Study of Route Optimization Considering Bottlenecks and Fairness Among Partial Paths

Toshihiro Matsui[1], Marius Silaghi[2], Katsutoshi Hirayama[3], Makoto Yokoo[4] and Hiroshi Matsuo[1]

[1]*Nagoya Institute of Technology, Gokiso-cho Showa-ku Nagoya 466-8555, Japan*

[2]*Florida Institute of Technology, Melbourne FL 32901, U.S.A.*

[3]*Kobe University, 5-1-1 Fukaeminami-machi Higashinada-ku Kobe 658-0022, Japan*

[4]*Kyushu University, 744 Motooka Nishi-ku Fukuoka 819-0395, Japan*

Keywords: Bottleneck, Fairness, Dynamic Programming, Search, A*, Route Optimization.

Abstract: Route optimization is an important problem for single agents and multi-agent systems. In route optimization tasks, the considered challenges generally belong to the family of shortest path problems. Such problems are solved using optimization algorithms, such as the A* algorithm, which is based on tree search and dynamic programming. In several practical cases, cost values should be as evenly minimized for individual parts of paths as possible. These situations are also considered as multi-objective problems for partial paths. Since dynamic programming approaches are employed for the shortest path problems, different types of criteria which can be decomposed with dynamic programming might be applied to the conventional solution methods. For this class of problems, we employ a leximax-based criterion, which considers the bottlenecks and unfairness among the cost values of partial paths. This criterion is based on a similar criterion called leximin for multi-objective maximization problems. It is also generalized for objective vectors which have variable lengths. We address an extension of the conventional A* search algorithm and investigate an issue concerning on-line search algorithms. The influence of the proposed approach is experimentally evaluated.

1 INTRODUCTION

Route optimization is a critical problem for single agents and multi-agent systems. Several tasks are based on the optimization of routes, such as route navigation for drivers, delivery services, and planning for mobile robots. The goal of the route optimization of agents is generally minimization of the total cost on the optimal route. The A* search algorithm is a fundamental path finding method that is based on best-first search and dynamic programming (Hart and Raphael, 1968; Hart and Raphael, 1972). On the other hand, in several practical problems, improving bottlenecks and fairness among the cost values of the individual parts in the optimal path might be an issue. This is also a multi-objective optimization problem where each objective corresponds to an individual part of a path. Several criteria, which select a solution to a multi-objective problem, are known as social welfare or scalarization functions (Sen, 1997; Marler and Arora, 2004). Leximin is a criterion that considers bottlenecks and fairness among the objectives for multi-objective maximization problems (Bouveret and Lemaître, 2009; Greco and Scarcello, 2013). This criterion is based on the dictionary order of vectors whose values are sorted in ascending order. Maximization on the leximin criterion maximizes the minimum objective and improves fairness. This approach has been applied to resource allocation problems with multiple objectives (Dritan and Pioro, 2008). For the minimization problems, similar a criterion where the objective values are sorted in descending order can be applied.

In this study, we focus on a property that a problem which is defined with this criterion can be decomposed into subproblems in ways similar to dynamic programming (Matsui et al., 2014; Matsui et al., 2015). Since the A* search algorithm is based on dynamic programming, we can employ a criterion that resembles leximin. Therefore, in this work we address the route optimization methods that improve the bottlenecks and the fairness of paths based on this type of criterion. We apply a modified *leximax* criterion that resembles leximin to the A* search algorithm. We also investigate the possibility of incremental optimization approaches for the exploration

Matsui, T., Silaghi, M., Hirayama, K., Yokoo, M. and Matsuo, H.
Study of Route Optimization Considering Bottlenecks and Fairness Among Partial Paths.
DOI: 10.5220/0006589000370047
In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART 2018) - Volume 1*, pages 37-47
ISBN: 978-989-758-275-2

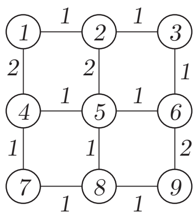

Figure 1: Lattice graph.

of agents in environments. In this investigation, we note that the proposed operations have a characteristic property where the lower bounds of the optimal path might not converge to the optimal values under the equation of the optimal principle. On the other hand, the upper bounds converge to the optimal ones. We address how this property can be mitigated in incremental optimization methods.

The rest of this paper is organized as follows. The next section introduces the backgrounds of this study, including path finding problems, related solution methods, and several concepts about bottlenecks and fairness. Then we present our proposed approach where a modified leximax criterion is applied to the A* search algorithm in Section 3. Section 4 describes approaches for the exploration of agents in environments. The proposed approach is experimentally evaluated in Section 5. Then several discussions and conclusions are shown in Sections 6 and 7.

2 PRELIMINARIES

2.1 Route Optimization Problems

Route optimization problems are generally based on the shortest path problem. We address a fundamental problem, which is defined with a weighted and undirected graph $G = \langle V, E\rangle$. V and E are sets of vertices and edges, respectively. We assume the edges are undirected. For each edge $e_{i,j} \in E$, which is connected to $s_i, s_j \in V$, its cost value $w_{i,j} = \omega(e_{i,j})$ is defined with function $\omega : e_{i,j} \to \mathbb{Z}+$, where $\mathbb{Z}+$ is a set of positive integer values.

Path P is defined as a sequence of vertices $(s_1, s_2, \cdots, s_n) \in V^n$, where edge $e_{i,i+1} \in E$ exists between each pair of vertices $s_i, s_{i+1} \in V$. The cost of path P is evaluated as $\sum_1^{n-1} w_{i,j}$. The shortest path from start node $s_s \in V$ to goal node $s_g \in V$ is defined as the path with the minimum cost value among the paths from s_s to s_g. Such a shortest path is considered the optimal route.

In general settings, the aggregation of the cost values is defined with a summation operator to evaluate the total summation of the cost values in a path. The goal of the route optimization problem is to minimize the cost of the routes for the same pair of start and goal nodes.

In this work, we assume that the cost values take a small number of integer values, such as $\{1,2\}$ or $\{1,\cdots,10\}$. The cost values basically represent levels of unfavorableness.

In the following, we interchangeably use vertices and nodes. In addition, we mainly address lattice graphs, as shown in Fig. 1, for simple discussions of such issues as heuristic distances. Here, the numbers of the nodes correspond to identifiers of them. The numbers for the edges correspond to their cost values.

2.2 Path Finding Algorithm based on Dynamic Programming

A* search (Hart and Raphael, 1968; Hart and Raphael, 1972) is a path finding algorithm based on best-first search and dynamic programming. This algorithm performs a search process from a start node to a goal node. In the process, the estimated cost values (distances) of the optimal path are updated for the visited nodes by dynamic programming. When the goal node is found, the optimal path is determined from the stored information. In the exploration process, a search method extends the nodes based on their estimated cost values.

For each node $s_i \in V$, its estimated cost value of optimal path $f(s_i)$ is defined:

$$f(s_i) = g(s_i) + h(s_i). \quad (1)$$

Here $g(s_i)$ is the estimated cost value from the start node to s_i, and $h(s_i)$ is the estimated cost value from s_i to the goal node. While $g(s_i)$ is updated based on dynamic programming in the exploration process, $h(s_i)$ is given as a heuristic value [1]. $h(s_i)$ is admissible when it is a correct lower bound value. For lattice graphs, $h(s_i)$ can be defined based on several distance functions, such as the Manhattan and Euclid distances, considering the lower bound cost values of the edges.

The algorithm consists of two phases based on a dynamic programming manner. In the first phase, an optimal and complete version of best-first search is performed from the start node s_s updating each $f(s_i)$. After the goal node s_g is extracted, the second phase is performed from the goal node to the start node to compose the shortest path. See (Hart and Raphael, 1968; Hart and Raphael, 1972; Russell and Norvig, 2003) for details of the algorithm.

[1] The actual algorithm maintains $f(s_i)$ as well as $g(s_i)$.

2.3 Real-time Search Algorithm

Real-time search algorithms are also path finding methods based on search methods and dynamic programming. Here the solution methods are designed so that an agent employs exploration and exploitation in a path finding. Initially, the agent is placed at the start node with estimated cost values (e.g. $h(s_i) = 0$ for all $s_i \in V$). It explores in the environment and learns the estimated optimal cost value of each node. Based on these estimated cost values, the agent searches for the goal node. After the agent arrives at the goal node, it starts its next tour with the updated values.

An important point of this approach is that dynamic programming is assumed to be the agent's learning process. With an appropriate strategy and learning rule, the estimated optimal cost value of each node is improved by the number of tours. Learning Real-Time A* (LRTA*) is such an algorithm. With it, an agent performs the following steps at each node s_i in a tour:

1. For all nodes s_j adjacent to s_i, the agent evaluates estimated cost value $f(s_j) = w_{i,j} + h(s_j)$.
2. The agent updates the estimated cost value so that $h(s_i) \leftarrow \min_{s_j} f(s_j)$.
3. It moves to the next node so that $s_i \leftarrow \arg\min_{s_j} f(s_j)$.

Note that in the above rule, the $h(s_i)$ value is both the estimation value to be optimized and the penalty value that is employed to escape from cyclic paths. See (Barto et al., 1995) for details of this algorithm.

2.4 Bottlenecks and Fairness Among Edges

Next we investigate the improvement of bottlenecks and fairness among edges. In the example in Fig. 1, let the start node be $s_s = 1$ and the goal node be $s_g = 9$. In the minimization of conventional total cost values, one optimal path is $(1,2,3,6,9)$ and its cost is 5. On the other hand, for minimization where the edges of the maximal cost values are reduced, one optimal path is $(1,2,3,6,5,8,9)$, and its cost is 6. Note that there are no edges with cost value 2 on this path. Here the goal of the problem is not only to reduce the number of edges with the maximum cost values but also to reduce the total cost value by improving the fairness among edges, if possible. Such paths might be investigated when an appropriate route must avoid highly unsatisfactory specific residents and extra loads of bottleneck facilities.

2.5 Multi-objective Optimization Problems

In the class of multi-objective optimization problems, multiple objectives are simultaneously optimized. The above route optimization problem with bottlenecks and fairness is also a multi-objective optimization problem. Here we consider the following multi-objective optimization problem: MOP.

Definition 1 (MOP)**.** *MOP is defined with $\langle X, D, F\rangle$. X is a set of variables, D is a set of domains of variables, and F is a set of objective functions. Variable $x_i \in X$ takes value from finite and discrete set $D_i \in D$. For set of variables $X_i \subseteq X$, function $f_i \in F$ is defined as $f_i(x_{i,1}, \cdots, x_{i,k}) : D_{i,1} \times \cdots \times D_{i,k} \rightarrow \mathbb{N}$, where $x_{i,1}, \cdots, x_{i,k} \in X_i$. $f_i(x_{i,1}, \cdots, x_{i,k})$ is simply denoted by $f_i(X_i)$. The goal of the problem is to simultaneously optimize the objective functions under a criterion.*

A combination of the values of the objective functions is represented as an objective vector.

Definition 2 (Objective vector)**.** *Objective vector* $\mathbf{v}$ *is defined as $[v_1, \cdots, v_K]$. For assignment $\mathcal{A}$ to the variables in X_j, v_j is defined as $v_j = f_j(\mathcal{A}_{\downarrow X_j})$.*

Here the ideal goal is to maximize all the values of the objective functions. However, in general cases, the goal cannot be achieved since there are trade-offs between the objectives. Therefore, based on the Pareto dominance between objective vectors, one of the Pareto optimal solutions is selected (Sen, 1997; Marler and Arora, 2004).

2.6 Leximin

Since there are many Pareto optimal solutions in general cases, several social welfare criteria and scalarization functions are employed to select a solution (Sen, 1997; Marler and Arora, 2004).

Summation $\sum_{j=1}^{K} f_j(X_j)$, which is found in conventional social welfare, considers the efficiency of the objectives. While maximization on the summation achieves Pareto optimality, fairness among the objectives is ignored. *Maximin* maximizes minimum objective value $\min_{j=1}^{K} f_j(X_j)$. Even though this improves the worst case value among the objectives, the solution is not Pareto optimal. To ensure Pareto optimality, such additional tiebreaker criteria as summation are necessary. Moreover, since only the minimum objective value is improved, other objective values are not distinguished.

Leximin is defined as the dictionary order on objective vectors whose values are sorted in ascending order (Bouveret and Lemaître, 2009; Greco and Scarcello, 2013; Matsui et al., 2014; Matsui et al., 2015).

Definition 3 (Sorted objective vector)**.** *The values of sorted objective vector* $\mathbf{v}$ *are sorted in ascending order.*

Definition 4 (Leximin)**.** *Let* $\mathbf{v} = [v_1, \cdots, v_K]$ *and* $\mathbf{v}' = [v'_1, \cdots, v'_K]$ *denote the sorted objective vectors whose length is* K*. The order relation, denoted with* $\prec_{leximin}$*, is defined as follows.* $\mathbf{v} \prec_{leximin} \mathbf{v}'$ *if and only if* $\exists t, \forall t' < t, v_{t'} = v'_{t'} \wedge v_t < v'_t$.

Leximin is a criterion that repeats the comparison between the minimum values in the vectors. Since maximization on leximin is a subset of maximin, it improves the worst case values. In addition, this maximization relatively improves the fairness and ensures Pareto optimality.

The addition of two sorted objective vectors is defined with concatenation and resorting.

Definition 5 (Addition of sorted objective vectors)**.** *Let* $\mathbf{v}$ *and* $\mathbf{v}'$ *denote vectors* $[v_1, \cdots, v_K]$ *and* $[v'_1, \cdots, v'_{K'}]$*, respectively. The addition of two vectors,* $\mathbf{v} \oplus \mathbf{v}'$*, is represented as* $\mathbf{v}'' = [v''_1, \cdots v''_{K+K'}]$*, where* $\mathbf{v}''$ *consists of all the values in* $\mathbf{v}$ *and* $\mathbf{v}'$*. In addition, the values in* $\mathbf{v}''$ *are sorted in ascending order.*

For the addition of sorted objective vectors, the following invariance exists (Matsui et al., 2014).

Proposition 1 (Invariance of leximin on addition)**.** *Let* $\mathbf{v}$ *and* $\mathbf{v}'$ *denote sorted objective vectors of the same length. In addition,* $\mathbf{v}''$ *denotes another sorted objective vector. If* $\mathbf{v} \prec_{leximin} \mathbf{v}'$*, then* $\mathbf{v} \oplus \mathbf{v}'' \prec_{leximin} \mathbf{v}' \oplus \mathbf{v}''$.

Based on this invariance, dynamic programming can be applied to solve optimization problems with the leximin criterion (Matsui et al., 2014; Matsui et al., 2015). However, we assume that the original problem is decomposed into subproblems with objective vectors of the same length.

Moreover, a sorted objective vector can be represented as a vector of the sorted pairs of an objective value and the count of the value (Matsui et al., 2014). This representation corresponds to run-length encoding and a sorted histogram. The comparison and the addition of two sorted objective vectors can be directly performed on this representation.

2.7 Theil Index

As addressed above, several criteria, including summation and leximin, ensure Pareto optimality. Pareto optimality is important in situations with selfish members; however, other measurements of inequality are also critical. To evaluate the paths shown in later sections, we employ the Theil index, a well-known measurement of inequality.

Definition 6 (Theil Index)**.** *For n objectives, Theil index* T *is defined as*

$$T = \frac{1}{n}\sum_i \frac{v_i}{\overline{v}} \log \frac{v_i}{\overline{v}} \tag{2}$$

where v_i *is the utility or the cost value of an objective and* $\overline{v}$ *is the mean utility value for all the objectives.*

The Theil index takes a value in $[0, \log n]$. When all utilities or cost values are identical, the Theil index value is zero. Inequalities on different number of members can be compared using it. Note that the minimization on the leximin criterion does not assure the decrement of the Theil index value, since it is basically a sequence of improvements of the worst value.

3 ROUTE OPTIMIZATION CONSIDERING BOTTLENECKS AND FAIRNESS

We address the route optimization problems that consider bottlenecks and fairness among individual edges. As shown in the previous section, the optimization problem with the leximin criterion is decomposed with dynamic programming. Therefore, we investigate the approach that replaces the aggregation of the cost values in the A* algorithm from the summation to a leximin-like criterion. Assuming the new criterion for this optimization, the problem is redefined as follows.

The *shortest* path finding problem is defined with a weighted and undirected graph $G = \langle V, E \rangle$. The sets V, E, weight values of edges, and Path P are defined similar to the original definition. The cost of path P is aggregated as a objective vector instead of the summation. Then the cost is compared based on a criterion which is similar to the leximin, while it is designed for minimization problems and the variable length of vectors. The shortest path from start node $s_s \in V$ to goal node $s_g \in V$ is defined as the path with the minimum cost value based on this criterion. Such a shortest path is considered the optimal route.

The following two modifications must be addressed to employ the approach for maximization on the leximin criterion.

- The maximization on leximin is replaced by the minimization on *leximax*.
- The operations are extended for the subproblems of different lengths of vectors.

3.1 Minimization on Leximax

For minimization problems, maximization on leximin is replaced by the minimization of leximax. Although leximax is similarly defined as leximin, the ordering of the values in the objective vectors is inverted.

Definition 7 (Descending sorted objective vector)**.** *The values of a descending sorted objective vector are sorted in descending order.*

Definition 8 (Leximax)**.** *Let* $\mathbf{v} = [v_1, \cdots, v_K]$ *and* $\mathbf{v}' = [v'_1, \cdots, v'_K]$ *denote descending objective vectors whose lengths are* K*. The order relation, denoted with* $\prec_{leximax}$*, is defined as follows.* $\mathbf{v} \prec_{leximax} \mathbf{v}'$ *if and only if* $\exists t, \forall t' < t, v_{t'} = v'_{t'} \land v_t < v'_t$.

With this modification, the worst case values are inverted to maximum cost values. Thus the objective of the problem is also inverted to the minimization of the aggregated objective values. While the addition of two descending sorted objective vectors is similarly defined as the leximin, the ordering of the values in each concatenated vector is the opposite.

3.2 Comparison of Vectors with Different Lengths

In route optimization problems, when the path lengths are different, the lengths of the corresponding objective vectors are also different. Therefore, we employ the variable-length leximax, *vleximax*, whose definition is extended for objective vectors of different lengths.

Definition 9 (Vleximax)**.** *Let* $\mathbf{v} = [v_1, \cdots, v_K]$ *and* $\mathbf{v}' = [v'_1, \cdots, v'_{K'}]$ *denote descending sorted objective vectors whose lengths are* K *and* K'*, respectively. For* $K = K'$*,* $\prec_{vleximax}$ *is the same as* $\prec_{leximax}$*. In other cases, zero values are appended to one of the vectors so that the both vectors have the same number of values. Then the vectors are compared based on* $\prec_{leximax}$.

Intuitively, this comparison is based on two modified vectors which have the same sufficient length by padding blanks with zeros. Consider the two modified vectors of infinite length assuming the cost of zero for each unused edge, which can contain the extra edges outside of the system. Since the comparison of leximax is based on tie-breaks from the beginning of the both vectors, the redundant parts of zeros in both the vectors can be ignored.

In the A* search algorithm for comparing two paths based on vleximax, the corresponding descending sorted objective vector should be appropriately aggregated.

3.3 Heuristic Distance Function

In the A* search algorithm, estimated cost value $h(v_i)$, which is given by a heuristic distance function, should be a lower bound value that does not exceed the optimal cost value. For example, in the case of lattice graphs, such a heuristic function can be defined with the lower bound cost value for all the edges and the Manhattan distance. For minimization on the summation, a heuristic value is the product of the Manhattan distance and the lower bound cost value.

For minimization on the vleximax, such a heuristic value is an objective vector that consists of duplicates of the lower bound cost values, where the vector length is identical to the Manhattan distance.

3.4 Correctness and Complexity of Solution Method

When the objective vector of estimated cost $h(v_i)$ of a heuristic distance function is a correct lower bound one, the A* search algorithm selects one of the optimal paths. For lattice graphs, the Manhattan distance from a node to the goal node is the minimum length of the possible vector. Therefore, the objective vector whose length is the same as the Manhattan distance and whose values are duplicates of the lowest cost value for all the edges is a lower bound for the remaining optimal path. On the other hand, in general cases, designing efficient heuristic functions might not be easy.

In addition, for minimization on summation, heuristic cost value $h(v_i)$ can be simply defined as zero. Similarly, for minimization on leximax, an empty vector can be employed as $h(v_i)$.

The comparison of two vectors of different lengths based on vleximin activates a tiebreaker, where the shorter vector is selected when the two vectors are identical on their parts of the same length. In this case, fewer edges are preferred. Since the estimation cost is a lower bound, no incorrect path can be selected as the optimal path, even if the lengths of the descending sorted objective vectors are different. Therefore, the solution method returns one of the optimal paths.

Even though the overhead of the computation related to vleximax is significantly larger than that of the summation and the comparison on scalar values, it is polynomial with the length of each vector. When the sorted objective vector is represented as run-length encoding or a histogram, the space complexity of each vector is $O(n)$ for n types of objective values. If an array is employed for this vector representation, the addition of two vectors increments the

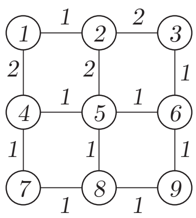

Figure 2: Lattice graph with *walls*.

count values whose complexity is $O(n)$. The complexity for the comparison of two vectors is also $O(n)$.

4 INCREMENTAL OPTIMIZATION

Next we focus on how the real-time search algorithm can be generalized with the leximax criterion. Unfortunately, this is impossible due to a problematical monotonicity on cyclic paths.

Consider the case shown in Fig. 2, where the agents start from node 1. For the nodes adjacent to node 1, $h(2)+w_{1,2}=[\,]+[1]=[1]$ and $h(4)+w_{1,4}=[\,]+[2]=[2]$. With the vleximax and the rules based on the LRTA* shown in Section 2.3, the agent moves to node 2 and updates $h(1)$ to $[1]$. Then for the nodes adjacent to node 2, $h(1)+w_{1,2}=[1]+[1]=[1,1]$, $h(3)+w_{2,3}=[\,]+[2]=[2]$, and $h(5)+w_{2,5}=[\,]+[2]=[2]$. Therefore, the agent returns to node 1 and updates $h(2)$ to $[1,1]$. In the third step, for the nodes adjacent to node 1, $h(2)+w_{1,2}=[1,1]+[1]=[1,1,1]$ and $h(4)+w_{1,4}=[\,]+[2]=[2]$. Therefore, the agent returns to node 2 again and repeats this round-trip forever to add cost value 1 to $h(1)$ and $h(2)$.

The above example reveals the necessity of other approaches for exploration in the case of sorted objective vectors with variable lengths when there can be cyclic paths. Such cyclic paths can be detected with a threshold length. Then some such incorrect vectors can be replaced by appropriate vectors that break the cyclic movements. However, such an approach might be problematic, since the invariance of vleximin does not hold and may affect the correctness of the dynamic programming.

4.1 Episode-based Approach

Here we address more safe approaches with a relatively direct extension of conventional search algorithms. Since the dynamic programming is correct, we employ episode-based learning, where the learning phases are separated from the exploration phase. This approach is also called off-line learning.

Assume that a complete path between the start and goal nodes was obtained from an exploration phase. Cyclic paths are allowed to increase the learning opportunities.

Then the path is scanned from the goal node to the initial start node by updating corresponding estimated values $h(s_k)$ except the goal node. Note that here s_k denotes the k^{th} value from the initial start node on a path:

1. The agent evaluates $f(s_{k+1})=w_{i,j}+h(s_{k+1})$, where $w_{i,j}$ corresponds to edge $e_{i,j}$ between s_k and s_{k+1}.
2. If $h(s_k)$ has not been updated yet, it is updated by $f(s_{k+1})$. Otherwise, it is updated by $\min(f(s_{k+1}),h(s_k))$.

In the example of Fig. 2, assume that an episode of nodes $(1,2,3,6,5,2,3,6,9)$ has been performed in the initial trial. Since the node 9 is the goal node, $h(9)$ holds empty vector $[]$. Then $h(6)$ is updated by $f(9)=[]+[1]=[1]$. Similarly, for their previous part of nodes $(5,2,3)$, $h(3)=[1,1]$, $h(2)=[1,1,2]$, $h(5)=[1,1,2,2]$ are updated. However, for their previous node 6, $h(6)$ holds its vector by $\min(f(5),h(6))=\min([1,1,1,2,2],[1])=[1]$. $h(3)=[1,1]$ and $h(2)=[1,1,2]$ are also unchanged for the previous part of nodes $(3,2)$. Finally, $h(1)$ is updated by $[1,1,1,2]$.

The above $h(s_k)$ is the upper bound of the optimal cost value from s_k to the goal node, since it is updated by the propagation from the goal node. When the agent's explorations are sufficient, $h(s_k)$ converges to the optimal value, since the algorithm exactly performs partial updates of the dynamic programming.

4.2 Boundaries of Paths

While the above episode-based approach needs complete paths to the goal nodes, it converges with appropriate exploration strategies. The other problem of the above simple update rule is that it does not employ the information of neighborhood nodes which are not on the path. Also, the algorithm cannot evaluate the lower bound cost values which can be employed by best-first strategies.

Here we address the *lower bound* of optimal cost value $\underline{h}(s_i)$ and the upper bound value $h(s_i)$. The boundaries $\underline{h}(s_i)$ and $h(s_i)$ of the estimated cost values are initialized as follows:

1. Except for the goal node, $\underline{h}(s_i)$ and $h(s_i)$ are initialized to $[\,]$ and $[\top\cdots\top]$, respectively, where $\top$ denotes the maximum cost value. $h(s_i)$ must contain a sufficient number of duplicates of the maximum cost value to exceed the other objective vectors in the manner of vleximin.

2. On the other hand, for the goal node, the initial value of $\underline{h}(s_i)$ and $h(s_i)$ is $[\,]$.

Except for the goal node, node s_i updates its $\underline{h}(s_i)$ and $h(s_i)$ as follows:

$$\underline{h}(s_i) \leftarrow \max(\underline{h}(s_i), \min_{s_j}(w_{i,j} + \underline{h}(s_j))) \tag{3}$$

$$h(s_i) \leftarrow \min(h(s_i), \min_{s_j}(w_{i,j} + h(s_i))), \tag{4}$$

where s_j denotes all the nodes adjacent to s_i. Here we update $\underline{h}(s_i)$ and $h(s_i)$ at the same time.

When a path is obtained from an exploration phase, the following operations are performed.

1. The path is scanned from the goal node to the first node.
2. Except for the goal node, each node s_i on the path updates its $\underline{h}(s_i)$ and $h(s_i)$.

With the above boundaries the exploration and learning phases perform parts of dynamic programming. We consider that the boundaries of node s_i have converged when $\underline{h}(s_i) \geq h(s_i)$. In the case of the conventional summation, the boundaries eventually converge. However, in the case of the vleximax, the convergence is not guaranteed. Consider the example shown in Fig. 2 again. Assuming that $\underline{h}(3)$, $\underline{h}(4)$, and $\underline{h}(5)$ take zero, from the initial state, $\underline{h}(1)$ and $\underline{h}(2)$ increase such that $[\,],[1],[1,1],[1,1,1],\cdots,[1,\cdots,1]$ by turns. These vectors never overcome a vector $[2]$. Similar cases occur in actual propagations even if an episode does not contain cyclic paths. For a non cyclic path $(1,2,3,6,9)$, the lower bounds are updated for the sequence of nodes 1 and 2 in reverse order. If other paths of different episodes contain these two nodes, they are repeatedly updated in a way that is similar to the above manner.

This situation resembles LRTA* which is not easily generalized with vleximax. In this problem, the upper bounds of cost vectors still follow the principle of optimality. On the other hand, the lower bounds might not converge to the optimal vectors, while their lengths monotonically increase. This property also resembles that of negative cyclic paths. The A* algorithm dose not affected by this property, since it is based on the propagation of cost values from the node of zero cost. In addition, the algorithm does not assume cyclic paths. However, LRTA* is affected by the property, since its behavior is completely based on the lower bounds. Therefore, the on-line search might be caught by cyclic paths.

To mitigate such situations, we revise the *corrupted* lower bound as follows.

1. When the length of $\underline{h}(s_i)$ exceeds the number of edges, $\underline{h}(s_i)$ is replaced by $\underline{f}(s_j)$ which is the second smallest value: $\underline{f}(s_j) = \mathrm{secondmin}_{s_k}(w_{i,k} + \underline{h}(s_k))$. If such all values are corrupted, it means that there is no information to fix the boundaries. Therefore, the agent do nothing and wait for future propagations from outside.
2. This modification may cause the situation of $\underline{h}(s_i) \not\preceq h(s_i)$ as a result of propagations. In this case, $\underline{h}(s_i)$ is replaced by $h(s_i)$.

This approach is not exact but based on the immediate convergence of upper bounds. Since we employ an off-line learning which updates both upper and lower bounds simultaneously from the node of zero cost (i.e. the goal node), the convergence of upper bounds will be faster than the lower bounds in general cases. In addition, the revision of a *broken* lower bound vector is performed when its length exceeds a threshold. It also delays the convergence of the lower bounds. When these assumptions are sufficiently satisfied, it is expected that the solution will resemble the one of A*.

4.3 A Heuristic Exploration

The exploration strategies should cover all solutions with boundaries. Even though arbitrary exploration strategies can be employed, we employ a heuristic exploration strategy as follows. We added the following information to each node s_i.

- the counter $vstcnt_i$ of visits to the node in each tour. The counter is reset to zero in the beginning of each exploration process.
- the last visit time $lasttime_i$ to the node. This information is stored through the optimization process. We employ a logical time which is incremented after it is stored to a $lastime_i$. In the initial state, all $lasttime_i$ are set to zero.

Similarly, the following information is added to each edge.

- the counter $selcnt_{i,j}$ of selection of the edge $e_{i,j}$ in each tour. The counter is reset to zero in the beginning of each exploration process.

With the above information, the following rules are applied to an agent on s_i adjacent to nodes s_j.

1. If s_j is the goal node, s_j has the first priority.
2. For s_j and an edge $e_{i,j}$, if $\underline{h}(s_j) \neq h(s_j)$ and $selcnt_{i,j} = 0$, s_j has the second priority. This rule assures to evaluate unexplored edges even if those cost values are relatively high.
3. The node s_j who has a smaller $vstcnt_i$ has the third priority. With this rule, the agent will avoid cyclic paths if possible.

Table 1: Solution qualities: lattice graph of 10×10 nodes.

Cost	Alg.	Solution quality				
		sum.	min.	max.	len.	theil
$[1,2]$	sum.	20.7	1	2	18	0.039
	lxm.	21.9	1	2	19.6	0.032
$[1,5]$	sum.	34.3	1	4	18	0.135
	lxm.	41.3	1	3.4	22.4	0.095
$[1,10]$	sum.	58.6	1	7.7	18.4	0.223
	lxm.	74.4	1	6.6	23.2	0.147

Table 2: Solution qualities: Lattice graph of 100×100 nodes.

Cost	Alg.	Solution quality				
		sum.	min.	max.	len.	theil
$[1,2]$	sum.	213.2	1	2	198	0.025
	lxm.	286.9	1	2	282.8	0.006
$[1,5]$	sum.	346.1	1	5	199.6	0.132
	lxm.	446.7	1	3.6	275.6	0.085
$[1,10]$	sum.	580.8	1	9.5	202.8	0.215
	lxm.	960.3	1	6.8	350	0.128

Table 3: Computational cost: lattice graph of 100×100 nodes.

cost	alg.	iter.	num. of opn. nodes	exec. time [s]
$[1,2]$	sum.	7217	7537	0.022
	lxm.	8359	8883	0.070
$[1,5]$	sum.	9989	9996	0.020
	lxm.	9497	9923	0.078
$[1,10]$	sum.	9996	9999	0.019
	lxm.	8468	9182	0.172

4. When $vstcnt_i$ are identical, the node s_j who has a lower $\underline{h}(s_j)$ has the fourth priority. This is the best-first strategy, which is guided by the above rules.

5. When $vstcnt_i$ and $\underline{h}(s_j)$ are are identical, respectively, then the node s_j with older $lasttime_i$ has the fifth priority. With this rule, ties will be explored deterministically.

Due to the above boundaries and the heuristic exploration, the resulting solution method might be an inexact method; however they fit particular intuitions based on our experience. As the first study, we experimentally employ the above approach assuming simple graphs. The necessity of such a guided approach for the vleximax criterion reveals the difficulty of designing an on-line optimization algorithm for this class of problems.

Table 4: Solution qualities: lattice graph of 10×10 nodes (start nodes in the middle).

cost	alg.	solution quality				
		sum.	min.	max.	len.	theil
$[1,2]$	sum.	11.8	1	2	10	0.045
	lxm.	12.5	1	1.8	11	0.037
$[1,5]$	sum.	20	1	3.7	10	0.133
	lxm.	26.4	1	3.4	14.2	0.096
$[1,10]$	sum.	34.5	1.2	7.1	10.4	0.199
	lxm.	50.6	1.1	6.3	15.4	0.134

Table 5: Solution qualities: Lattice graph of 100×100 nodes (start nodes in the middle).

cost	alg.	solution quality				
		sum.	min.	max.	len.	theil
$[1,2]$	sum.	108.5	1	2	100	0.027
	lxm.	149.5	1	1.8	147	0.006
$[1,5]$	sum.	178.5	1	4.9	100.8	0.132
	lxm.	223.2	1	3.4	135	0.087
$[1,10]$	sum.	301	1	9	103.2	0.213
	lxm.	592.1	1	6.3	214.8	0.130

Table 6: Computational cost: lattice graph of 100×100 nodes (start nodes in the middle).

cost	alg.	iter.	num. of opn. nodes	exec. time [s]
$[1,2]$	sum.	1993	2182	0.004
	lxm.	7013	7371	0.066
$[1,5]$	sum.	4702	4860	0.009
	lxm.	9214	9702	0.105
$[1,10]$	sum.	7023	7217	0.016
	lxm.	7759	8402	0.175

5 EVALUATION

We experimentally evaluated the proposed approach. First, we evaluated the modified A* search algorithm. The example problems are based on lattice graphs that consist of 10×10 and 100×100 nodes. Start node s_s and goal node s_g are the left-top and right-bottom nodes. Each edge has an integer cost value in $[1,2]$, $[1,5]$, or $[1,10]$. The cost value is randomly set based on uniform distribution. Ten problem instances are averaged for each set of parameters. We experimentally compared the conventional method based on the summation 'sum.' and the proposed method based on leximax (vleximax) 'lxm.'. The experiment was performed on a computer with a Core i7-3930K CPU (3.20 GHz), 16-GB memory, Linux 2.6.32, and g++ (GCC) 4.4.7.

Tables 1 and 2 show the solution qualities. The cost values in each solution are evaluated with the summation, the minimum value, the maximum value,

Table 7: Solution qualities: lattice graph of 10×10 nodes (lxm) (A* and learning).

cost	alg.	solution quality				
		sum.	min.	max.	len.	theil
$[1,2]$	lxm. A*	21.9	1	2	19.6	0.032
	lxm. lrn.	21.9	1	2	19.6	0.032
$[1,5]$	lxm. A*	41.3	1	3.4	22.4	0.095
	lxm. lrn.	41.3	1	3.4	22.4	0.095
$[1,10]$	lxm. A*	74.4	1	6.6	23.2	0.147
	lxm. lrn.	74.4	1	6.6	23.2	0.147

Table 8: Solution qualities: lattice graph of 20×20 nodes (lxm) (A* and learning).

cost	alg.	solution quality				
		sum.	min.	max.	len.	theil
$[1,2]$	lxm. A*	46.8	1	1.9	44.8	0.016
	lxm. lrn.	46.8	1	1.9	44.8	0.016
$[1,5]$	lxm. A*	77.4	1	3.4	45.4	0.102
	lxm. lrn.	77.4	1	3.4	45.4	0.102
$[1,10]$	lxm. A*	145.7	1	6.6	50.6	0.145
	lxm. lrn.	145.7	1	6.6	50.6	0.145

the number of values (len.), and the Theil index (theil). The Theil index is a criterion of unfairness, where smaller values represent less unfairness. The results show that the solution method 'lxm.' reduced bottlenecks and unfairness with trade-offs between them and the summation. It also reduced the Theil index on average. In addition, the maximum cost value was reduced whenever possible.

Table 3 shows the computational cost. We evaluated the computational cost as the number of operations for the nodes (iter.), the number of extended nodes (num. of opn. nodes), and the execution time. The result reveals that the heuristic cost function based on the Manhattan distance on the grids is not very efficient. While most of the nodes were extended, there were several opportunities for pruning, even for the solution method 'lxm.' The execution time of 'lxm.' significantly exceeded that of 'sum.' due to the operations on the sorted objective vectors and vleximax. But overhead might be allowed in relatively small problems.

With the same set of graphs, we evaluated the cases where a start node s_s is almost in the center of a graph, while a goal node is the right-bottom node. Tables 4 and 5 show the solution qualities. The results resemble the cases of the previous setting. Similarly, Table 6 shows the computational cost. In this result, the number of extracted nodes is relatively less than that of the previous setting. It is considered as the effect of the heuristic distance function of the A* algorithm. On the other hand, the reduction for 'lxm.' is relatively small. This reveals the difficulty of designing heuristic distance functions for the criterion.

We also evaluated the optimization methods for the exploration agents. Here 10×10 and 20×20 lattice graphs with left-top start nodes s_s and right-bottom goal nodes s_g are employed. The A* algorithm based on vleximax ('lxm. A*') and the incremental solution method ('lxm. lrn.') were evaluated. With preliminary experiments, we set appropriate parameters so that the incremental solution method obtains episodes and the solution quality converges. Tables 7 and 8 shows the solution quality. Since both methods obtain similar results, it is considered that the heuristic approaches of the incremental solution method are relatively reasonable.

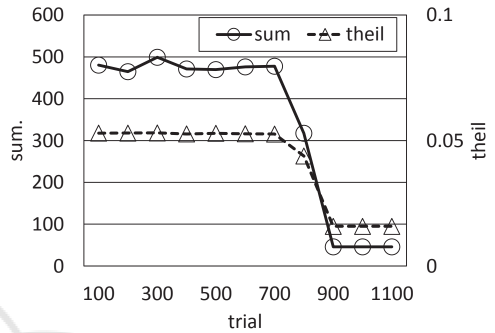

Figure 3: Incremental optimization with vleximax: 20×20 nodes, $w_{i,j} = [1,2]$.

Figures 3-5 show the learning progress of the incremental solution methods. Each graph shows the anytime curves of the summation and the Theil index for an actual result. The samples are averaged for every 100 trials. Note that the results can vary for instances, since we employed a relatively exhaustive exploration based on best-first search. The results show that the summation and the Theil index were gradually improved and converged. In addition, the Theil index was relatively small in early trials when the range of cost values was also narrow.

6 DISCUSSION

Since maximization on (v)leximax improves the worst case values, the summation of the cost values is increased as a trade-off. When the range of the cost values is relatively large, the trade-off is emphasized, and hence the increment of the summation of the cost values grows. The length of the optimal vector, namely the number of edges in the optima path, also grows. Additional approaches, such as limitations on the ranges of the cost values or controls of

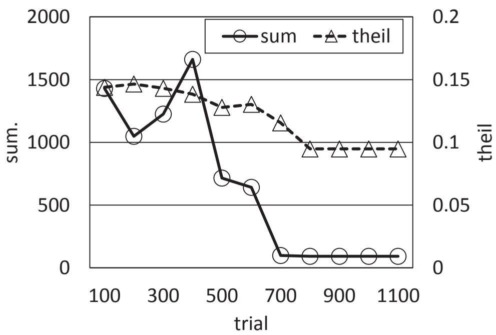

Figure 4: Incremental optimization with vleximax: 20×20 nodes, $w_{i,j} = [1,5]$.

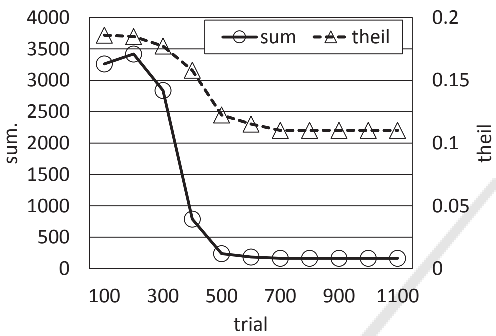

Figure 5: Incremental optimization with vleximax: 20×20 nodes, $w_{i,j} = [1,10]$.

criteria, are necessary to reduce the trade-offs. On the other hand, the proposed method's solution can provide an analysis based on a criterion that addressed bottlenecks and fairness.

In this work, we assumed lattice graphs and employed a distance function based on the Manhattan distance, which relays the possible minimum length of the vectors. For other topologies of the graphs and more efficient estimations, additional considerations are necessary.

In general, most criteria that strictly address fairness cannot be easily decomposed into parts based on dynamic programming. In this study, we focused on how leximin/leximax-based criteria can be applied to dynamic programming methods for path finding problems. Even though the aggregation and the comparison of the criteria can almost be directly applied, the exploration process needs other methods to avoid incorrect results due to the problematical monotonicity of lower bound vectors on cyclic paths.

In addition, leximin/leximax is different from summation in several aspects. For example, when the two neighboring edges are aggregated into an edge, the resulting edge can be related to the aggregation of the two original cost values. While the resulting cost value is still a scalar value for the summation, a vector of two values is necessary to maintain the original information for the leximin/leximax. Such generalization needs more investigation.

Since this criterion is simply defined without parameters, the trade-offs among bottleneck, fairness and effectiveness are fixed. Several modifications or different criteria that can be decomposed with dynamic programming will be necessary to maintain the trade-offs.

7 CONCLUSION

We addressed route optimization methods that consider bottlenecks and fairness on optimal paths using maximization on leximax criteria. The experimental results shows that our proposed approach reduced the cases of the worst cost values and relatively improved the fairness in the optimal path. Future work will improve the solution methods including better heuristic functions and investigations about similar criteria with more appropriate tunings of trade-offs between efficiency and fairness. The opportunities of on-line learning and reinforcement learning will also be interesting issues.

ACKNOWLEDGEMENTS

This work was supported in part by JSPS KAKENHI Grant Number JP16K00301.

REFERENCES

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138.

Bouveret, S. and Lemaître, M. (2009). Computing leximin-optimal solutions in constraint networks. *Artificial Intelligence*, 173(2):343–364.

Dritan, N. and Pioro, M. (2008). Max-min fairness and its applications to routing and load-balancing in communication networks - a tutorial. *IEEE Communications Surveys and Tutorials*, 10(4):5–17.

Greco, G. and Scarcello, F. (2013). Constraint satisfaction and fair multi-objective optimization problems: Foundations, complexity, and islands of tractability. In *Proc. 23rd International Joint Conference on Artificial Intelligence*, pages 545–551.

Hart, P., N. N. and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Science and Cybernetics*, 4(2):100–107.

Hart, P., N. N. and Raphael, B. (1972). Correction to 'a formal basis for the heuristic determination of minimum-cost paths'. *SIGART Newsletter*, (37):28–29.

Marler, R. T. and Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26:369–395.

Matsui, T., Silaghi, M., Hirayama, K., Yokoo, M., and Matsuo, H. (2014). Leximin multiple objective optimization for preferences of agents. In *17th International Conference on Principles and Practice of Multi-Agent Systems*, pages 423–438.

Matsui, T., Silaghi, M., Okimoto, T., Hirayama, K., Yokoo, M., and Matsuo, H. (2015). Leximin asymmetric multiple objective DCOP on factor graph. In *18th International Conference on Principles and Practice of Multi-Agent Systems*, pages 134–151.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall.

Sen, A. K. (1997). *Choice, Welfare and Measurement*. Harvard University Press.