involving criteria \( i \) and \( j \). But since the solution is also \( K \)-optimal, then by (4) the following is true:

\[
\neg \exists x \text{ s.t. } f_i(x) \leq f_i(x^*), i
\]

\[
\text{and } f_j(x) < f_j(x^*) \text{ for some } j \tag{12}
\]

Computational experience shows that a solution discovered in a bi-objective sub-problem is almost certain to be \( K \)-optimal, but it is not guaranteed that this is the case.

**Example 1.** Consider a tri-objective problem in which the complete solution set is

\[
\{(1,3,5), (2,4,2), (3,1,7), (4,9,1), (5,2,2)\}.
\]

Assume we solve three bi-objective sub-problems: one optimizing \( k_1 \) and \( k_2 \), one optimizing \( k_2 \) and \( k_3 \), and one optimizing \( k_1 \) and \( k_3 \). The solution sets are shown in Table 2. The solution sets only show those objectives under consideration. Note, however, that the entire nondominated solution set can be found in the union of the three sets in Table 2:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k_1, k_2 )</td>
<td>{(1,3), (3,1)}</td>
</tr>
<tr>
<td>( k_2, k_3 )</td>
<td>{(1,7), (9,1), (2,2)}</td>
</tr>
<tr>
<td>( k_1, k_3 )</td>
<td>{(1,5), (2,2), (4,1)}</td>
</tr>
</tbody>
</table>

\[
\{(1,3,5), (3,1,7)\} \cup \{(3,1,7), (4,9,1), (5,2,2)\} \cup \{(1,3,5), (2,4,2), (4,9,1)\} = \{(1,3,5), (2,4,2), (3,1,7), (4,9,1), (5,2,2)\}.
\]

Based on the observations above, it is possible to find an approximation set for \( K \)-criteria problems by simultaneously solving bi-objective sub-problems and merging the solutions.

**Multicriteria Shortest Path Algorithms.** If we begin...
with an algorithm that provides a solution set for bi-criteria sub-problems, a reasonable approximation of the non-dominated solution set for the tri-criteria \((K = 3)\) and quad-criteria \((K = 4)\) problems can be found by merging these solution sets. This requires solving 3 and 6 sequences of bi-criteria problems in parallel, respectively. From this discussion we can see at least three strategies exist:

1. Obtain an approximation of the Pareto-optimal set by solving for supported solutions in bi-criteria sub-problems using a weighted-sum approach. Each processor solves a distinct set of bi-criteria sub-problems. Solutions are merged.

2. Apply the weighted-sum algorithm directly to the tri-criteria and quad-criteria problems to find the Pareto-optimal set of supported solutions. Each processor uses a range of weights, and solutions are merged.

3. Obtain an approximation set by solving for all solutions to the bi-criteria sub-problems using label correcting, then merging the solutions.

We provide a computational study of the first two strategies using problem instances with \(K = 3\) and \(4\) and using from 1 to 8 processors.

**Complexity and Scalability.**—Using the first strategy, which is based on Proposition 1, we obtain an approximation of the Pareto-optimal set by solving for the supported solutions of the bi-criteria sub-problems using a weighted-sum approach. The solutions are merged, according to the definition (10) of “Merge().” There are \(Q\) Pareto-optimal solutions, \(\{1,...,Q\}\), discovered in parallel in the bi-criteria sub-problems. The discovery of each of the \(Q\) solutions requires \(O(n^2)\) since the “shortest path” algorithm is based on Dijkstra’s Algorithm which is \(O(n^2)\). Therefore the discovery of all \(Q\) solutions on \(p\) processors requires \(O(Qn^2/p)\), assuming an even distribution of the solutions across the processors. However, in the worst case, it is possible that one of the processors could discover all \(Q\) of the solutions. Recall that we are not using just any arbitrary value for \(p\), but rather the very specific value

\[
p = \binom{K}{2}.
\]

Concretely, this implies that for \(K = 3, p = 3\) and for \(K = 4, p = 6\). In either case, \(p\) could be considered as only a constant in the complexity analysis. This leaves us with a complexity of \(O(Qn^2)\) for the discovery of the solutions. Although it is possible to create a problem instance in which \(Q\) is exponential to the problem size, in practice the size of the solution set is moderated by the restriction that the only solutions allowed are those that are supported in one of the bicriteria sub-problems. Computational experience shows that we can expect in the search for solutions using either of the first two approaches above, that \(Q < n\). Assuming a communications constant of \(\kappa\) to transmit a single solution from the processor on which it is discovered to processor \(P_o\) for output, the communication of the solutions requires \(O(Q\kappa)\). The time required for both computation of the solutions and communication is, therefore, \(O(Qn^2/p + Q\kappa)\).

The time required for the merge of solutions on processor \(P_o\) is, worst case, \(O(Q)\) using a naïve merge algorithm, but using that presented in Brumbah-Smith and Shier (1989), it is \(O(Q)\). Therefore the total time can be expected to be \(O(Qn^2/p + Q\kappa)\). For a large problem the first term can be expected to outweigh the second, and the computational time will be driven by the problem size. For a small problem the second term, communication of the results, will outweigh the first term.

Using the second strategy, the number of iterations depends on the value of \(\lambda\) for \(K = 2\) and the size of the weight distribution table for \(K = 3\) or \(K = 4\). Assuming we generalize and denote by \(\Phi\) the number of iterations, the computational complexity is
Parallel Algorithms for Multicriteria Shortest Path Problems

Published by Arkansas Academy of Science, 2006

129

O(Φn²/p + Qk). We can therefore expect the performance of the first algorithm to be superior to that of the second in terms of quality of the solution set, but the second algorithm will be superior in terms of scalability in that it can run on a cluster of any size.

Results and Discussion

The graphs used for this study were complete graphs with edge costs generated randomly. The results follow the expectations generated by studying the complexity analysis. Figs. 3 and 4 show the results of applying both algorithms to graphs with 500 vertices and 1000 vertices, respectively, with K = 3 criteria. For a problem of either size, we can see that the benefit diminishes beyond the use of 6 processors. With less than 6 processors, performance degrades due to the nature of the first term, specifically its division by p. For more than 6 processors the performance degrades due to increased communication requirements. Figs. 5 and 6 show the results of both algorithms applied to graphs with 500 vertices and 1000 vertices, respectively, with K = 4 criteria. It is difficult to draw a direct comparison between Algorithm 1 and Algorithm 2 in terms of speedup because Algorithm 1 requires a very specific decomposition for parallel computing and a specific number of processors. We can, however, see that Algorithm 1 is generally equal or better, regardless of how many processors are used in Algorithm 2. In all cases the quality of the solution obtained by using Algorithm 1 is superior, since it includes all supported nondominated solutions and some unsupported nondominated solutions. Algorithm 2 finds only the supported nondominated solutions. Due to the static nature of the problem decomposition for Algorithm 1, data pertaining to speedup is not available, so studies of its performance with increasing problem size were conducted. Fig. 7 shows an example of the increase in computational time for increasing problem size observed using Algorithm 1.

Conclusions

Either methodology is easily expandable to problem instances involving more than 4 criteria, although in practice it is unlikely that a network or transportation related problem would have more than 4 or 5 criteria. Even so, both procedures can accommodate higher dimensions. For Algorithm 1, we solve a 5-criteria problem by solving 10 bi-criteria sub-problems, and a 6-criteria problem by solving 15 bi-criteria sub-problems. This requires p = 10 and p = 15 processors, respectively, but for a smaller cluster, multiple solution sets can be found on the same processor. The selection of only the supported solutions to the bi-criteria sub-problems provides a natural filter to the solution set size. As an extension to this research, a search for an approximation set could be implemented by solving for all solutions to the bi-criteria sub-problems using label correcting, then merging the solutions.

Acknowledgments.—The author acknowledges the numerous contributions of Dr. Yupo Chan.

Literature Cited


Application 46:79-86.


