Speed-up of Labelling Algorithms for Biobjective Shortest Path Problems

Andrea Raith
Department of Engineering Science
University of Auckland
New Zealand
a.raith@auckland.ac.nz

Abstract

There is a range of algorithms available to solve biobjective shortest path problems. Here, we focus on biobjective labelling algorithms and propose an acceleration technique, which is easily implemented. We compare the performance of the algorithms with and without the proposed improvements on the basis of test instances with three different network structures. The usage of different data structures within label setting algorithms and their effect on run times is also discussed.

Key words: shortest path problem, label correcting algorithm, label setting algorithm, biobjective optimization, multiobjective optimization.

Introduction

We begin with a formal introduction of the biobjective shortest path problem. Let $G = (N, A)$ be a *directed network* with a set of nodes $N = \{1, \ldots, n\}$ and a set of arcs $A \subseteq N \times N$. Two positive costs $c_{ij} = (c^1_{ij}, c^2_{ij}) \in \mathbb{N} \times \mathbb{N}$ are associated with each arc $(i, j) \in A$. A *path* p in G from node $i_0 \in N$ to node $i_l \in N$ is a sequence $p = \{(i_0, i_1), (i_1, i_2), \ldots, (i_{l-1}, i_l)\}$ of arcs in A. We denote the set of all paths from s to t by $\mathcal{P}_{st}$. The *biobjective shortest path problem* (BSP) with *source node* $s \in N$ and *target node* $t \in N$ can be formulated as:

$$\begin{array}{ll} \min & z(p) = \begin{cases} z_1(p) = \sum_{(i,j)\in p} c^1_{ij} \\ z_2(p) = \sum_{(i,j)\in p} c^2_{ij} \end{cases} \\ \text{s.t.} & p \in \mathcal{P}_{st}. \end{array}$$

The aim is finding all *efficient* paths, i.e. those with the property that none of the two objectives can be improved without worsening the other one. We call path cost vectors of efficient paths *non-dominated*. We consider only solution approaches that generate a *complete* set of solutions, i.e. for all possible non-dominated path cost vectors, there has to be at least one efficient path.

In a recent paper Raith and Ehrgott (2009), we compared different approaches to solve the biobjective shortest path problem. An important class of solution algorithms are biobjective label correcting and label setting algorithms. We found labelling algorithms to be competitive with others such as the two phase method on some problem instances that were investigated.

There is a multitude of papers dedicated to biobjective or multiobjective shortest path problems, in particular about solving them with labelling algorithms. Label setting algorithms for the biobjective problem were discussed by (Hansen 1980; Tung and Chew 1988), whereas (Martins 1984; Tung and Chew 1992; Martins and Santos 2000; Guerriero and Musmanno 2001; Paixão and Santos 2007) tackle the multiobjective problem. Papers on label correcting algorithms include (Henig 1985; Brumbaugh-Smith and Shier 1989; Skriver and Andersen 2000) in the biobjective case and (Corley and Moon 1985; Hartley 1985; Martins and Santos 2000; Guerriero and Musmanno 2001; Sastry, Janakiraman, and Mohideen 2003; Paixão and Santos 2007) in the multiobjective case. For a more detailed discussion of related literature, refer to Raith and Ehrgott (2009) and the references therein.

In the literature, there is no mention of a bounded labelling algorithm such as the one presented in this paper: Here, we aim at further improving the efficiency of biobjective and multiobjective labelling algorithms by exploiting the fact that the cost vector of every enumerated path from source node s to target node t dominates other paths in the network. It may not always be necessary to extend a path to the target node to confirm that it is dominated. All s-t paths enumerated at any time while the algorithm runs, may dominate other paths at t but also at any other node of the network (as arc costs are assumed to be positive). This means that it may be possible to delete paths at an early stage of the algorithm rather than extending them to the target node. In a similar way, bounds derived from supported solutions found in phase 1 of the two phase method can significantly speed up labelling algorithms in phase 2 as demonstrated in Raith and Ehrgott (2009).

We also discuss how different data structures may impact the run times of biobjective label setting algorithms.

The remainder of the paper is organized as follows. Labelling algorithms are introduced in Section 1. We propose bounded biobjective labelling algorithms in Section 2. Different data structure for use in biobjective label setting algorithms are introduced in Section 3. Numerical results are presented in Section 2.2 and 3.2 respectively.

1 Biobjective Labelling Algorithms

All biobjective label correcting as well as label setting algorithms follow the same basic concept. We first introduce a biobjective label correcting algorithm and then highlight the differences to a label setting one.

Initially, the only labelled node is the source node s with its label set $Labels(s) = \{(0,0)\}$. All labels at a particular node i are extended along all outgoing arcs (i,j). Dominated labels are eliminated from those extended labels and the labels already present at the end node j. The remaining labels form the new label set at node j. Whenever the label set of a node changes, the node has to be marked for reconsideration. At reconsideration, the mark of the node is deleted. The algorithm terminates as soon as there are no more nodes marked for reconsideration.

When traversing an outgoing arc from a node with multiple labels, every label has to be extended along this arc and tested for dominance with the labels of the end node of the arc, which is called *merging*. Merging is the most expensive component of a biobjective label correcting algorithm. The label sets are ordered so that the first component is increasing to reduce computational effort of the merge operation.

The algorithm described here follows a *node-selection* strategy as in every iteration a marked node is selected and all its labels are extended. We refer to *biobjective label correcting with node-selection* as C.

Another strategy in label correcting algorithms is called *label-selection*. Here, labels are considered separately, not together with all other labels at the same node. Instead of keeping track of nodes marked for reconsideration, individual labels are marked whenever they change. In every iteration a single (marked) label is selected and extended as described above. Every label that changes is marked, i.e. entered into the list of marked labels. We just mention label correction with label-selection for completeness, but do not implement it here as we are only extending the labelling algorithms from our previous article (Raith and Ehrgott 2009).

A biobjective label setting algorithm always follows the label-selection principle as one has to ensure that the selected label corresponds to an efficient path from s to the node at which the label is situated. We can ensure this by selecting a lexicographically minimal label amongst all tentative labels. This label is then extended and fixed, as it corresponds to an efficient sub-path and is therefore guaranteed to remain non-dominated. We refer to *biobjective label setting* as S.

For more details on label correcting and label setting algorithms as well as pseudocode, the reader is referred to Raith and Ehrgott (2009).

2 Bounded Labelling

We first discuss modifications to the labelling algorithms and then present some numerical results.

2.1 Modification of Labelling Algorithms

C stops when there are no marked nodes any more, whereas S stops once all labels are fixed. In both cases, all efficient paths from s to all other nodes are obtained, including of course the ones to the target node t.

While a labelling algorithm runs, every label at t corresponds to the cost vector of a path from s to t that is currently not dominated. Therefore, this label dominates parts of the objective space as no label that is dominated by it can represent an efficient path. Labels at *any* node of the network that are dominated by a label at t can be deleted as path costs are non-negative so that once a label anywhere in the network is dominated by a label at t it remains dominated. It is therefore not necessary to extend this label until its path reaches t, it can be deleted as soon as dominance is detected.

We propose a bounded labelling algorithm by modifying any of the labelling algorithms as follows: The algorithm runs as described in Section 1 while there is no label at node t. Once there is at least one label at t, one can start checking bounds. For each newly generated label, one checks whether it is dominated by at least one label at target node t. The labels at the target node, $l_1 = (z_1^1, z_2^1), \ldots, l_m =$

(z_1^m, z_2^m), are sorted by increasing first objective value, i.e. $z_1^1 < z_1^2 < \ldots < z_1^m$ and $z_2^1 > z_2^2 > \ldots > z_2^m$. It is checked if the newly generated label $l = (z_1, z_2)$ is dominated:

```
1: set $dominated$ = FALSE and $i = 1$
2: while ($dominated$ == FALSE) and ($i \leqq m$) and ($z_1^i \leqq z_1$) do
3:    if $z_2^i \leqq z_2$ then
4:       set $dominated$ = TRUE
5:    else
6:       $i = i + 1$
7:    end if
8: end while
```

Only labels that are not dominated by any of the labels at t are retained, all others are deleted. It is easy to implement this check into any labelling algorithm. The resulting algorithms are called *bounded labelling* algorithms and denoted by bC and bS. Although, we formulate the dominance check for the biobjective problem here, our idea is easily applicable to multiobjective problems.

It should be noted that biobjective labelling algorithms actually yield the shortest path from s to all other nodes, not just to the target node t. With our improvement the algorithm is restricted to finding the shortest paths from s to t. If the aim was obtaining shortest paths from s to a small number of target nodes $t_1, \ldots, t_l$, the bound check could be modified to only deleting a label if it is dominated by at least one label at *each* target node $t_1, \ldots, t_l$. But considering too many target nodes might diminish the effectiveness of the bounds.

In some cases it can be determined that a label cannot be dominated by any of the labels at t without actively checking all of them. Let the labels at the target node, $l_1, \ldots, l_m$, be sorted by increasing first objective value as above. A label $l = (z_1, z_2)$ at any node will clearly not be dominated by any of the labels $l_1, \ldots, l_m$ if $z_1 < z_1^1$ or $z_2 < z_2^m$. Note that checking $z_1 < z_1^1$ is included as part of the while loop of the dominance check. If $z_2 < z_2^m$, it is not necessary to check a label l against all labels $l_1, \ldots, l_m$ when it is clear a priori that l cannot be dominated. We implemented this additional condition, but achieved no further run time improvements, which is why the corresponding results are not reproduced here.

It should be noted that an approach related to the proposed bounding is A* search, whose multiobjective extension was introduced by Stewart and White (1991) and further extended by Mandow and Pérez de la Cruz (2003), Mandow and Pérez de la Cruz (2006). The concept of A* search is based on guiding the search towards the target by selecting a label based on its value plus some (heuristic) prediction of the label's distance to its destination. In the process dominance by existing labels at the target can also be tested. Here, we only compare existing labels to those at target nodes.

2.2 Numerical Experiments

All numerical experiments are conducted using the same problem instances as in Raith and Ehrgott (2009). The networks have three different basic structures. The first network type consists of road networks of the US, namely those of the three states Washington DC (DC), Rhode Island (RI), and New Jersey (NJ). For each state, nine instances with different source and target nodes were generated. We

Table 1: Average run times for biobjective labelling algorithms for road networks (DC, RI, NJ), grid networks and NetMaker networks (NM).

	avg run time (sec)		ratio *bC* over *C*			run time (sec)		ratio *bS* over *S*		
name	C	bC	avg	min	max	S	bS	avg	min	max
DC	0.26	0.06	0.33	0.00	1.00	0.12	0.03	0.06	0.00	0.15
RI	5.42	2.11	0.34	0.03	0.87	1.80	0.63	0.39	0.04	0.95
NJ	30.46	10.23	0.33	0.02	0.78	19.74	5.95	0.28	0.01	0.83
grid	2.74	3.79	1.13	0.82	1.78	19.47	21.40	1.01	0.77	1.19
NM	368.21	0.00	0.00	0.00	0.00	1205.15	0.00	0.00	0.00	0.00

refer to these instances as DC, RI, and NJ. Networks in the second group have a rectangular grid structure, and instances are denoted by grid. There are 33 different grid networks. The last group consists of random networks with a structure similar to the NetMaker networks described by Skriver and Andersen (2000). We have 60 different NM networks. Instances in this group are denoted by NM. All instances vary in problem size and number of efficient solutions.

In Table 1 the average run time of the different algorithms is listed in columns C, bC, S, and bS. For C and S we use the run times from Raith and Ehrgott (2009). The average, minimal and maximal ratio of run times of the bounded algorithms over the original ones is listed in columns *bC over C* and *bS over S*. If this ratio is equal to 1, run times of both approaches are identical. If the ratio is between 0 and 1, the bounded algorithms have a shorter run time than the original ones. Otherwise, if the ratio is larger than 1, the original algorithms perform better.

All algorithms were implemented in C and compiled with the gcc compiler (version 4.1.1). Numerical tests are performed on a Linux computer with 2.40GHz Intel®*Core*™2 Duo processor and 2GB RAM. Run time is measured in seconds with a precision of 0.01 seconds, a run time of 0.00 represents any run time < 0.01. Algorithms were stopped after 3600 seconds. When the run time of the label correcting or label setting algorithms is $\leqq 0.1$, we do not consider the corresponding ratio in the calculation of the average ratio, as the run time itself is too small to enable meaningful comparisons.

In the case of road networks (rows DC, RI, and NJ in Table 1), the bounded algorithms bC and bS both improve the run time significantly when compared with the original algorithms C and S. For the three network types we observe an average ratio of 0.33 for bounded label correcting and 0.28 for bounded label setting. These ratios indicate that the achieved run time improvements of the bounded labelling algorithms are significant.

Grid networks (row 'grid' in Table 1) on the other hand have a network structure that does not permit (average) improvement of run time through bounded label correcting, exhibited by the average ratio $1.13 > 1$. Here, the additional effort of checking for every newly created label whether it is dominated by any of the (often many!) labels at t seems much larger than what is saved by discarding labels occasionally. For the label setting algorithm, we observe similar run times of the bounded algorithm compared to the original one with an average ratio of 1.01.

The most striking results appear for the NetMaker instances (row 'NM' in Table 1). Here, run time is always reduced to $\leqq 0.01$ although the run time of C and S is very high in most cases leading to an average ratio of 0.00 in all four cases. It is interesting to note that even though the NM instances investigated have different sizes, reflected in increasing run times of C and S, as NM instances grow larger.

However, both bC and bS consistently finish in less than 0.01 seconds. This can be explained via the structure of the networks. The n nodes are numbered consecutively. From every node, arcs can only reach a certain number of nodes "forward" and "backward" leading to an elongate network structure with many possible paths from source node $s = 1$ to origin node $t = n$. The instances were constructed to wrap around, so that arcs from nodes with low numbers that reach backwards may connect to a node with very high number, i.e. close to the target node. Note that this may not be exactly the network structure the authors of Skriver and Andersen (2000) had in mind originally. In many instances, only few efficient paths exist which are also very short as they reach "backwards" from the nodes with low numbers to those with high numbers. For any labelling algorithm to finish, however, it is necessary to generate the paths from the source to all other nodes, whereby many long paths are enumerated that can never be efficient once they reach t. The bounds are very effective here, because efficient s-t paths are found quickly and have fairly low costs in both components, so that many labels can be discarded.

3 Data Structure in Biobjective Label Setting Algorithms

An another approach to improve the computational performance of biobjective labelling algorithms is to select the most appropriate data structures. An example of similar work for single objective label setting and correcting algorithms is Cherkassy, Goldberg, and Radzik (1996), where the computational performance based on different data structures was compared.

3.1 Data Structures

We focus here on biobjective label setting algorithms, which require, in each iteration, the selection of tentative labels that are guaranteed to remain non-dominated as briefly discussed at the end of Section 1. Criteria for selection are for example the selection of a tentative label which (Tung and Chew 1988; Paixão and Santos 2007) is lexicographically minimal, has minimal z_1 value (if all $c_{ij} > 0$), has minimal z_1 or minimal z_2 value (if all $c_{ij} > 0$), or has minimal $z_1 + z_2$ value. With each selection strategy it is necessary to extract a minimal label of some kind from the set of tentative labels. Hence, it is worth investigating how to most efficiently achieve this within the label setting algorithm.

We briefly describe four data structures we use below. Figure 1 shows the same set of tentative labels inserted into the different data structure explained below based on lexicographic ordering of z_1 and z_2. The set of labels is $(10, 10)$, $(10, 21)$, $(12, 28)$, $(13, 14)$, $(13, 15)$, $(13, 28)$, $(14, 28)$, $(16, 15)$, $(17, 28)$, $(19, 23)$, $(21, 23)$, $(24, 15)$.

A very simple data structure is to keep the tentative labels in an ordered linked list, which means the first element is always minimal, the second element is the second smallest, etc.

While it is important to have access to the minimal element, all remaining tentative labels do not need to be ordered. Thus a binary heap or Fibonacci heap, which keep the set of tentative labels only partially ordered, may increase the efficiency (over an ordered list) of inserting and extraction of tentative labels (Ahuja, Magnanti, and Orlin 1993). A binary heap is a tree, in which every node has a smaller key (value by which is is sorted) than its two child nodes, hence the root node is the

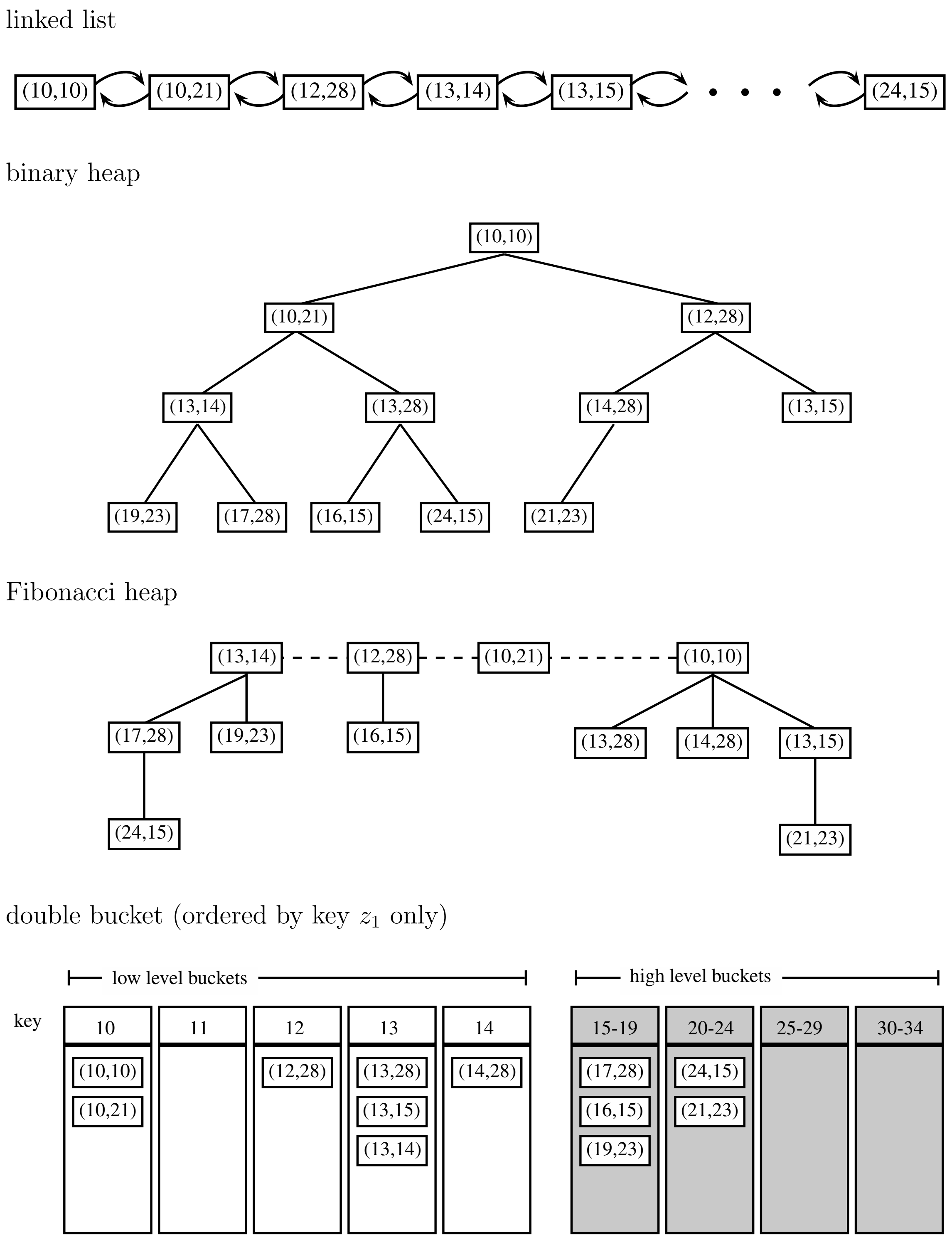

Figure 1: Same set of tentative labels in different data structures.

Table 2: Average run times for biobjective label setting algorithm with different data structures for different network types.

	avg run time (sec)				average ratio binary heap over ...		
name	list	Fibonacci Heap	binary heap	double bucket	list	Fibonacci heap	double bucket
DC	1.67	0.26	0.13	0.12	0.08	0.49	1.18
RI		3.62	2.28	2.08		0.57	1.14
NJ		32.41	22.92	20.72		0.66	1.12
grid	151.82	26.09	22.34	47.4	0.2	0.66	0.79
NM		548.51	511.53	632.86		0.9	0.62

Note: NM results are for 40 smallest NM networks only, which already have high run times.

Numerical tests performed in slightly different computer hence run times differ slightly from those in Table 1.

minimal element but other elements are only partially ordered. A Fibonacci heap contains a forest of trees of different sizes. Again, each parent node has smaller key than its child nodes, but the tree structure is more flexible as every node can have several child nodes. The minimal element is the root node of one of the trees.

It was found by Cherkassy, Goldberg, and Radzik (1996) that double-buckets are an efficient data structure for single objective label setting. A bucket contains labels with a certain key value. A double bucket data structure has two levels of buckets: Low level buckets collect labels that have the same key, whereas high level buckets each collect labels within a range of keys. The buckets for immediate consideration are low level buckets, labels can be selected from the non-empty bucket with lowest key until all low level buckets are empty. Then, elements of the first high level bucket are sorted into the low level buckets for easy access.

Results for single objective shortest path problems are not directly transferable to bi- or multiobjective shortest path problems as there are many more labels in the latter case, due to multiple tentative labels at each node. Hence, we study the effect of using different data structures for the biobjective case.

It should be noted that the first to investigate the effect of data structures on run time of multiobjective shortest path algorithms are Paixão and Santos (2007), compare binary heaps, linked lists, and a single bucket data structure.

3.2 Numerical Experiments

We investigate how the usage of the data structures discussed in Section 3.1 affect the run times for the problem instances discussed in Section 2.2 above. In all our experiments we select lexicographically minimal labels as this appears to be the most common selection criterion in the literature. A double bucket data structure considers all elements that have the same key as being equal, i.e. any minimal element can be selected. This would require many low level buckets for different combinations of the two components of tentative labels. To efficiently use the data structure, we choose to use minimal z_1 as key in this case, which is sufficient as we assume $c_{ij} > 0$.

Table 2 shows average run times of biobjective label setting with each of the data structures used to manage tentative labels. It should be noted that the list data structure mostly had excessive run times, hence results are only reported for DC networks and grid networks.

As the data structure used within the biobjective label setting algorithm in Section 2 is the binary heap, we compare all run times with those of the binary heap implementation. Hence ratios of the binary heap run time over all other run times are computed. Ratios > 1 indicate an approach is, on average, faster than binary heap, and ratios < 1 indicate an approach is slower.

As expected the list data structure consistently performs worst as the computational effort of keeping all labels ordered is large. All other data structures perform worse than the binary heap except for the double bucket data structure in case of the road network instances (DC, NJ, RI). Furthermore, the double bucket data structure clearly outperforms the list data structure but also the Fibonacci heap.

Conclusion and Future Research

The proposed bounded labelling techniques for BSP were shown to be able to improve the run time of biobjective labelling algorithms for two out of three network types. Unfortunately we did not observe any improvements in run time in case of grid networks. Performance of the label correcting algorithms for road networks was improved by 67% on average, that of the label setting algorithm even by an average 72%. The most impressive run time improvements were seen for NetMaker networks where the bounds were able to exploit the network structure very efficiently leading to run time improvements of 100%, i.e. decreasing run time to $\leqq 0.01$ for all instances.

Both for the biobjective and the multiobjective problem, a promising approach might be to compute a few single objective shortest s-t paths initially, such as the lexicographically minimal shortest paths or maybe some weighted sum solutions. These initial solutions can be exploited as bounds from the start of the biobjective labelling algorithm overcoming the problem that bounds can only be used in the bounded labelling algorithms once the first label of an s-t path is generated.

It was also shown how different data structures in biobjective label setting may affect computational performance. The adaptation of speed-up techniques known for single objective shortest path problems to biobjective and multiobjective problems would certainly be worthwhile investigating, for example data structures can be exploited as outlined above. Of course the effect on run time of many more data structure can be studied, also for multiobjective label correcting algorithms.

Acknowledgments

I thank Matthias Ehrgott for his comments and suggestions. Thank you also to Kuan-Min Lin who implemented different data structures in bi-objective label setting algorithms as part of his part IV project in Engineering Science in 2009.

References

Ahuja, R.K., T.L. Magnanti, and J.B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall.

Brumbaugh-Smith, J., and D. Shier. 1989. “An empirical investigation of some bicriterion shortest path algorithms.” *European Journal of Operational Research* 43 (2): 216–224.

Cherkassy, B.V., A.V. Goldberg, and T. Radzik. 1996. “Shortest path algorithms: Theory and experimental evaluation.” *Mathematical Programming* 73:129–174. Implementations of algorithms obtainable at `http://www.avglab.com/andrew/soft.html`.

Corley, H.W., and I.D. Moon. 1985. "Shortest Paths in Networks with Vector Weights." *Journal of Optimization Theory and Applications* 46 (1): 79–86.

Guerriero, F., and R. Musmanno. 2001. "Label Correcting Methods to Solve Multicriteria Shortest Path Problems." *Journal of Optimization Theory and Applications* 111 (3): 589–613.

Hansen, P. 1980. "Bicriterion Path Problems." Edited by G. Fandel and T. Gal, *Multiple Criteria Decision Making, Theory and Application. Proceedings of the 3rd International Conference, Hagen/Königswinter 1979*, Volume 177 of *Lecture Notes in Economics and Mathematical Systems.* Springer Verlag, Berlin, 109–127.

Hartley, R. 1985. "Vector Optimal Routing by Dynamic Programming." In *Mathematics of Multiobjective Optimization*, edited by P. Serafini, CISM International Centre for Mechanical Sciences – Courses and Lectures, 215–224. Wien: Springer Verlag.

Henig, M. 1985. "The Shortest Path Problem with Two Objective Functions." *European Journal of Operational Research* 25:281–291.

Mandow, L., and J.L. Pérez de la Cruz. 2003. "Multicriteria heuristic search." *European Journal of Operational Research* 150:253–280.

———. 2006. "Comparison of heuristics in multiobjective A* search." *Current Topics in Artificial Intelligence, Selected Papers from the 11th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2005)*, Lecture Notes in Artificial Intelligence.

Martins, E.Q.V. 1984. "On a multicriteria shortest path problem." *European Journal of Operational Research* 16:236–245.

Martins, E.Q.V, and J.L. Santos. 2000. "The labelling algorithm for the multiobjective shortest path problem." Technical Report, Universidade de Coimbra, Portugal, Departamento de Matématica. `http://www.mat.uc.pt/~eqvm/cientificos/investigacao/mo_papers.html`.

Paixão, J.M., and J.L. Santos. 2007. "Labelling methods for the general case of the multi-objective shortest path problem - a computational study." Technical Report 07–42, Universidade de Coimbra.

Raith, A., and M. Ehrgott. 2009. "A comparison of solution strategies for biobjective shortest path problems." *Computers & Operations Research* 36:1299–1331.

Sastry, V.N., T.N. Janakiraman, and S.I. Mohideen. 2003. "New Algorithms For Multi Objective Shortest Path Problem." *Opsearch* 40 (4): 278–298.

Skriver, A.J.V., and K.A. Andersen. 2000. "A label correcting approach for solving bicriterion shortest-path problems." *Computers & Operations Research* 27:507–524.

Stewart, B.S., and C.C. White. 1991. "Multiobjective A*." *Journal of the Association for Computing Machinery* 38:775 – 814.

Tung, C.T., and K.L. Chew. 1988. "A Bicriterion Pareto-optimal path algorithm." *Asia-Pacific Journal of Operational Research* 5:166–172.

———. 1992. "A multicriteria Pareto-optimal path algorithm." *European Journal of Operational Research* 62:203–209.