

Investigation of MAPF Problem Considering Fairness and Worst Case

Toshihiro Matsui^a

Nagoya Institute of Technology, Gokiso-cho Showa-ku Nagoya Aichi 466-8555, Japan

Keywords: Multiagent Pathfinding Problem, Leximin/Leximax, Fairness, Worst Case.

Abstract: We investigate multiagent pathfinding problems that improve fairness and the worst case among multiple objective values for individual agents or facilities. Multiagent pathfinding (MAPF) problems have been widely studied as a fundamental class of problem in multiagent systems. A common objective to be optimized in MAPF problem settings is the total cost value of the moves and actions for all agents. Another optimization criterion is the makespan, which is equivalent to the maximum cost value for all agents in a single instance of MAPF problems. As one direction of extended MAPF problems, multiple-objective problems have been studied. In general, multiple objectives represent different types of characteristics to be simultaneously optimized for a solution that is a set of agents' paths in the case of MAPF problems, and Pareto optimality is regarded as a common criterion. Here, we focus on an optimization criterion related to fairness and the worst case among the agents themselves or the facilities affected by the agents' plans, and this is also a subset of the makespan criterion. This involves several situations where the utilities/costs, including robots' lifetimes and related humans' workloads, should be balanced among individual robots or facilities without employing external payments. The applicability of these types of criteria has been investigated in several optimization problems, including distributed constraint optimization problems, multi-objective reinforcement learning, and single-agent pathfinding problems. In this study, we address the case of MAPF problems and experimentally analyze the proposed approach to reveal its effect, as well as related issues, in this class of problems.


1 INTRODUCTION

We investigate multiagent pathfinding problems that improve fairness and the worst case among multiple objective values for individual agents or facilities. Multiagent pathfinding (MAPF) problems have been widely studied as a fundamental class of problem in multiagent systems. This problem is the basis for various systems using agents that simultaneously move through an environment, such as robot navigation, autonomous taxiing of airplanes, and video games (Ma et al., 2017).

Various types of optimal and quasi-optimal solution methods, including the CA* algorithm (Silver, 2005), Conflict Based Search (CBS) (Sharon et al., 2015), push-and-rotate (De Wilde et al., 2014; Luna and Bekris, 2011), PIBT (Okumura et al., 2022) and LaCAM (Okumura, 2023) have been studied. Other studies have used the solvers of general optimization problems or translated the MAPF problem into a standard optimization problem. Moreover, different extended classes of problems and dedicated solution

methods have also been proposed (Miyashita et al., 2023; Yakovlev and Andreychuk, 2017; Andreychuk et al., 2022; Andreychuk et al., 2021).

A common target for optimization in MAPF problem settings is the total cost value of moves and actions for all agents. Another optimization criterion is the makespan, which is equivalent to the maximum cost value for all agents in a single instance of MAPF problems. As one direction of extended MAPF problems, multiple-objective problems have been studied (Weise et al., 2020; Wang et al., 2024). In general, multiple objectives represent different types of characteristics to be simultaneously optimized for a solution that is a set of agents' paths in the case of MAPF problems, and Pareto optimality is regarded as a common criterion. Here, we focus on an optimization criterion related to fairness and the worst case among the agents themselves or the facilities affected by the agents' plans, and this is also a subset of the makespan criterion. This involves several situations where the utilities/costs, including robots' lifetimes and related humans workloads, should be balanced among individual robots or facilities with-

^a  <https://orcid.org/0000-0001-8557-8167>

out employing external payments. The applicability of these types of criteria has been investigated in several optimization problems, including distributed constraint optimization problems (Matsui et al., 2018a; Matsui et al., 2018c), multi-objective reinforcement learning (Matsui, 2019), and single-agent pathfinding problems (Matsui et al., 2018b). In this study, we address the case of MAPF problems and experimentally analyze the proposed approach to reveal its effect, as well as related issues, in this class of problems.

The rest of this paper is organized as follows. In the next section, we present the background of our study, including multiagent pathfinding problems, solution methods, the leximax criterion, and application of this criterion to pathfinding problems. The details of our proposed approach are then presented in Section 3. We apply the above criterion to the cases of both agents and facilities on maps. We then experimentally investigate these approaches in Section 4 and conclude our paper in Section 5.

2 BACKGROUND

2.1 MAPF Problem

The multiagent pathfinding (MAPF) problem is an optimization problem to find the set of shortest paths (sequences of agents' actions) of multiple agents where there are no collisions among the paths. A MAPF problem is generally defined by an undirected graph $G = \langle V, E \rangle$ consisting of sets V and E of edges and vertices representing a two-dimensional map, a set of agents A , and a set of pairs of start and goal vertices (v_i^s, v_i^g) of individual agents $a_i \in A$, where $v_i^s, v_i^g \in V$. Each agent a_i must move from v_i^s to v_i^g without colliding with other agents' paths, including stay/wait actions, and the paths should be minimized with optimization criteria. In a standard setting, all move/stay actions of agents take the same cost value, and the sum of costs or the makespan among agents' paths should be minimized. As a commonly used assumption, we ignore the existence of agents who reached their goals. There are two cases of collision paths; 1) two agents are located at the same vertex at the same time (a vertex collision) and 2) two agents move on the same edge at the same time from both ends of the edge (a swapping collision). A typical basic problem employs a graph based on a four-connected grid map with obstacles, and it considers discrete time steps.

While there are different types of solution methods, we employ our customized version of the Conflict Based Search (CBS) algorithm (Sharon et al.,

2015) as a standard base method for the new class of MAPF problems tackled in this study.

2.2 CBS

The Conflict Based Search (CBS) algorithm (Sharon et al., 2015) is a fundamental complete solution method for solving MAPF problems. Although this type of algorithm faces the issue of limited scalability, a simple version is often employed as a base solver for an extended problem in the first steps of study.

This solution method consists of two layers of search. In the high-level layer, agents' collisions are managed as constraints using a best-first tree-search algorithm. We employ a version of binary-tree search. This tree is called a constraint tree (CT), where its node (CT node) represents a set of paths for all agents that are individually computed for each agent. Therefore, a CT node can be evaluated with an optimization criterion while disregarding collisions of the agents' paths. A CT node also has a set of constraints (a_i, v, t) . Here, a constraint inhibits agent a_i from locating at vertex v at time step t . The root CT node has no constraint, and there can generally be a degree of conflict among several paths of agents. In every step in CT search, a single node is selected in a best-first manner, and the first conflict in the paths, if one exists, is selected for a pair of agents. For a vertex conflict (a_i, a_j, v, t) , two child CT nodes who individually have one of two new constraints (a_i, v, t) and (a_j, v, t) are expanded. For a swap conflict, two child CT nodes with the same form of constraint are generated by considering the agents' moves. The path of a_i or a_j in a child CT node is then updated under its new constraint. If there is no conflict among paths in a selected CT node, that becomes the solution.

In the low-level layer, a pathfinding algorithm on a time-space graph, such as the A* algorithm (Hart and Raphael, 1968; Hart and Raphael, 1972; Russell and Norvig, 2003), is used to find a single agent's path for a CT node under its newly imposed constraints. Here, an agent selects either a move or stay action at each time step, and each action generally takes one time step.

Although there are several optimal and quasi-optimal extended variations (Ma et al., 2019; Barer et al., 2014) to mitigate the relatively high computational cost of the optimal search method, we employ a simple version of CBS under a limited range of problems for our first study.

2.3 Multi-Objective Optimization and Leximin/Leximax

In this study, we focus on a specific case of multiple objectives involving fairness among agents or among facilities that are affected by the agents' paths.

General multi-objective optimization problems have multiple objective functions that need to be simultaneously optimized. A fundamental minimization problem is defined as $\langle X, D, F \rangle$, where X , D , and F are a set of variables, a set of domains of variables, and a set of objective functions, respectively. Variable $x_i \in X$ takes a value in finite and discrete set $D_i \in D$. For the set of variables $X_j \subseteq X$, the j th single objective function $f_j(X_j) \in F$ is defined as $f_j(x_{j,1}, \dots, x_{j,k}) : D_{j,1} \times \dots \times D_{j,k} \rightarrow \mathbb{N}$, where $x_{j,1}, \dots, x_{j,k} \in X_j$. An objective vector \mathbf{v} , defined as $[v_1, \dots, v_k]$, represents multiple objective values of the objective functions, where $v_j = f_j(\mathcal{A}_{|X_j})$ for assignment \mathcal{A} to the variables in X_j .

The ideal goal of the problem is to simultaneously minimize the cost values of all objective functions. Since this cannot generally be fully achieved due to trade-offs between the objectives, Pareto optimality has been considered as a way to filter locally optimal solutions. Moreover, several social welfare criteria and scalarization functions have been employed to select a solution from a typically enormous number of Pareto optimal solutions (Sen, 1997; Marler and Arora, 2004).

Although the minimization on total cost $\sum_{j=1}^k f_j(X_j)$ is Pareto optimal, it only considers the global cost value. The minimization on the maximum cost $\max_{j=1}^k f_j(X_j)$ improves the worst case (min-max). However, it does not ensure the Pareto optimality due to its insufficient filtering of solutions. The ties of maximal cost values are often broken by summation.

Another criterion based on the maximum cost value lexicographically compares the cost values in two objective vectors by repeatedly comparing them from the worst (maximum) values to the best (minimum) values. This is a variant of leximin (Bouveret and Lemaître, 2009; Greco and Scarcello, 2013; Matsui et al., 2018a; Matsui et al., 2018c) for maximization problems, and thus we denote it as leximax.

For minimization problems, we employ vectors of sorted objective values, which have an inverted order from that for the leximin case, as well as a comparison operator (Matsui et al., 2018b).

Definition 1 (Descending sorted objective vector). *The values of a descending sorted objective vector are sorted in descending order.*

Definition 2 (Leximax). *Let $\mathbf{v} = [v_1, \dots, v_K]$ and $\mathbf{v}' = [v'_1, \dots, v'_K]$ denote descending objective vectors whose lengths are K . The order relation, denoted as \prec_{leximax} , is defined as $\mathbf{v} \prec_{\text{leximax}} \mathbf{v}'$ if and only if $\exists t, \forall t' < t, v_{t'} = v'_{t'} \wedge v_t < v'_{t'}$.*

The minimization on leximax improves the worst case as a variant of min-max and it also relatively improves the fairness among the objective cost values. A benefit of this criterion among fairness criteria is its decomposability. The addition of two descending sorted objective vectors provides concatenation and resorting operations.

Definition 3 (Addition of descending sorted objective vectors). *Let \mathbf{v} and \mathbf{v}' denote vectors $[v_1, \dots, v_K]$ and $[v'_1, \dots, v'_{K'}]$, respectively. The addition of two vectors, $\mathbf{v} \oplus \mathbf{v}'$, is represented as $\mathbf{v}'' = [v''_1, \dots, v''_{K+K'}]$, where \mathbf{v}'' consists of all values in \mathbf{v} and \mathbf{v}' . In addition, the values in \mathbf{v}'' are sorted in descending order.*

The addition of vectors ensures a kind of monotonicity that is necessary in optimization methods such as tree search and dynamic programming (Matsui et al., 2018a; Matsui et al., 2018b).

Proposition 1 (Invariance of leximax upon 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 $\mathbf{v} \prec_{\text{leximax}} \mathbf{v}'$, then $\mathbf{v} \oplus \mathbf{v}'' \prec_{\text{leximax}} \mathbf{v}' \oplus \mathbf{v}''$.*

When relatively fewer types of discrete objective values are employed, the descending 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., 2018a; Matsui et al., 2018b). This run-length encoding is also considered a sorted histogram, and the redundant length of the original descending sorted vector is reduced. In addition, the comparison and addition operators on the encoded vectors are available.

The applicability of these types of criteria has been investigated in several optimization problems, including distributed constraint optimization problems (Matsui et al., 2018a; Matsui et al., 2018c), multi-objective reinforcement learning (Matsui, 2019), and single-agent pathfinding problems (Matsui et al., 2018b).

2.4 Pathfinding by Fairness and Worst Case Among Individual Edge Costs

In a previous study (Matsui et al., 2018b), the applicability of the leximax criterion was further investigated for shortest pathfinding problems. Here, each edge in a graph in a traditional shortest pathfinding problem has a discrete move cost value within a relatively

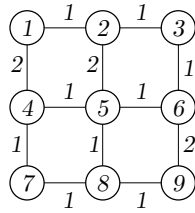


Figure 1: Lattice graph (Matsui et al., 2018b).

limited range such as $\{1, \dots, 5\}$ or $\{1, \dots, 10\}$, representing certain cost levels. Then the cost values of paths are aggregated and compared in the manner of a lexicmax criterion using an extended set of descending sorted objective vectors of arbitrary length.

In Figure 1, one of the optimal paths from start vertex 1 to goal vertex 9, under the minimization of conventional total cost values, is $(1, 2, 3, 6, 9)$, and its cost is 5. In the case of the minimization of the maximal cost values, one optimal path is $(1, 2, 3, 6, 5, 8, 9)$, where its cost is 6, and the latter path contains no edges with a cost value of 2. The goal of the problem is not only to reduce the number of edges with maximum cost values but also, if possible, to reduce the total cost value while improving the fairness among edges. This problem setting relates to cases where a route should avoid highly affected residents or extra loads at bottleneck facilities by compromising traditional optimality.

To employ best-first search and dynamic programming methods, including the Dijkstra's algorithm and the A* algorithm (Hart and Raphael, 1968; Hart and Raphael, 1972; Russell and Norvig, 2003), the operations of lexicmax are extended for subproblems of different lengths of vectors, which represent different lengths of paths. By extending the lexicmax to objective vectors of different lengths, the variable-length lexicmax, *vleximax*, is employed.

Definition 4 (Vleximax). *Let $\mathbf{v} = [v_1, \dots, v_K]$ and $\mathbf{v}' = [v'_1, \dots, v'_{K'}]$ 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 $\prec_{leximax}$.*

This comparison considers two descending sorted objective vectors that are modified from the original ones so that these vectors have the same sufficient length by padding blanks with zero cost values. The comparison of lexicmax is based on tie-breaks on objective values from the beginning of both vectors, and the redundant last parts of zeros can be ignored.

In the A* search algorithm, vertices are evaluated with descending sorted objective vectors, and the vectors are appropriately aggregated and compared. In

addition, the heuristic function for the A* algorithm can be generalized as a lower bound vector that does not exceed the optimal cost value, while the gap between an intuitively admissible heuristic distance and the true one is relatively large.

Moreover, a previous study also addressed the case of Learning Real-Time A* algorithm (Barto et al., 1995), which is an on-line search method related to reinforcement learning. Although the A* algorithm with vleximax performs correctly for traditional two dimensional maps, the on-line search case suffers from a property of the criterion. When the length of descending sorted objective vectors is not limited, the length of a lower bound vector can grow infinitely by adding lower bound objective values. This phenomenon resembles the case of negative cycles in pathfinding problems. As the result, the lower bound cost vector of a path never reaches a corresponding upper bound cost vector, while the upper bound can well decrease to the optimal one. Several approaches can mitigate this issue.

3 APPLICATION OF LEXIMAX CRITERION TO SOLUTION METHOD

We extend the MAPF problem by applying a lexicmax criterion. We address both cases of the lexicmax among agents and the vleximax among vertices in all agents' paths.

3.1 Extended Problem with Vertex Costs and Additional Criteria

We employ a MAPF problem where each vertex v in the graph of a map has a cost $c(v)$ that takes its value from a relatively limited range of discrete values such as $\{1, \dots, 5\}$ or $\{1, \dots, 10\}$. The lower and upper limit cost values are denoted by c^\perp and c^\top , and we define that $c^\perp = 0$. In the two examples above, c^\top must be greater than 5 and 10. We also define a move/stay cost $c(v', v) = c(v)$ from vertex v' to v for all v' adjacent to v , including the wait action on a time-space graph. Namely, the graphs are extended to directed graphs. The traditional MAPF is the case of $c(v) = 1$.

We employ the CBS algorithm as the basis of a solution method and apply lexicmax based operations. The aggregation and optimization criteria of cost values in a MAPF problem can be separated into those among agents' paths and others among components in each path. We denote the combination of these two classes of criteria using the form of (among agent)-

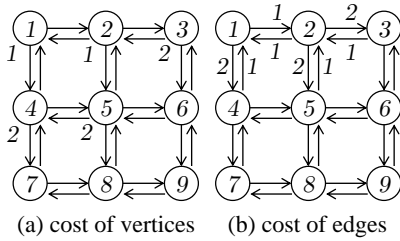


Figure 2: Lattice graph with walls.

(among path element). For a fundamental MAPF problem, the traditional sum-of-costs is denoted by sum-sum, while the makespan is identical to max-sum. We address other cases below.

3.2 Leximax Among Agents

A relatively intuitive extension of leximax-sum applies the leximax criterion to the cost values among conventional agents' paths. The high-level search of the CBS algorithm is modified to employ the descending sorted objective vector, aggregation operator, and comparison operator of leximax for the aggregation of agents' paths, while each agent's pathfinding in the lower-level is based on the conventional A* algorithm with the summation of cost values on a time-space graph.

An objective value in a sorted vector is a cost value of an individual agent's path, and the first objective value is identical to the case of minimizing makespan (max-sum). The ties of solutions with the makespan criterion are broken by employing the leximax comparison operator. In the minimization of cost vectors among agents, we simply employ the upper limit cost value $c^\top = \infty$ as a default value for cost vectors, and the default upper limit vector contains $|A|$ cost values of c^\top .

Although this case of optimization is a natural extension of the case of minimizing makespan, it might cause combinatorial explosion in dense settings of agents at an earlier stage than the case of makespan. The leximax criterion could have a bias to expand a specific CT node with its detailed tie-break in the high-level search, and it is not possible to ensure that such CT nodes provide promising results.

3.3 Vleximax Among Vertices

The pathfinding part of each agent in the low-level search of the CBS algorithm can employ the vleximax criterion as an extended case of single-agent pathfinding (Matsui et al., 2018b), while we must find the paths on a time-space graph. Here, we focus on the balance of cost values among not agents but ver-

tices related to particular humans or facilities by assuming that for a unit move/stay action, an agent requires the cost of its destination vertex of the unit action. In the high-level search, the aggregation of cost values among agents should also be modified to employ vleximax, and this case is denoted by vleximax-vleximax.

Although this modification appears to be a natural extension, it raises several challenging issues due to the property of the criterion that resemble those in the case of the previous work (Matsui et al., 2018b). First, in the pathfinding for each agent, we solve the paths on a time-space graph, and the number of paths to be explored is substantially unbounded for a simple best-first search method, even if there is a time limit.

Pathfinding methods generally depend on tension applied to the paths that are implicitly provided by the summation criterion of cost values among paths. Otherwise, they depend on a limited search space that can be sufficiently explored. While this is a common issue, our case is more problematic.

In the case of minimization with the vleximax criterion, the length of the lower bound of the descending sorted vector can grow infinitely. In the example shown in Figure 2, which resembles one in the previous study (Matsui et al., 2018b), an agent should move from vertex 1 to vertex 9. We note that the search is actually performed on a time-space graph where the initial time step is $t = 0$. Here, we consider lower bound cost vectors $g(t, v)$ of paths containing a partial path from the start vertex 1 to each vertex (t, v) . The global lower bound function $g(t, v)$ consists of the summation of $f(t, v)$ and $h(t, v)$, which represent traversed and remaining estimated distances. For the A* algorithm, assume a heuristic function $h(t, v)$ that returns a vector consisting of identical lower bound values, where the length of the vector is identical to the Manhattan distance from v to vertex 9. The lower bound cost value is extracted from each problem. Here $g(0, 1) = f(0, 1) + h(0, 1) = [] + [1_{(4)}]$ for vertex $(0, 1)$, and $c_{(l)}$ denotes that cost value c is contained l times.

When vertex $(0, 1)$ is extracted, its adjacent vertices have $g(1, 1) = [1] + [1_{(4)}]$, $g(1, 2) = [1] + [1_{(3)}]$, and $g(1, 4) = [2] + [1_{(3)}]$. With the best-first search under vleximax, vertex $(1, 2)$ is then selected and extracted.

The vertices adjacent to vertex $(1, 2)$ have $f(2, 1) = [1, 1] + [1_{(4)}]$, $f(2, 2) = [1, 2] + [1_{(3)}]$, $f(2, 3) = [1, 2] + [1_{(2)}]$, and $f(2, 5) = [1, 2] + [1_{(2)}]$. Therefore, vertex $(2, 1)$ is selected under vleximax, and the path where the agent moves to its start location is expanded. This round-trip expansion between vertex 1 and vertex 2 infinitely repeats, adding the

minimum cost value of 1 to corresponding vectors.

To avoid this situation, we first limit the maximum length of paths for each agent a_i as

$$l_i^{dist} = c^{dist} \times d^*(v_i^s, v_i^g). \quad (1)$$

Here, c^{dist} is a parameter with a sufficient margin, and $d^*(v, v')$ is the distance (e.g. the length of the shortest path) between vertices v and v' on a graph representing a two-dimensional map with unit cost values. For the additional limitation based on the distance in the traditional problem, we additionally compute the distances in the A* algorithm, and l_i^{dist} is applied to the length of an objective vector. However, this limitation of the total path length is insufficient when agents have relatively long paths. Since the paths based on vleximax tend to wind while avoiding vertices with higher cost values, the paths can easily conflict and increases unpromising CT nodes in the CBS. Therefore, we need to apply some tension to the paths in addition to the simple *boxing* of paths. Here, we employ a proportional constraint to the paths. If an expanded vertex (t, v) in the A* search algorithm satisfies the condition

$$f'(t, v) + \frac{d^*(v, v_i^g)}{d^*(v_i^s, v_i^g)} \times l_i^{dist} \leq l^{dist}, \quad (2)$$

the vertex is stored in a priority queue to be searched, and it is ignored otherwise. Here, $f'(t, v)$ represents the length of an objective vector for the traversed path, which has the traditional distance on a map with unit cost values.

The high-level search in the CBS algorithm also suffers from related issues. Since the best-first search for a CT does not consider the possibility of conflicts, it can easily expand unpromising CT nodes. Unfortunately, this highly affects the case of vleximax.

To mitigate this issue, we also have to limit the search space of a CT tree. We limit the depth of the CT tree by

$$l^{cnst} = c^{cnst} \times |A|, \quad (3)$$

where c^{cnst} is a parameter with a sufficient margin.

In addition, we also limit the beam of the CT search by b^{depth} and b^{radix} and limit the number of expanded CT nodes $l^{breadth}$ from the depth of b^{depth} as follows.

$$l^{breadth} = 2^{(b^{depth}-1)} \times b^{radix}^{(l^{cnst}-b^{depth}+1)} \quad (4)$$

in the case of binary-tree search. Here, $1 < l^{breadth} < 2$ and $b^{depth} \leq l^{cnst}$.

We also apply the limitation of CT search to the cases of not vleximax but leximax operators.

While these limitations compromise the optimality, the effect of leximax and vleximax can be obtained if there is room for some solutions preferred

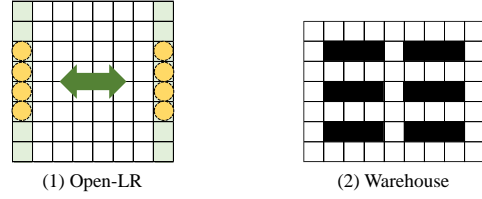


Figure 3: Map instances.

over that based on sum-sum in the limited search space. The main aim of above the mitigation techniques is to extend the limited range of solvable problems in the first experimental analysis, and there might be opportunities to apply more efficient techniques in a future study.

4 EVALUATION

4.1 Settings

We experimentally evaluated our proposed approach. In the first investigation, we employed a CBS-based solution method without major techniques for efficiency, and this also limited the scale of problems, including the number of agents and the size of maps. Figure 3 shows the map instances employed in the experiment. In the settings of Open-LR, the halves of agents are initially located at the left or right sides of the map, and their goals are the opposite side of the map. In the case of Warehouse, start/goal locations of agents were randomly selected from non-obstacle vertices/cells without overlap under uniform distribution. Although we also evaluated the optimization criteria for several different maps having some structures such as rooms, in addition to the case of Warehouse, we found that the results resembled. We present the results of the cases with integer cost values of $[1, 5]$ and $[1, 10]$.

We evaluated the following versions of solution methods that use different combinations of optimization criteria.

- sum-sum: Sum-of-cost (SoC).
- max-sum: Makespan.
- ms-sum: Ties of worst case cost values among agents are broken by the summation (lexicographically augmented weighted Tchebycheff function).
- lxm-sum: Leximax among cost values of agents' paths.
- vms-vms: Lexicographic augmented weighted Tchebycheff function for the cost values among vertices in all agents' paths.

Table 1: Solved problems (Open-LR, Agents).

Vertex cost	Alg. \ A	2	4	6	8	10	12	14	16
5	sum-sum	1	1	1	1	1	1	1	0
	max-sum	1	1	1	1	1	1	0	0
	max-sum-l*	1	1	1	1	1	1	1	0
	ms-sum	1	1	1	1	1	1	1	0
	ms-sum-l*	1	1	1	1	1	1	1	0
	lxm-sum	1	1	1	1	1	1	1	0
lxm-sum-l*	1	1	1	1	1	1	1	0	
10	sum-sum	1	1	1	1	1	1	1	0
	max-sum	1	1	1	1	1	1	0	0
	max-sum-l*	1	1	1	1	1	1	1	0
	ms-sum	1	1	1	1	1	1	1	0
	ms-sum-l*	1	1	1	1	1	1	1	0
	lxm-sum	1	1	1	1	1	0	0	0
lxm-sum-l*	1	1	1	1	1	0	0	0	

* -l: limitation of CT with $(c^{cnst}, b^{depth}, b^{radix})=(2, 16, 1.25)$ for $|A| \geq 10$.

- vlxm-vlxm: Vleximax for the cost values among vertices in all agents' paths.

Several methods employ additional techniques to limit the size of search spaces, and those parameters are also presented in the results.

We evaluated the number of solved instances by the solvers. For solved instances, the numbers of expanded nodes in the CBS algorithm, including the A* algorithm in its low-level layer, were evaluated. For the solution quality, we evaluated the SoC, makespan and the Theil index (Matsui et al., 2018b), which is a measurement of inequality. For n objectives, Theil index T is defined as

$$T = \frac{1}{n} \sum_i \frac{v_i}{\bar{v}} \log \frac{v_i}{\bar{v}} \quad (5)$$

where v_i is the utility/cost value of an objective and \bar{v} is the mean value for all of the objectives. The Theil index takes a value in $[0, \log n]$. If all utility/cost values are identical, the Theil index takes zero. Inequalities based on different numbers of members can be compared using population independence. We note that the minimization of the leximin criterion does not precisely minimize inequality, but rather is employed as a tool to analyze the results.

Except for the case of Open-LR shown in Figure 3, the results were averaged over ten instances for each problem setting by randomly assigning initial locations of agents within the limitations of the settings. The results were aggregated for the instances whose all trials were completed. We performed the experiment on a computer with g++ (GCC) 8.5.0 -O3, Linux 4.18, Intel (R) Core (TM) i9-9900 CPU @ 3.10 GHz, and 64 GB memory.

4.2 Results

Tables 1 and 2 show the number of solved problems for the case of Open-LR. Here, we mainly concen-

Table 2: Solved problems (Open-LR, Vertices).

vertex cost	alg. \ A	2	4	6	8	10	12	14	16
5	vms-vms-lt*	1	0	0	0	0	0	0	0
	vms-vms-lp*	1	1	1	1	1	1	1	0
	vlxm-vlxm-lt*	1	0	0	0	0	0	0	0
	vlxm-vlxm-lp*	1	1	1	1	1	1	0	0
10	vms-vms-lt*	1	0	0	0	0	0	0	0
	vms-vms-lp*	1	1	1	1	1	1	1	0
	vlxm-vlxm-lt*	1	0	0	0	0	0	0	0
	vlxm-vlxm-lp*	1	1	1	1	1	1	0	0

* -lt: limitation of CT and the total path length without the proportional constraint $(c^{cnst}, c^{dist}, b^{depth}, b^{radix})=(2, 2, 0, 2)$, -lp: limitation of CT and the total path length with proportional constraint using the same parameter.

Table 3: Expanded nodes (Open-LR, Agents, 10 agents).

Vertex cost	Alg.	#CT nodes expanded	#constraints of best slt.	#A* nodes processed		
				Total	Ave.	Max.
5	sum-sum	3871	13	1186931	305.9	700
	max-sum	4017	13	1449448	360.0	706
	max-sum-l*	4017	13	1449448	360.0	706
	ms-sum	787	13	258205	324.4	639
	ms-sum-l*	787	13	258205	324.4	639
	lxm-sum	23013	13	8363639	363.3	706
lxm-sum-l*	10273	13	4327912	365.3	706	
10	sum-sum	1265	10	786986	617.7	1121
	max-sum	543	9	358693	649.8	1121
	max-sum-l*	543	9	358693	649.8	1121
	ms-sum	749	10	496742	655.3	1121
	ms-sum-l*	749	10	496742	655.3	1121
	lxm-sum	3207	9	2325004	722.9	1131
lxm-sum-l*	2571	9	1895327	710.9	1131	

* -l: limitation of CT with $(c^{cnst}, b^{depth}, b^{radix})=(2, 16, 1.25)$ for $|A| \geq 10$.

Table 4: Expanded nodes (Open-LR, Vertices, 10 agents).

Vertex cost	Alg.	#CT nodes expanded	#constraints of best slt.	#A* nodes processed		
				Total	Ave.	Max.
1	vms-vms-lt*	3749	10	213346	56.8	80
	vms-vms-lp*	6653	10	352260	52.9	72
	vlxm-vlxm-lt*	3749	10	213346	56.8	80
	vlxm-vlxm-lp*	6653	10	352260	52.9	72
5	vms-vms-lt*	24821	11	2508100	100.2	138
	vms-vms-lp*	83820	12	11770645	100.7	134
	vlxm-vlxm-lt*					
	vlxm-vlxm-lp*					
10	vms-vms-lt*	57397	9	6180128	106.1	139
	vms-vms-lp*	69593	12	9244360	98.9	140
	vlxm-vlxm-lt*					
	vlxm-vlxm-lp*					

* -lt: limitation of CT and the total path length without the proportional constraint $(c^{cnst}, c^{dist}, b^{depth}, b^{radix})=(2, 2, 0, 2)$, -lp: limitation of CT and the total path length with proportional constraint using the same parameter.

trated on the settings that can be solved by a traditional method using SoC. We limited the solution methods based on a safeguard within the number of 5×10^5 CT nodes and 500 time steps for actions. In addition, the execution of several instances was terminated at the cut-off time of five minutes. In the results, we employed the additional methods to limit search spaces for several sizes of the problems. The blank cells in the tables represent that the experiment was omitted for small sizes of problems or relatively large-scale problems could not be solved.

Table 5: Solution quality (Open-LR, Agents).

Vertex cost	A Alg.	6			8			10			12		
		SoC	MS	Th	SoC	MS	Th	SoC	MS	Th	SoC	MS	Th
5	sum-sum	108	21	0.011	149	21	0.009	193	22	0.011	245	25	0.012
	max-sum	108	21	0.011	149	21	0.009	193	22	0.011	245	25	0.012
	max-sum-l*							193	22	0.011	245	25	0.012
	ms-sum	108	21	0.011	149	21	0.009	193	22	0.011	245	25	0.012
	ms-sum-l*							193	22	0.011	245	25	0.012
	lxm-sum	108	21	0.011	149	21	0.009	193	22	0.011	248	25	0.009
lxm-sum-l*							193	22	0.011	248	25	0.009	
10	sum-sum	197	37	0.012	274	39	0.011	355	41	0.012	450	47	0.014
	max-sum	197	37	0.012	274	39	0.011	355	41	0.012	455	46	0.012
	max-sum-l*							355	41	0.011	455	46	0.012
	ms-sum	197	37	0.012	274	39	0.011	355	41	0.012	455	46	0.012
	ms-sum-l*							355	41	0.012	455	46	0.012
	lxm-sum	197	37	0.012	274	39	0.011	355	41	0.011	455	46	0.012
lxm-sum-l*							355	41	0.011	455	46	0.012	

* -l: limitation of CT with $(c^{cnst}, b^{depth}, b^{radix})=(2, 16, 1.25)$ for $|A| \geq 10$.

Table 6: Solution quality (Open-LR, Vertices).

Vertex cost	A Alg.	6			8			10			12		
		SoC	MS	Th	SoC	MS	Th	SoC	MS	Th	SoC	MS	Th
5	sum-sum	108	5	0.185	149	5	0.175	193	5	0.158	245	5	0.149
	max-sum	108	5	0.185	149	5	0.175	193	5	0.158	245	5	0.149
	vms-vms-lt*												
	vms-vms-lp*	116	5	0.175	161	5	0.163	209	5	0.146	260	5	0.132
	vlxm-vlxm-lt*												
	vlxm-vlxm-lp*	117	5	0.175	166	5	0.160	214	5	0.148	273	5	0.135
10	sum-sum	197	10	0.231	274	10	0.229	355	10	0.204	450	10	0.195
	max-sum	197	10	0.231	285	10	0.228	355	10	0.204	455	10	0.198
	vms-vms-lt*												
	vms-vms-lp*	211	10	0.226	301	10	0.200	385	10	0.181	482	10	0.168
	vlxm-vlxm-lt*												
	vlxm-vlxm-lp*	212	10	0.229	302	10	0.204	392	10	0.188	501	10	0.167

* -lt: limitation of CT and the total path length without the proportional constraint $(c^{cnst}, c^{dist}, b^{depth}, b^{radix})=(2, 2, 0, 2)$, -lp: limitation of CT and the total path length with proportional constraint using the same parameter.

Table 7: Solution quality (Warehouse, Agents).

Vertex cost	A Alg.	4			6			8			10		
		SoC	MS	Th	SoC	MS	Th	SoC	MS	Th	SoC	MS	Th
5	sum-sum	65.7	24.2	0.097	107.7	27.4	0.078	147.9	27.7	0.075			
	max-sum	65.7	24.2	0.097	109.7	26.9	0.074	149.7	27	0.070			
	max-sum-l*										200.6	29	0.082
	ms-sum	65.7	24.2	0.097	109.3	26.9	0.075	149.1	27	0.070			
	ms-sum-l*										198.1	29	0.083
	lxm-sum	65.7	24.2	0.097	109.6	26.9	0.074	149.2	27	0.069			
lxm-sum-l*										198.8	29	0.081	
10	sum-sum	122	45.7	0.101	198.5	50.5	0.078	272.8	50.8	0.073			
	max-sum	123.8	45.1	0.097	202.2	48.8	0.073	274.6	49.7	0.068			
	max-sum-l*										366.6	54.1	0.081
	ms-sum	123.8	45.1	0.097	201.7	48.8	0.073	273.8	49.7	0.068			
	ms-sum-l*										365.1	54.1	0.082
	lxm-sum	123.8	45.1	0.097	203.6	48.8	0.070	277.5	49.7	0.067			
lxm-sum-l*										367.2	54.1	0.079	

* -l: limitation of CT with $(c^{cnst}, b^{depth}, b^{radix})=(2, 16, 1.25)$ for $|A| \geq 10$.

Table 8: Solution quality (Warehouse, Vertices).

Vertex cost	A Alg.	2			4			6			8		
		SoC	MS	Th	SoC	MS	Th	SoC	MS	Th	SoC	MS	Th
5	sum-sum	31	5	0.125	65.7	5	0.124	107.7	5	0.131	147.9	5	0.122
	max-sum	31.5	5	0.128	65.7	5	0.124	109.7	5	0.132	149.7	5	0.126
	vms-vms-lt*	31.2	5	0.125	65.9	5	0.123	108.5	5	0.140	148.6	5	0.127
	vms-vms-lp*	31.1	5	0.121									
	vlxm-vlxm-lt*	32.1	5	0.126	67.6	5	0.125	115.6	5	0.139	159.7	5	0.125
	vlxm-vlxm-lp*	31.2	5	0.117									
10	sum-sum	57.5	9.4	0.161	122	9.3	0.154	198.5	9.7	0.167	272.8	9.9	0.161
	max-sum	58.4	9.4	0.168	123.8	9.3	0.161	202.2	9.7	0.181	274.6	10	0.164
	vms-vms-lt*				128.7	9.1	0.157	206.4	9.3	0.176	289.1	9.5	0.156
	vms-vms-lp*												
	vlxm-vlxm-lt*				130.4	9.2	0.155	219.6	9.3	0.177			
	vlxm-vlxm-lp*	57.5	9.4	0.149									

* -lt: limitation of CT and the total path length without the proportional constraint $(c^{cnst}, c^{dist}, b^{depth}, b^{radix})=(2, 2, 0, 2)$, -lp: limitation of CT and the total path length with proportional constraint using the same parameter.

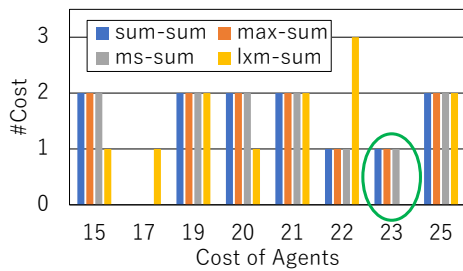


Figure 4: Solution quality (Open-LR, Agents) (12 agents, vertex cost = 5).

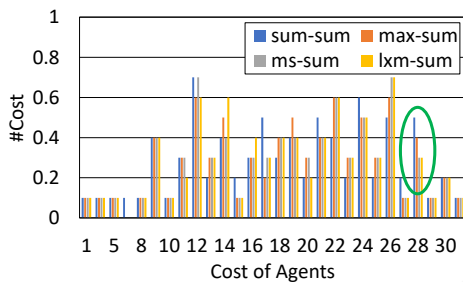


Figure 5: Solution quality (Warehouse, Agents) (8 agents, vertex cost = 5).

While the methods with a conventional criterion found solutions in the relatively easy settings, several settings were difficult for the leximax-based methods, which reveals the necessity of appropriate techniques to effectively exclude unpromising solutions from search spaces. In several results, the suitable limitation of search space increased the success cases of the search.

Tables 3 and 4 show the number of expanded nodes in search processes for the case of Open-LR. In both the high- and low-level searches, the size of the searched space for leximax variants is generally greater than the cases of traditional optimization criteria. Although this additional computational cost is inherent in the specific criterion that considers fairness, the result revealed the poor compatibility between the leximax variants and the best-first search methods without any limitation on search spaces. While the criteria of leximax and vleximax required additional computation to operate sorted objective vectors, that was acceptable in comparison to the serious situations of the excessive expansion of CT nodes.

In our preliminary experiment, the limitation to winding paths in low-level search often locally inconsistent with desired moves avoiding/waiting other agents' paths due to the limited length and the constraint to keep some tension of paths. Although this situation is not the case of the optimization among agents' paths, it specifically deteriorated the results of the optimization with vleximax. This revealed the necessity of further investigation for more flexible re-

striction of the paths by considering the interaction among agents having different lengths of paths.

In the high-level search, the best-first search of the pure CBS algorithm that is not aware of the relationship among the inserted constraints and their resulting paths often expanded unpromising CT nodes. Therefore, the search trees which is limited by some simple strategies of beam search is often filled by such CT nodes. To address this situation, more sophisticated versions, in which both levels of search cooperate, should be applied to this class of problems.

Tables 5-8 show the solution quality among different optimization criteria. Since leximax and maxsum are variants of 'max', the maximum cost value is theoretically identical. However, we employed a certain limitation on the search space for leximax and maxsum, and excessive settings of this limitation reduced the characteristics of the criteria. In well-controlled cases of the optimization among agents' paths, the leximax variants reduced the number of maximum cost values by compromising on the total cost value. Consequently, the Theil index values were relatively decreased with the leximax variants. In the results of the optimization among the agents' paths, the maximum cost values were almost identical in most problem settings. In particular, the case of open grid with relatively fewer number of agents appeared to reduce the opportunities of difference of results.

In several cases of the optimization among agents' paths, leximax criterion reduced the number of higher cost values in objective vectors. Figures 4 and 5 show the cases of the optimization among agents' paths, where leximin criterion worked in accordance with its theoretical property.

However, in the case of the optimization among path elements (i.e., vertices' costs), the results by vleximax revealed the difficulty of tuning the search. When the optimization criteria of leximin variants are affected by some disturbance in search processes, that often fail to handle the worst case cost values. As a result, this can totally increase cost values, including the worst value, although the fairness among the cost values is relatively preserved.

Regarding the scalability, the complete algorithms are not so promising for large-scale problems, and those methods are often employed to evaluate relatively small problems in prototyping steps of new classes of extended problems. Although we followed this manner as the first study, the result revealed the incompatibility of vleximax criterion with the solution methods based on simple best-first search.

A possible approach might employ appropriate bounds of search space by referring the result of the standard summation criterion and performs dedicated

search strategies to efficiently cover the bounded search space. Opportunities might also exist to employ local search methods that start from reasonable initial solutions based on other optimization criteria and find better solutions improving fairness and the worst case.

5 CONCLUSIONS

We investigated multiagent pathfinding problems that improve both fairness and the worst case among multiple objective values involving individual agents or facilities. In the study, we applied variants of the leximax criterion to MAPF problems and evaluated this method with extended versions of the CBS algorithm. The results revealed issues in controlling a search with the leximax variants at both levels of CBS when employing best-first search, while some effect of the criterion was obtained in the optimization among agents' paths.

Although we addressed the case with the CBS algorithm as a standard approach in our first study, the result revealed several issues regarding the incompatibility between the vleximax criterion and simple best-first search methods. As discussed in the previous section, opportunities might exist to additional extension to more appropriately guide the best-first search by considering the relationship among agent's paths and the cooperation of high- and low- level search methods. The results might also suggest that this kind of criterion is more compatible with other approaches, including incomplete solution methods. Partially greedy approaches such as variants of the CA* algorithm or local search methods, rather than comprehensive methods based on a fully best-first approach, also should be addressed. Our future work will address an investigation in this direction as well as analysis in more practical problem domains. While we concentrated on the comparison of a few optimization criteria as the first study, more extensive survey regarding relating classes of MAPF/planning problems will also be included in our future work.

ACKNOWLEDGEMENTS

This study was supported in part by The Public Foundation of Chubu Science and Technology Center (thirty-third grant for artificial intelligence research) and JSPS KAKENHI Grant Number 22H03647.

REFERENCES

- Andreychuk, A., Yakovlev, K., Boyarski, E., and Stern, R. (2021). Improving Continuous-time Conflict Based Search. In *Proceedings of The Thirty-Fifth AAAI Conference on Artificial Intelligence*, volume 35, pages 11220–11227.
- Andreychuk, A., Yakovlev, K., Surynek, P., Atzmon, D., and Stern, R. (2022). Multi-agent pathfinding with continuous time. *Artificial Intelligence*, 305:103662.
- Barer, M., Sharon, G., Stern, R., and Felner, A. (2014). Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *Proceedings of the Annual Symposium on Combinatorial Search*, pages 19–27.
- 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.
- De Wilde, B., Ter Mors, A. W., and Witteveen, C. (2014). Push and rotate: A complete multi-agent pathfinding algorithm. *J. Artif. Int. Res.*, 51(1):443–492.
- Greco, G. and Scarcello, F. (2013). Constraint satisfaction and fair multi-objective optimization problems: Foundations, complexity, and islands of tractability. In *Proceedings of the Twenty-Third 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.
- Luna, R. and Bekris, K. E. (2011). Push and swap: Fast cooperative path-finding with completeness guarantees. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, volume 1, pages 294–300.
- Ma, H., Harabor, D., Stuckey, P. J., Li, J., and Koenig, S. (2019). Searching with consistent prioritization for multi-agent path finding. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, pages 7643–7650.
- Ma, H., Li, J., Kumar, T. S., and Koenig, S. (2017). Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the Sixteenth Conference on Autonomous Agents and MultiAgent Systems*, pages 837–845.
- 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. (2019). A Study of Joint Policies Considering Bottlenecks and Fairness. In *Proceedings of the*

- Eleventh International Conference on Agents and Artificial Intelligence*, volume 1, pages 80–90.
- Matsui, T., Matsuo, H., Silaghi, M., Hirayama, K., and Yokoo, M. (2018a). Leximin asymmetric multiple objective distributed constraint optimization problem. *Computational Intelligence*, 34(1):49–84.
- Matsui, T., Silaghi, M., Hirayama, K., Yokoo, M., and Matsuo, H. (2018b). Study of route optimization considering bottlenecks and fairness among partial paths. In *Proceedings of the Tenth International Conference on Agents and Artificial Intelligence*, pages 37–47.
- Matsui, T., Silaghi, M., Okimoto, T., Hirayama, K., Yokoo, M., and Matsuo, H. (2018c). Leximin multiple objective dcops on factor graphs for preferences of agents. *Fundam. Inform.*, 158(1-3):63–91.
- Miyashita, Y., Yamauchi, T., and Sugawara, T. (2023). Distributed planning with asynchronous execution with local navigation for multi-agent pickup and delivery problem. In *Proceedings of the Twenty-Second International Conference on Autonomous Agents and Multiagent Systems*, page 914–922.
- Okumura, K. (2023). LaCAM: search-based algorithm for quick multi-agent pathfinding. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, pages 11655–11662.
- Okumura, K., Machida, M., Défago, X., and Tamura, Y. (2022). Priority Inheritance with Backtracking for Iterative Multi-Agent Path Finding. *Artificial Intelligence*, 310.
- 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.
- Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2015). Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*, 219:40–66.
- Silver, D. (2005). Cooperative Pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 117–122.
- Wang, F., Zhang, H., Koenig, S., and Li, J. (2024). Efficient approximate search for multi-objective multi-agent path finding. In *Proceedings of the Thirty-fourth International Conference on Automated Planning and Scheduling*, pages 613–622.
- Weise, J., Mai, S., Zille, H., and Mostaghim, S. (2020). On the scalable multi-objective multi-agent pathfinding problem. In *Proceedings of the 2020 IEEE Congress on Evolutionary Computation*, pages 1–8.
- Yakovlev, K. S. and Andreychuk, A. (2017). Any-angle pathfinding for multiple agents based on SIPP algorithm. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling*, pages 586–593.