An Efficient Label-setting Algorithm for the Bi-objective Shortest Path Problem

Antoine Giret[1], Yannick Kergosien[1], Gael Sauvanet[2] and Emmanuel Neron[1]

[1]*Université François Rabelais de Tours, CNRS, LI EA 6300, OC ERL CNRS 6305, 64 avenue Jean Portalis, 37200 Tours, France*

[2]*La Compagnie des Mobilités, 30 rue André Theuriet, 37000 Tours, France*

Keywords: Label-setting Algorithm, Pareto front, Bi-Objective Shortest Path Problem, Cycling, Exact Method.

Abstract: In this paper, we consider a classical Bi-objective Shortest Path problem (BSP) that takes into account both distance and insecurity criteria. This work is in collaboration with an enterprise who provide a web platform called Géovélo that aims to propose routes for cycling. We propose a new exact method to solve a BSP, called Label Setting algorithm with Dynamic update of Pareto Front (LSDPF), which aims to find all non-dominated solutions of the problem. Different exploration strategies have been proposed and tested. Numerical experiments on real data sets and on instances of the literature were conducted. Comparison with recent benchmarks algorithms solving BSP - the bounded Label Setting algorithm by (Raith, 2010) and the pulse algorithm by (Duque et al., 2015) - shows that our method outperform these benchmarks algorithms.

1 INTRODUCTION

For economical and ecological reasons, alternative transportation such as cycling is expanding. Public actors have initiated several programs to adapt the road network for cycling (e.g., greenway, cycling lane), because users seek to be able to move safely. For a cyclist, travelling time and distance are not the only criteria to consider; they may also consider safety, difficulty and tourist attractions.

This work is in collaboration with *La Compagnie des Mobilités* who provide a web platform called Géovélo (http://www.geovelo.fr). Géovélo aims to propose routes for cycling. These routes consider two criteria which are safety (minimising lack of security), and distance. Both criteria are very conflicting, mainly because most road networks are designed for cars.

The studied problem is a Bi-Objective Shortest Path Problem: let $G = (V,A)$ be a directed graph where V is the set of nodes ($|V| = n$) and A the set of arcs ($|A| = m$). $\{c^1_{ij}, c^2_{ij}\}$ denotes the costs associated with each arc (i, j), where c^1_{ij} represents its distance and c^2_{ij} its insecurity. The problem consist in finding a path $\mathcal{P}$ from a source node s to a target node t minimizing both criteria $\{\sum_{(i,j)\in\mathcal{P}} c^1_{ij}, \sum_{(i,j)\in\mathcal{P}} c^2_{ij}\}$. The result of the problem is a set of strictly non-dominated solutions also called Pareto front defined in (Ulungu and Teghem, 1994).

The Bi-Objective Shortest Path Problem, noted BSP, is a specific case of the Multi-Objective Shortest Path Problem, that is one of the most studied problems among the Multi-Objective Combinatorial Optimisation (MOCO) problems. (Serafini, 1987) has proved that the BSP is $\mathcal{NP}$-Hard. (Tarapata, 2007) presents a state-of-the-art MOSP problem.

The first methods solving BSP are label-correcting and label-setting methods proposed by (Hansen, 1980) and (Martins, 1984). Labelling methods can be seen as a generalization of existing methods solving the mono-objective problem (Bertsekas, 1998) and they can be decomposed into two types: label-correcting and label-setting. The difference between these types is how the label queue is managed. Label-setting methods often use a lexicographic order to ensure that an explored label corresponds to a non-dominated path (see (C.T. Tung, 1992)). Label-correcting methods use a simple order to manage label queue exploration (e.g., FIFO) to simplify the data structure of the queue (see (Brumbaugh-Smith and Shier, 1989) and (Skriver and Andersen, 2000)). There are two different selection types for the label-correcting methods: a label is managed separately in label selection, whereas all labels on a selected node are extended at the same time in node selection. (Guerriero and Musmanno, 2001) compares different node and label selection strategies for the label-correcting method. Node selection is generally used

Giret, A., Kergosien, Y., Sauvanet, G. and Neron, E.
An Efficient Label-setting Algorithm for the Bi-objective Shortest Path Problem.
DOI: 10.5220/0005757101970203
In *Proceedings of 5th the International Conference on Operations Research and Enterprise Systems (ICORES 2016)*, pages 197-203
ISBN: 978-989-758-171-7

with label-correcting methods, whereas label selection must be used with label-setting methods.

Another type of method to solve BSP is Ranking method (Climaco and Martins, 1982) that are based on the *k*th-shortest path problem ((Carlyle and Wood, 2005)). These methods find the shortest path optimizing one objective, then the second shortest path, and so on. In addition to labelling and ranking methods, the two-phase approach is also commonly used to solve BSP problems. (Ulungu and Teghem, 1995) introduced this concept. The two-phase method is basically used to compute separately supported (i.e., located on the boundary of the convex hull) and non-supported solutions. All solutions in the first phase can be computed by a single objective method with a weighted sum objective. In the second phase, a bi-objective method enumerates all non-supported solutions. During this phase, the search space is restricted by the solutions computed during the first phase. (Raith and Ehrgott, 2009) has shown that this approach is efficient and proposed a comparison of the solution strategies for finding all efficient paths. (Raith, 2010) focuses on label-correcting and label-setting algorithm (bLSET) and propose an acceleration technique, improving the efficiency of these algorithms. He showed that it is not always necessary to continue the search to the target node to confirm that it is dominated. According to (Demeyer and al., 2013), bLSET algorithm presented better computational times among labelling approaches proposed during this period.

Recently, (Duque et al., 2015) proposed a new exact method, called Pulse algorithm, for the BSP and large-scale road networks. Pulse algorithm is based on recursive method using pruning strategies that accelerate the graph exploration. The results shown that the proposed algorithm outperform the bLSET algorithm on very-large scale instances from the DIMACS dataset.

This work is an extension of (Sauvanet and Neron, 2010) and aims to propose a new exact method to solve BSP. The proposed method, called Label Setting algorithm with Dynamic update of Pareto Front (LSDPF), is based on existing methods previously cited. However, some improvement techniques have been added. The paper is organized as follows. Section 2 presents the LSDPF algorithm with several exploration strategies. Some numerical results are given in section 3 in order to test the different exploration strategies and a parameter of the algorithm. Then computational experiments compare our proposed method to the bLSET algorithm developed by (Raith, 2010) and the Pulse algorithm developed by (Duque et al., 2015). Finally section 4 concludes the paper and gives some future work directions.

2 LABEL SETTING ALGORITHM WITH DYNAMIC UPDATE OF PARETO FRONT (LSDPF)

The LSDPF algorithm is based on a two-phase method introduced by (Ulungu and Teghem, 1995). The first phase aims to compute dominated solutions by solving several Mono-Objective Shortest Path Problems. The second phase aims to get only non-dominated solutions and is based on classic label-setting algorithm which has been introduced in (Martins, 1984). The specificity of the proposed algorithm is that the final Pareto front can dynamically evolve at each iteration of the label-setting algorithm, and not necessary when the target node is reached.

2.1 First Phase

As explained shortly before, the first phase consists in running a set of Mono-Objective Shortest Path Problems (using a simple adaptation of (Dijkstra, 1959)), and is composed by two steps.

The first step provides an upper bound of a given criterion by solving a Mono-Objective Shortest Path Problem from s to t, minimizing the other criterion. These upper bounds allow to reduce the exploration of the graph for the next step.

The second step consists in solving reverse one-to-all Mono-Objective Shortest Path Problems from t, where each objective function is defined by a linear combination of both criteria:

$$\alpha \sum_{(i,j)\in\mathcal{P}} c^1_{ij} + (1-\alpha) \sum_{(i,j)\in\mathcal{P}} c^2_{ij}$$

We note $\mathcal{A}$ the set of all α values tested. After each one-to-all resolution, each traversed node is characterized by a pair of values (d_i^α, s_i^α), representing a feasible path from i to t with a distance equal to d_i^α and an insecurity equal to s_i^α. This search is stopped at a node i when both d_i^α and s_i^α are greater than upper bounds found in the first step. Only non dominated pairs - in Pareto sense - of values (d_i^α, s_i^α) are saved.

Two specific cases are important: $\alpha = 1$ and $\alpha = 0$. Both allows to determine the lower and the upper bound for each criterion at each node:

- For $\alpha = 1$, is obtained at each node $i \in V$ the lower bound of the distance criterion defined by $(LB_i^1 = d_i^1)$, which is associated to an upper bound of the insecurity criterion defined by $(UB_i^2 = s_i^1)$,

- For $\alpha = 0$, is obtained at each node $i \in V$ the lower bound of the insecurity criterion defined by ($LB_i^2 = s_i^0$), which is associated to an upper bound of the distance criterion defined by ($UB_i^1 = d_i^0$).

Finally, the set of pairs (d_s^α,s_s^α) is used to initialize the Pareto front. Additional one-to-one Mono-Objective Shortest Path from s to t can be done (i.e. using more values of α), in order to extend the initial Pareto Front (cf figure 1).

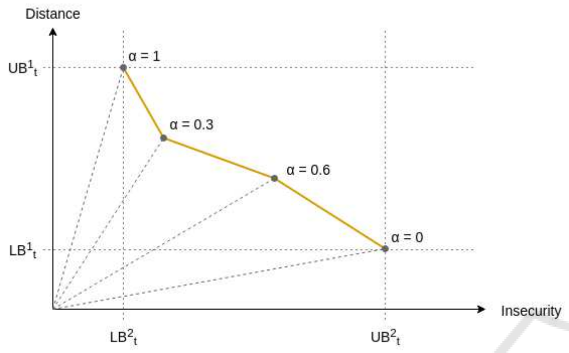

Figure 1: Initialisation of the Pareto front.

2.2 Second Phase

The second phase is based on a label-setting algorithm with additional improvements related to the Bi-Objective Shortest Path problem. These improvements allow to reduce the number of labels created and thus finding the final Pareto front faster. The proposed method (LSDPF) is presented in the algorithm 1. Let us define:

- s and t respectively the source and target nodes.
- i and j the indexes of nodes and u, v, w and x the indexes of labels.
- l_u a label, defined by a tuple (i, vd, vs) where:
  - i is the node associated to the label.
  - vd and vs represent respectively the values of distance and insecurity of a feasible path from s to i.
- LN_i the set of labels at node i.
- Q contains unexplored labels.
- $l_v \prec_p l_u$ means that the label l_v is strictly dominated by l_u in the Pareto sense.

The final Pareto front contained all non-dominated solutions is obtained at node t: it corresponds to the set of labels LN_t.
The main steps of the LSDPF algorithm are:

- Line 1: the FIRSTPHASE function represents the first phase of the proposed method explained in the section 2.1. It allows to determine all pairs (d_i^α,s_i^α). The Pareto front is initialized in line 4.
- Line 7: the NEXTLABEL function defines the order to explore the label, which affects the evolution of the Pareto front and thus the convergence of the algorithm. Three strategies explained later have been tested.
- Line 12: the current label is explored if and only if it is not dominated by another label existing on the same node.
- Line 13: the current label l_u is explored if and only if it is promising, i.e. if $(j, vd' + LB_j^1, vs' + LB_j^2)$ is not dominated by a label existing on the target node. $(j, vd' + LB_j^1, vs' + LB_j^2)$ represents a not

Algorithm 1: Label Setting algorithm with Dynamic update of Pareto Front (LSDPF).

```
1:  FIRSTPHASE()
2:  $LN_i \leftarrow \emptyset, \forall i = \{1,\dots,n\}$
3:  $LN_s \leftarrow \{(s,0,0)\}$
4:  $LN_t \leftarrow \{\cup_{\forall\alpha}(d_s^\alpha, s_s^\alpha)\}$
5:  $Q \leftarrow \{(s,0,0)\}$
6:  while $Q \neq \emptyset$ do
7:      $(i,vd,vs) \leftarrow$NEXTLABEL(Q)
8:      $Q \leftarrow Q \backslash (i,vd,vs)$
9:      for all $(i,j) \in A$ do
10:         $(j,vd',vs') \leftarrow (j, vd + c_{ij}^1, vs + c_{ij}^2)$
11:         $l_u \leftarrow (j,vd',vs')$
12:         if $\nexists l_v \in LN_j / l_v \prec_p l_u$ then
13:             if $\nexists l_v \in LN_t / l_v \prec_p (j, vd' + LB_j^1, vs' + LB_j^2)$ then
14:                 if $j \neq t$ then
15:                     $Q \leftarrow Q \cup l_u$
16:                 end if
17:                 $LN_j \leftarrow LN_j \cup l_u$
18:                 UPDATELABELS(Q, $LN_j$, $l_u$)
19:                 DPF($LN_t$, $(j,vd',vs')$)
20:             end if
21:         end if
22:     end for
23: end while
24: return $LN_t$
```

Algorithm 2: UPDATELABELS.

```
1: function UPDATE(Q, $LN_j$, $l_u$)
2:     for all $l_v \in LN_j / l_u \prec_p l_v$ do
3:         $LN_j \leftarrow LN_j \setminus l_v$
4:         if $l_v \in Q$ then
5:             $Q \leftarrow Q \setminus l_v$
6:         end if
7:     end for
8: end function
```

Algorithm 3: DPF.

```
1:  function DPF($LN_t$, $(j, vd', vs')$)
2:      for all $\alpha \in \mathcal{A}$ do
3:          $l_v \leftarrow (t, vd' + d_j^\alpha, vs' + s_j^\alpha)$
4:          if $\nexists l_w \in LN_t / l_w \prec_p l_v$ then
5:              $LN_t \leftarrow LN_t \cup l_v$
6:              for all $l_x \in LN_t / l_v \prec_p l_x$ do
7:                  $LN_t \leftarrow LN_t \setminus l_x$
8:              end for
9:          end if
10:     end for
11: end function
```

necessary feasible path from s to t, containing two subpaths. The first subpath is given by the label l_u (which is a feasible path from s to j), and the second subpath corresponds to the lower bounds of both criteria to reach t from j. In other words, the label l_u cannot give better solutions than those in the current Pareto front. During the search, the Pareto front will iteratively contain better solutions involving that more and more labels will not be explored. This explains why the initialization of the Pareto front is important as well as its updates (line 19).

- Line 18: the UPDATELABELS function is presented in the algorithm 2. The goal of this function is to update Q and LN_j, according to the current explored label l_u. All dominated labels in Q and LN_j are deleted since they are not interesting ones.
- Line 19: the DPF function is presented in the algorithm 3. It allows to dynamically update the Pareto front. From the current explored label l_u, several feasible paths from s to t can be computed thanks to each solution of Mono-Objective Shortest Path Problems determined in the first phase (cf figure 2). This kind of path is decomposed into two subpaths: a subpath from s to j given by the label l_u, and a second one from j to t given by the solution of the linear combination determined by α during the Mono-Objective Shortest Path Problem. This step aims to improve the Pareto front throughout the search, and thus reducing the number of explored labels (line 13 of algorithm 1).

2.3 Exploration Strategies

The exploration strategy determines in which order the labels will be explored (goal of the NEXTLABEL function in the algorithm). From a strategy to another one, the number of explored labels can be different

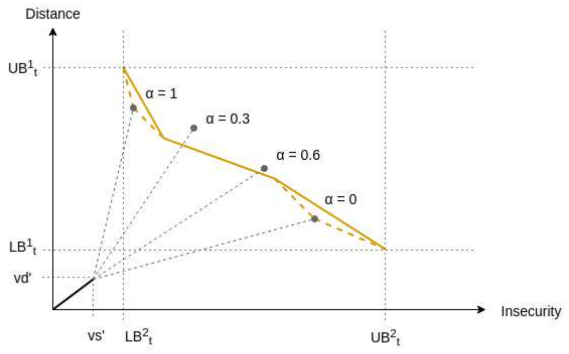

Figure 2: Dynamic update of the Pareto front.

as well as the computation time of the LSDFP algorithm. The selected strategy is thus important, three strategies are presented and have been tested.

2.3.1 Strategy 1

The first strategy is simple. It consists in selecting the label that contains the minimum vd' value among all labels presents in Q. If several labels have the same minimum value vd', the label with the smallest value of vs' is selected among these labels. The graphic evolution of the set of explored nodes/labels can be represented by a circle around the source node, which become increasingly larger. Only distance criterion as first criterion has been tested (and not the insecurity one). Indeed, the range of distance values is bigger than the range of insecurity values in cycling context. Considering the distance criterion first, the search will be guided to more interesting labels.

2.3.2 Strategy 2

The second strategy aims to explore the most promising label first. From the set of labels presents in Q, the next label to treat is the label with the smallest value $(vd' + LB_j^1)$. In case of equalities, the label with the smallest value $(vs' + LB_j^2)$ is selected. As previously, only distance criterion is tested as first criterion for the the same reason. The representation graph of explored labels is close to an ellipsoid around the source and target nodes.

2.3.3 Strategy 3

The last strategy uses a linear combination of $vd' + LB_j^1$ and $vs' + LB_j^2$. Unlike other strategies, this one will take into account both criteria to guide the exploration. Thus the next label presents in Q to explore is the label minimizing the function:

$$\beta * (vd' + LB_j^1) + (1 - \beta) * (vs' + LB_j^2)$$

3 COMPUTATIONAL EXPERIMENTS

The LSDPF algorithm has been implemented in C++ language using the library Boost http://www.boost.org. Computational experiments have been performed on an Intel Core i7 vPro processor quad-core with 8GB of RAM and using Linux system. First, numerical results are given for the three strategies and several values of $\mathcal{A}$ on real data sets taking into account the study case (Cycling). Next, the efficiency of the LSDPF algorithm with the best exploration strategy is compared to efficient benchmarks algorithms on instances of the literature.

3.1 Comparison of Exploration Strategies

To compare the different exploration strategies presented in section 2.3, we used real roads networks that come from the cooperative OpenStreetMap project (http://www.openstreetmap.org). Each node of the network represents an intersection and each arc between two intersections is composed by two values: the travel distance and the insecurity of the road. Three graphs have been extracted: Paris (France), Berlin (Germany) and San Francisco Bay Area (USA). The Paris graph used represents the city of Paris and its 30 adjacent towns, and contains 29,086 nodes and 64,538 arcs. The second graph considered in this study corresponds to the city of Berlin. This one is interesting because Germany is a country in which the OpenStreetMap community is active and thus safety informations are accurately reported. It contains 59,673 nodes and 145,840 arcs. The San Francisco graph is interesting because the road network structure of US cities is rather different and bigger than European cities. It contains 174,975 nodes and 435,959 arcs. For each graph, we randomly generated 50 instances (i.e. pair of source and target nodes) for Paris and Berlin graphs, and 25 instances for the San Francisco one.

The table 1 presents the average execution time (in milliseconds) of LSDFP algorithm for each strategy and each graph. The parameter β is equal to 0.5. The first column indicates the graph name, columns from 2 to 4 shows the average execution time of each strategy, and last column shows the average number of non-dominated solutions on the Pareto front.

The same interpretation can be done for all the graphs: the second strategy seems to be better than the first one as we might expect but also than third one, which can be surprising. The second strategy improves by approximately 20% the execution time

Table 1: Average execution time (ms) of LSDPF for each strategy.

Graph	Str. 1	Str. 2	Str. 3	$\lvert S \rvert$
Paris	194,8	158,6	403,6	117
Berlin	206,2	150,8	246	127
SF	87040,8	71815,2	715942,4	1280

of the first strategy and between 40% and 90% execution times of the third strategy. Results of the third strategy are not as good as expected. The linear combination of both criteria doesn't seems guide the exploration towards good solutions quickly.

The table 2 presents the average number of explored labels during the LSDPF algorithm, for each strategy and each graph. The first column indicates the graph name, columns from 2 to 4 shows average number of explored labels. An explored label means that it has been added to the Q set (line 15 of LSDPF algorithm).

Table 2: Average number of explored labels for each strategy.

Graph	Str. 1	Str. 2	Str. 3
Paris	110754	72118	66464
Berlin	124773	73400	68796
SF	14679416	11071874	10450375

The strategy exploring the biggest number of labels is the strategy one. The strategy 2 and 3 have approximatively the same number of explored labels. The result of the comparison between the strategies 1 and 2 is that when the number of explored label decreases, the execution time decrease. However the same interpretation cannot be done for the difference found between the strategies 1 and 3 and between the strategies 2 and 3. Even if the strategy 3 explores less labels, its average of execution times are bigger than others, meaning the linear combination of both criteria cannot be used to quickly guide the exploration towards good solutions.

3.2 Comparison of Several $\mathcal{A}$ Values

Only two strategies are used for the numerical experiments of several $\mathcal{A}$ values. Three cases have been tested:

- $\mathcal{A} = \{0;1\}$ $(\mid \mathcal{A} \mid = 2)$
- $\mathcal{A} = \{0;0.5;1\}$ $(\mid \mathcal{A} \mid = 3)$
- $\mathcal{A} = \{0;0.25;0.5;0.75;1\}$ $(\mid \mathcal{A} \mid = 5)$

Table 5 presents the average execution time (in ms) of the first phase for each case. Table 3 shows the average execution time (in ms) of the second phase for each case and each strategy (strategy 1 noted S1 strategy 2 noted S2). The total average execution time (in ms) of LSDPF algorithm is presented in table 4. Finally, the number of explored labels (in thousands) of LSDPF algorithm is given in table 6.

Table 3: Average execution time of second phase (ms) for several values of $\mathcal{A}$ several strategies.

Graph	$\mid\mathcal{A}\mid = 2$		$\mid\mathcal{A}\mid = 3$		$\mid\mathcal{A}\mid = 5$	
	S1	S2	S1	S2	S1	S2
Paris	165	129	154	163	142	156
Berlin	146	91	139	113	110	114
SF	86844	71666	64430	75884	67118	82026

Table 4: Average execution time of LSDPF (ms) for several values of $\mathcal{A}$ and several strategies.

Graph	$\mid\mathcal{A}\mid = 2$		$\mid\mathcal{A}\mid = 3$		$\mid\mathcal{A}\mid = 5$	
	S1	S2	S1	S2	S1	S2
Paris	195	159	193	201	207	221
Berlin	206	151	223	197	257	261
SF	87041	71862	64641	76094	67488	82396

From previous tables 5 to 6, it seems not interesting to use more than 2 α values whatever the strategy 1 or 2. The fact that the execution time increase when the number of α values increases in a given strategy can be explained by the additional consumption of CPU time needed by DPF function. Moreover, the degradation of execution time of the strategy 2 is more significant in comparison with the strategy 1.

3.3 Efficiency of LSDPF

In order to compare our results with benchmarks algorithms solving the BSP - the bounded Label Setting algorithm (blSET) (Raith, 2010) and the pulse algorithm (Duque et al., 2015) - computational experiments have been performed on instances from the 9th DIMACS challenge. Three graphs have been used. The first graph corresponds to New York City. This graph contains 264,346 nodes and 733,846 arcs. The second corresponds to another representation of the San Francisco Bay Area and contains 321,270 nodes and 800,172 arcs. The third corresponds to the state of Florida and contains 1,070,376 nodes and 2,712,798 arcs. Criteria considered for these graphs correspond to the physical distance and transit time between nodes. To fairly compare algorithms, the same pairs of source and target nodes considered in (Duque et al., 2015) have been used for each instance. As (Duque et al., 2015), the 30 instances of each graph has been clustered into the same three equal sized groups, denoted S (small), M (medium) and L (large), based on the number of non-dominated solutions found on the Pareto front.

The table 7 compares the average execution time

Table 5: Average execution time of first phase (ms) for several values of $\mathcal{A}$.

Graph	$\mid\mathcal{A}\mid = 2$	$\mid\mathcal{A}\mid = 3$	$\mid\mathcal{A}\mid = 5$
Paris	30	39	65
Berlin	60	84	147
SF	196	211	370

Table 6: Average number of explored labels (in thousands) for several values of $\mathcal{A}$ several strategies.

Graph	$\mid\mathcal{A}\mid = 2$		$\mid\mathcal{A}\mid = 3$		$\mid\mathcal{A}\mid = 5$	
	S1	S2	S1	S2	S1	S2
Paris	111	72	89	71	78	70
Berlin	125	73	100	72	79	71
SF	14679	11072	12355	11023	11289	10965

of the bLSET algorithm, the Pulse algorithm, and LSDPF algorithm. All times are in milliseconds. The first column indicates the cluster name, composed of the instance name and the letter from the group to which it belongs. Column 2 shows the average number of non-dominated solutions on the the Pareto front. Column 3, 4 and 5 presents the average execution time of the bLSET algorithm, Pulse algorithm and LSDPF algorithm. Values followed by $*$ means that at least one of the instances could not be solved in less than one hour and so the search has been stopped.

Table 7 shows that execution times of bLSET are average bigger than other algorithms. Regarding the Pulse and LSDPF algorithms, the Pulse algorithm is particularly efficient on instances with a small number of solutions unlike other methods. For these instances, LSDPF takes up to 7 times more to find the entire Pareto front but execution times remain reasonable since they are on average lower than 3 seconds. Nevertheless, we can see that LSDPF has the same average order execution times to solve instances whatever the number of solutions which is not the case of the two other algorithms where it appear to have an exponential time progression. Moreover they may not find the entire Pareto front in the case of larger instances (see table 8). To conclude, LSDPF is clearly the faster method to find the whole Pareto front for medium and larger instances.

Finally, the criterion used for the shortest paths search on these instances are distance and time, that are not completely independents criteria. The Pareto front thus contains fewer non-dominated solutions that it is possible to have using more conflicting criteria, such as distance and insecurity for example.

4 CONCLUSION

In this paper, we consider a classical Bi-objective

Table 7: Averaging times (ms) for the bLSET, the Pulse algorithm and the LSDPF on 9th DIMACS challenge graphs.

Cluster	$\|S\|$	bLSET	Pulse	LSDPF
NY-S	34.10	62390	320	2169
NY-M	147.40	301160	52320	3692
NY-L	422.70	881260	1367660*	945
BAY-S	8.80	6780	160	509
BAY-M	49.90	55240	5700	508
BAY-L	171.80	317430	105550	360
FLA-S	14.70	330120	350	2343
FLA-M	94.10	566150*	347910	1189
FLA-L	552.30	2627430*	888590*	2776

Table 8: Number of instances unsolved in less than an hour for bLSET, Pulse algorithm and the LSDPF.

Cluster	bLSET	Pulse	LSDPF
NY-L	0	3	0
FLA-M	1	0	0
FLA-L	6	1	0

Shortest Path Problem (BSP) which considers both distance and insecurity inspired from a cycling context. We proposed an exact algorithm based on label-setting methods and called LSDPF, in order to find all non-dominated solutions of the problem. One of the specificities of LSDPF algorithm is to dynamically update the whole Pareto front during the search allowing to reduce the number of explored labels. Different strategies have been proposed and tested. Numerical experiments on real data sets and on instances of the literature were conducted. Comparison with recent benchmarks algorithms to solve BSP (the bounded Label Setting algorithm (blSET) (Raith, 2010) and the pulse algorithm (Duque et al., 2015)) shows that our method outperform the benchmarks algorithms.

The first perspective of this work is to generalize our methods to several conflictual criteria on same type of instances. Another promising research direction is to use some graph contractions in the case of multi-criteria.

REFERENCES

Bertsekas, D. (1998). Network optimization: Continuous and discrete models.

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

Carlyle, W. and Wood, R. (2005). Near-shortest and k-shortest simple paths. *Networks*, 46:98–109.

Climaco, J. and Martins, E. (1982). A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11:399–404.

C.T. Tung, K. C. (1992). A multicriteria pareto-optimal path algorithm. *European Journal of Operational Research*, 62:203–209.

Demeyer, S. and al. (2013). Speeding up martins' algorithm for multiple objective shortest path problems. *4OR*, 11:323–348.

Dijkstra, E. (1959). A note on two problems in connection with graphs. *Numerische Mathematik*, (1):269–271.

Duque, D., Lozano, L., and Medaglia, A. (2015). An exact method for the biobjective shorstest path problem for large-scale road networks. *European Journal of Operational Research*, (242):788–797.

Guerriero, F. and Musmanno, R. (2001). Label correcting methods to solve multicriteria shortest path problems. *Journal of optimization theory and applications*, 111:589–613.

Hansen, P. (1980). Multiple criteria decision making theory and application. *Lecture Notes in Economics and Mathematical Systems*, 177:109–127.

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

Raith, A. (2010). Speed-up of labelling algorithms for biobjective shortest path problems. pages 313–322.

Raith, A. and Ehrgott, M. (2009). A comparison of solution strategies for biobjective shortest path problems. *Computers & OR*, 36(4):1299–1331.

Sauvanet, G. and Neron, E. (2010). Search for the best compromise solution on multiobjective shortest path problem. *Electronic Notes in Discrete Mathematics*, (36):615–622.

Serafini, P. (1987). Some considerations about computational complexity for multiobjective combinatorial problems. In *Lecture Notes in Economics and Mathematical Systems*, pages 222–232. Springer-Verlag.

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

Tarapata, Z. (2007). Selected multicriteria shortest path problems: An analysis of complexity, models and adaptation of standard algorithms. *International Journal Of Applied Mathematics And Computer Science*, 17:269–287.

Ulungu, E. and Teghem, J. (1994). Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3:83–104.

Ulungu, E. and Teghem, J. (1995). The two phases method : An efficient procedure to solve biobjective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165.