New Techniques and Algorithms for Multiobjective and Lexicographic Goal-Based Shortest Path Problems

PhD Dissertation

Francisco J. Pulido Arrebola
Supervised by: Dr. Lawrence Mandow

Dept. Lenguajes y Ciencias de la Computación
E.T.S. Ingeniería Informática
Universidad de Málaga

July 7, 2015

Outline

Outline

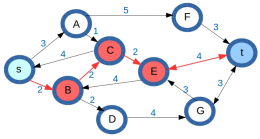
Outline

The Shortest Path Problem (SPP)

Definition

The SPP consists in finding a minimal cost path between two nodes in a graph, where the graph is composed of arcs labeled with the cost of the transition between nodes.

- Solved by Dijkstra in 1959 and by Hart et al. (A^*) in 1968 exploiting specific problem knowledge.



The introduction of this thesis begins with the shortest path problem. Given a problem that can be modeled as a graph with nodes representing different states of the problem and the arcs the cost of the transition between those states, the Shortest Path problem is the problem of finding the route with minimal cost between two given nodes, the start and the target nodes.

The Dutch computer scientist Dijkstra in 1959 presented the first algorithm to solve this problem and Hart et al. in 1968 presented A^*, the first optimal algorithm to solve the problem by using specific knowledge acquired from the problem to be solved. Since then, the shortest path problem has been one of the most studied problems in the Artificial Intelligence and Operational Research fields

The Shortest Path Problem (SPP)

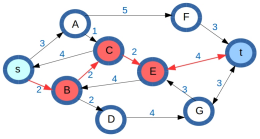
Definition

The SPP consists in finding a minimal cost path between two nodes in a graph, where the graph is composed of arcs labeled with the cost of the transition between nodes.

- Solved by Dijkstra in 1959 and by Hart et al. (A^*) in 1968 exploiting specific problem knowledge.

Research fields

- Artificial Intelligence
- Operational Research

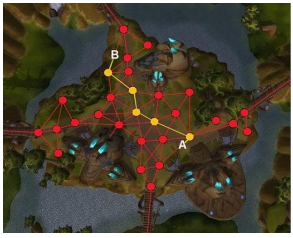


The introduction of this thesis begins with the shortest path problem. Given a problem that can be modeled as a graph with nodes representing different states of the problem and the arcs the cost of the transition between those states, the Shortest Path problem is the problem of finding the route with minimal cost between two given nodes, the start and the target nodes.

The Dutch computer scientist Dijkstra in 1959 presented the first algorithm to solve this problem and Hart et al. in 1968 presented A^*, the first optimal algorithm to solve the problem by using specific knowledge acquired from the problem to be solved. Since then, the shortest path problem has been one of the most studied problems in the Artificial Intelligence and Operational Research fields

Applications to the SPP

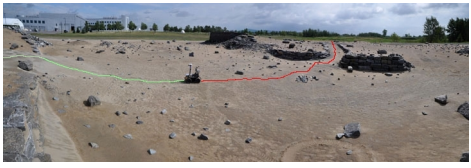
- Path planning in game maps.
  - The game map is represented as a graph.
  - The characters movement through the map is solved as a SPP.



One of the typical applications for the SPP is path planning in game maps. As we see in this picture, the possible movements of the characters within the map are represented as a graph.

Applications to the SPP

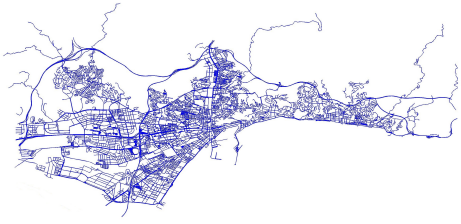
- Motion planning in mobile robots.
  - The terrain is represented as a graph.
  - The robot must find a path avoiding obstacles.



Another application for the SPP is motion planning in mobile robots, where the terrain represents the graph that the robot traverses avoiding the obstacles.

Applications to the SPP

- Route planning in road maps.
  - Real road networks are graphs.
  - Arcs represent road junctions.
  - Arcs are typically labeled with the distance between junctions.



And, perhaps, the route planning in road networks is one of the most natural and studied applications of the shortest path problem. We use in our daily basis google maps or GPS devices to help us plan our trips, and find for us the shortest route to our destination.

Multicriteria Search Problem (MSP)

Definition

MSP is the natural extension of SPP to the multicriteria case, where arcs are labeled with vectors instead of scalar values.

SPP	$\longrightarrow$	single-objective optimization problem
Decision problems	$\longrightarrow$	multiple conflicting criteria

SPP has been traditionally considered a single-objective optimization problem. Real decision problems $\longrightarrow$ frequently involve multiple conflicting criteria. What is the Multicriteria Search Problem? It's the natural extension of the Shortest Path Problem when several criteria are taken into account simultaneously. The solution to this problem is no longer a single optimal value, but a set of non-dominated or Pareto-optimal paths. We will get deeper into this concept in the following slides, but let's first talk about human decisions. IMP: Decisions are taken by humans

Human decisions

Humans make choices according to their personal preferences, which involve simultaneously multiple, and often, conflicting criteria.

How do we make decisions? We are complex, that's certain, and we make decisions considering multiple criteria, that's certain too. If I'm going to buy a new car, I will consider the price, speed, brand, color... and if I'm checking my GPS device and it is presented to me the shortest route ... it seems that something is missing, isn't it?

Human decisions

Humans make choices according to their personal preferences, which involve simultaneously multiple, and often, conflicting criteria.

New car?

- Price
- Speed
- Brand
- Color

How do we make decisions? We are complex, that's certain, and we make decisions considering multiple criteria, that's certain too. If I'm going to buy a new car, I will consider the price, speed, brand, color... and if I'm checking my GPS device and it is presented to me the shortest route ... it seems that something is missing, isn't it?

Human decisions

Humans make choices according to their personal preferences, which involve simultaneously multiple, and often, conflicting criteria.

New car?

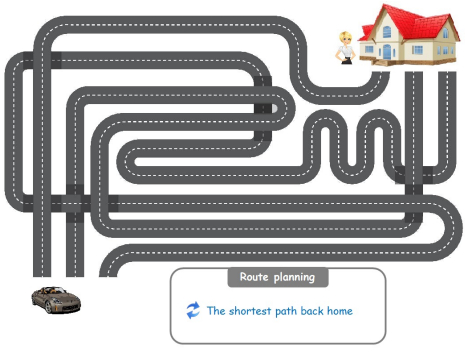
- Price
- Speed
- Brand
- Color

A route home?

- Distance?

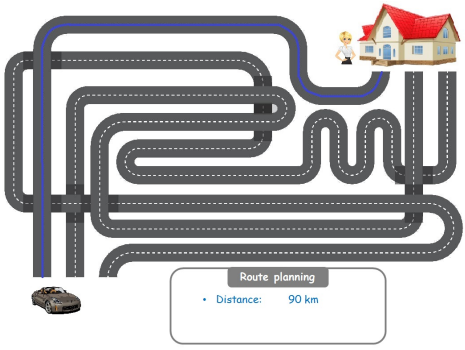
How do we make decisions? We are complex, that's certain, and we make decisions considering multiple criteria, that's certain too. If I'm going to buy a new car, I will consider the price, speed, brand, color... and if I'm checking my GPS device and it is presented to me the shortest route ... it seems that something is missing, isn't it?

Route planning in road maps



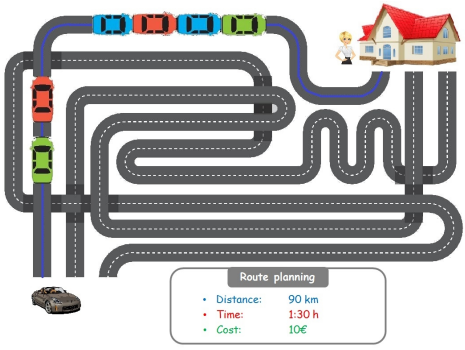
Let me now present an example of a typical route planning problem. I have finished an important meeting and I will use my fantastic route planning application to find me the shortest path back home.
The length of the shortest path from all possible routes is 90 km.
Wait! There's a traffic jam on that route and the travel time would be an hour and a half.
Maybe, what I do want is not the shortest but the fastest route.
This route is 10 kms longer but half an hour faster, it seems to me a good trade-off.
or not, there's a toll and this route is 8 euros more expensive than the last.
What about a cheaper route?
Well, this one is 20 kms longer than the first alternative, 10 minutes slower than the second, but cheaper.

Route planning in road maps



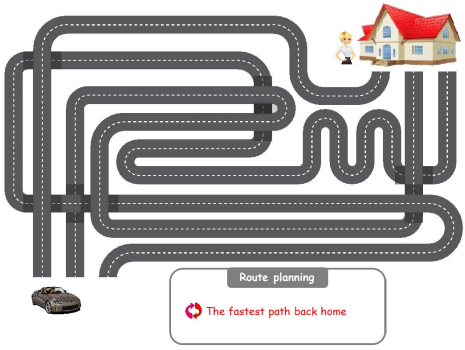
Let me now present an example of a typical route planning problem. I have finished an important meeting and I will use my fantastic route planning application to find me the shortest path back home.
The length of the shortest path from all possible routes is 90 km.
Wait! There's a traffic jam on that route and the travel time would be an hour and a half.
Maybe, what I do want is not the shortest but the fastest route.
This route is 10 kms longer but half an hour faster, it seems to me a good trade-off.
or not, there's a toll and this route is 8 euros more expensive than the last.
What about a cheaper route?
Well, this one is 20 kms longer than the first alternative, 10 minutes slower than the second, but cheaper.

Route planning in road maps



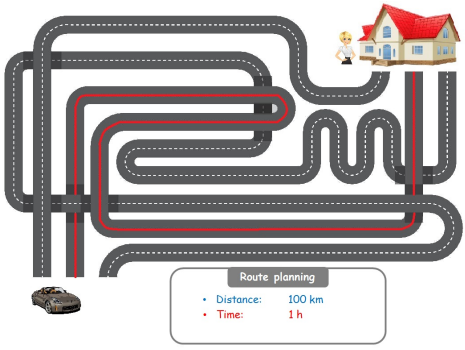
Let me now present an example of a typical route planning problem. I have finished an important meeting and I will use my fantastic route planning application to find me the shortest path back home.
The length of the shortest path from all possible routes is 90 km.
Wait! There's a traffic jam on that route and the travel time would be an hour and a half.
Maybe, what I do want is not the shortest but the fastest route.
This route is 10 kms longer but half an hour faster, it seems to me a good trade-off.
or not, there's a toll and this route is 8 euros more expensive than the last.
What about a cheaper route?
Well, this one is 20 kms longer than the first alternative, 10 minutes slower than the second, but cheaper.

Route planning in road maps



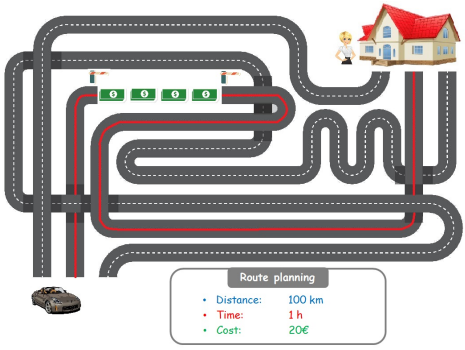
Let me now present an example of a typical route planning problem. I have finished an important meeting and I will use my fantastic route planning application to find me the shortest path back home.
The length of the shortest path from all possible routes is 90 km.
Wait! There's a traffic jam on that route and the travel time would be an hour and a half.
Maybe, what I do want is not the shortest but the fastest route.
This route is 10 kms longer but half an hour faster, it seems to me a good trade-off.
or not, there's a toll and this route is 8 euros more expensive than the last.
What about a cheaper route?
Well, this one is 20 kms longer than the first alternative, 10 minutes slower than the second, but cheaper.

Route planning in road maps



Let me now present an example of a typical route planning problem. I have finished an important meeting and I will use my fantastic route planning application to find me the shortest path back home.
The length of the shortest path from all possible routes is 90 km.
Wait! There's a traffic jam on that route and the travel time would be an hour and a half.
Maybe, what I do want is not the shortest but the fastest route.
This route is 10 kms longer but half an hour faster, it seems to me a good trade-off.
or not, there's a toll and this route is 8 euros more expensive than the last.
What about a cheaper route?
Well, this one is 20 kms longer than the first alternative, 10 minutes slower than the second, but cheaper.

Route planning in road maps

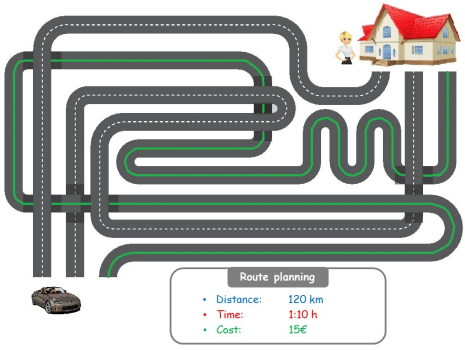


Let me now present an example of a typical route planning problem. I have finished an important meeting and I will use my fantastic route planning application to find me the shortest path back home.
The length of the shortest path from all possible routes is 90 km.
Wait! There's a traffic jam on that route and the travel time would be an hour and a half.
Maybe, what I do want is not the shortest but the fastest route.
This route is 10 kms longer but half an hour faster, it seems to me a good trade-off.
or not, there's a toll and this route is 8 euros more expensive than the last.
What about a cheaper route?
Well, this one is 20 kms longer than the first alternative, 10 minutes slower than the second, but cheaper.

Route planning in road maps

Let me now present an example of a typical route planning problem. I have finished an important meeting and I will use my fantastic route planning application to find me the shortest path back home.
The length of the shortest path from all possible routes is 90 km.
Wait! There's a traffic jam on that route and the travel time would be an hour and a half.
Maybe, what I do want is not the shortest but the fastest route.
This route is 10 kms longer but half an hour faster, it seems to me a good trade-off.
or not, there's a toll and this route is 8 euros more expensive than the last.
What about a cheaper route?
Well, this one is 20 kms longer than the first alternative, 10 minutes slower than the second, but cheaper.

Route planning in road maps



Let me now present an example of a typical route planning problem. I have finished an important meeting and I will use my fantastic route planning application to find me the shortest path back home.
The length of the shortest path from all possible routes is 90 km.
Wait! There's a traffic jam on that route and the travel time would be an hour and a half.
Maybe, what I do want is not the shortest but the fastest route.
This route is 10 kms longer but half an hour faster, it seems to me a good trade-off.
or not, there's a toll and this route is 8 euros more expensive than the last.
What about a cheaper route?
Well, this one is 20 kms longer than the first alternative, 10 minutes slower than the second, but cheaper.

Multicriteria Search Problem (MSP)

Definition

Dominance or Pareto preference $\prec$:

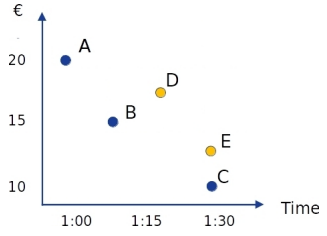
$$\vec{y} \prec \vec{y'} \quad \Leftrightarrow \quad \forall i \ (1 \leq \ i \leq \ q) \quad y_i \leq y'_i \ \wedge \ \vec{y} \neq \vec{y'}$$

where $\vec{y} = (y_1, y_2, ..., y_q)$ represents a solution path in the cost space.

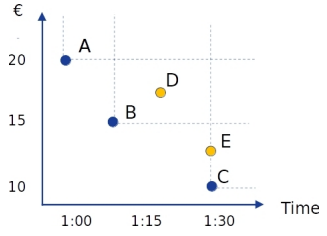
Example

Route cost representation: $\vec{y} = (y_1, y_2, y_3) \ \rightarrow \ \vec{y} = (120, 70, 15)$

Pareto optimality



Pareto optimality

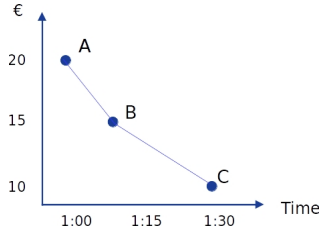


Multicriteria Search Problem (MSP)

Definition

The solution to a MSP, without any knowledge of the user preferences, is the set of non-dominated solution paths, or Pareto frontier.

Pareto optimality



Goal Programming

Definition

Goal Programming (GP) is one of the most successful techniques to solve multicriteria problems. It is also a natural way to express user preferences as targets or aspiration levels: $y_i \leq t_i$.

Definition

Lexicographic Goal Programming (LGP) ranks all goals in order of importance, dividing them into lexicographic levels of importance.

Definition

The deviation from a goal in a minimization problem represents a positive distance between the i-th goal and its aspiration level:
$d_i = \max(0, y_i - t_i)$.

Goal Programming

Example

Three criteria (economic cost (euros), travel time (minutes), and distance (km)) grouped in two priority levels where level 1 is infinitely more important than level 2. The importance of the criteria within a level is defined by weights as:

$$\begin{array}{lll} \text{Level 1:} & \textit{cost} \leq 15, & w_1 = 0.5 \\ & \textit{time} \leq 75, & w_2 = 0.5 \\ \text{Level 2:} & \textit{distance} \leq 120, & w_3 = 1. \end{array}$$

Example

Let's assume a solution path with cost $\vec{y} = (10, 60, 150)$,
its deviation from those goals is $\vec{d} = (0,\ 30) \rightarrow$
$(\max(0, (10 - 15) \times 0.5) + \max(0, (60 - 75) \times 0.5),\ (150 - 120) \times 1)$

Let me assume 3 criteria grouped in two priority levels, where the first level is infinitely more important than the second. Goals 2 and 3 share the same importance within level 2. Let's see some examples of vectors and their deviation: the first one (16,16,20) does not have any deviation from goals, none of objectives have overpassed the goals. This vector, however, has deviation of 4 units in the first level. In the last example, we observe deviation in both levels.

Goal Programming

Example

Three criteria (economic cost (euros), travel time (minutes), and distance (km)) grouped in two priority levels where level 1 is infinitely more important than level 2. The importance of the criteria within a level is defined by weights as:

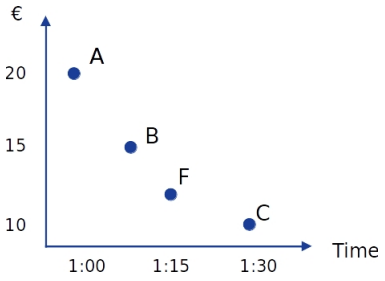
$$\begin{array}{lll} \text{Level 1:} & \textit{cost} \leq 15, & w_1 = 0.5 \\ & \textit{time} \leq 75, & w_2 = 0.5 \\ \text{Level 2:} & \textit{distance} \leq 120, & w_3 = 1. \end{array}$$

Example

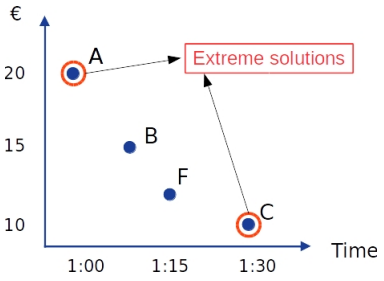
Let's assume a solution path with cost $\vec{y} = (10, 60, 150)$,
its deviation from those goals is $\vec{d} = (0,\ 30) \rightarrow$
$(\max(0, (10 - 15) \times 0.5) + \max(0, (60 - 75) \times 0.5),\ (150 - 120) \times 1)$

Let me assume 3 criteria grouped in two priority levels, where the first level is infinitely more important than the second. Goals 2 and 3 share the same importance within level 2. Let's see some examples of vectors and their deviation: the first one (16,16,20) does not have any deviation from goals, none of objectives have overpassed the goals. This vector, however, has deviation of 4 units in the first level. In the last example, we observe deviation in both levels.

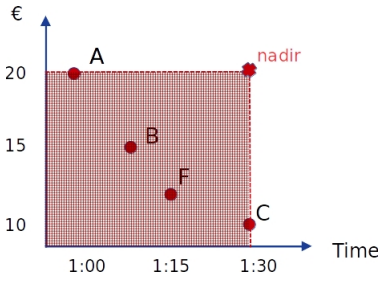
Lexicographic goals



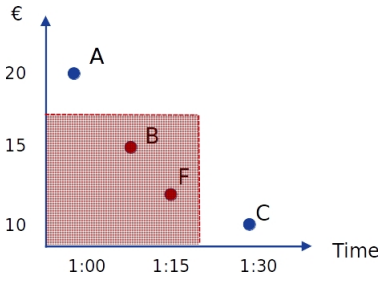
Lexicographic goals



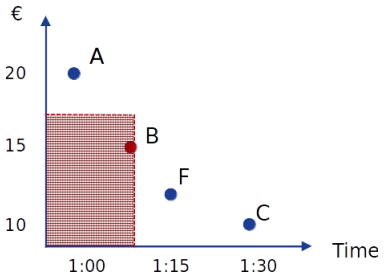
Lexicographic goals



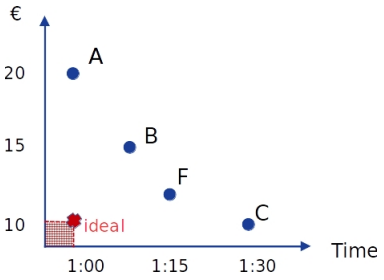
Lexicographic goals



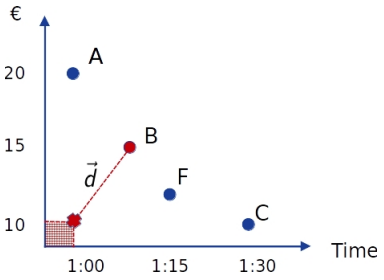
Lexicographic goals



Lexicographic goals



Lexicographic goals



LEXGO*

Definition

We define **lexicographic goal preferences** ($\prec_G$) as a partial order relation,

$$\vec{y} \prec_G \vec{y'} \quad \Leftrightarrow \quad \vec{d}(\vec{y}) \prec_L \vec{d}(\vec{y'}) \quad \vee \quad (\vec{d}(\vec{y}) = \vec{d}(\vec{y'}) \quad \wedge \quad \vec{y} \prec \vec{y'})$$

In order to return the set of efficient solutions we had to define, first, which are those solutions. We define the partial order relation lexicographic goal preferences as: one vector dominates another according to the goals if either its deviation is lexicographically better or their deviations are equal but that the vector dominates the alternative.

MSP with lexicographic goals

Definition

The solution to a MSP with lexicographic goals is the set of non-dominated solution paths that satisfy the goals, or the subset that minimizes deviation if these cannot be fully satisfied.

Outline

Research goals

Tackle the MSP with lexicographic goals from two algorithmic perspectives:

This dissertation tackles the MSP with lexicographic goals. Two different approaches are proposed to solve such problem, considering when the goals are taken into account: A priori, where we devise a new specifically design algorithm to return goal-optimal solutions, and a posteriori, performing a full multiobjective search and then extracting the goal-optimal solutions from the Pareto frontier.

Research goals

Tackle the MSP with lexicographic goals from two algorithmic perspectives:

A posteriori Preferences of the decision maker are provided after the execution of the algorithm.

A priori Preferences of the decision maker are provided before the execution of the algorithm.

This dissertation tackles the MSP with lexicographic goals. Two different approaches are proposed to solve such problem, considering when the goals are taken into account: A priori, where we devise a new specifically design algorithm to return goal-optimal solutions, and a posteriori, performing a full multiobjective search and then extracting the goal-optimal solutions from the Pareto frontier.

Research goals

Tackle the MSP with lexicographic goals from two algorithmic perspectives:

A posteriori Preferences of the decision maker are provided after the execution of the algorithm.

A priori Preferences of the decision maker are provided before the execution of the algorithm.

- **A posteriori** algorithms obtain the whole Pareto frontier and then, extract the goal-optimal solutions.
- **A priori** algorithms will return only goal-optimal solutions.

This dissertation tackles the MSP with lexicographic goals. Two different approaches are proposed to solve such problem, considering when the goals are taken into account: A priori, where we devise a new specifically design algorithm to return goal-optimal solutions, and a posteriori, performing a full multiobjective search and then extracting the goal-optimal solutions from the Pareto frontier.

Outline

Properties of multicriteria algorithms

Definition

An algorithm is admissible if it guarantees to find the optimal solution to the problem.

We will analyze two formal properties for the algorithms we have proposed: the admissibility, or in other words, do they find the whole set of Pareto or goal-optimal solutions? And the second one, how efficient are these algorithms according to the number of explored labels?

Properties of multicriteria algorithms

Definition

An algorithm is admissible if it guarantees to find the optimal solution to the problem.

Definition

We will measure the efficiency of algorithms according to the number of explored labels to find the solution.

We will analyze two formal properties for the algorithms we have proposed: the admissibility, or in other words, do they find the whole set of Pareto or goal-optimal solutions? And the second one, how efficient are these algorithms according to the number of explored labels?

Outline

A posteriori algorithms

Extensions of A^* to the multiobjective case to calculate the Pareto frontier:

MOA*	(Stewart & White, 1991)
TC	(Tung & Chew, 1992)
NAMOA*	(Mandow & Pérez de la Cruz, 2005)

All can be used with heuristic functions as lower bounds

A posteriori algorithms

Extensions of A^* to the multiobjective case to calculate the Pareto frontier:

MOA*	(Stewart & White, 1991) → pathological behaviour
TC	(Tung & Chew, 1992) → less efficient than NAMOA*
NAMOA*	(Mandow & Pérez de la Cruz, 2005)

All can be used with heuristic functions as lower bounds

NAMOA*

NAMOA* properties

Analogously to A^*:

- Admissible when provided with a consistent heuristic function.
- It explores an optimal number of labels in its class.
- It improves its efficiency with more informed lower bounds.

Since this algorithm is the base of our research work, let's name first its formal properties, which are analogous to A^* properties: NAMOA* is an admissible algorithm when provided with consistent lower bounds, explores an optimal number of labels in its class, and improves its efficiency with more informed lower bounds.

NAMOA*

Relevant features of NAMOA*:

- Label selection policy
- Two sets of labels for each node n: $G_{op}(n)$ and $G_{cl}(n)$.
- A set COSTS of solutions.
- Discarding rules of dominated paths

On the other hand, the main features of NAMOA* can be enumerated as: a flexible label selection policy, two sets of non-dominated labels store the open and closed nodes that reach each node, a set of non-dominated solutions, and the three procedures used by NAMOA* to discard dominated alternatives: checking against Gop, Gcl, and COSTS.

Label selection policy

NAMOA* can be used with any path selection policy that assures the best label according to that policy is a non-dominated label.

NAMOA* uses a priority queue to sort the open paths, like A^* selects the minimum f-value from the queue, NAMOA* uses any label selection policy that can assure the best label according to that policy is always a non-dominated label. In this work, we have considered two different orders, lexicographic and linear aggregation. The lexicographic order selects the path with minimum c1 and to break ties would use c2, meanwhile, the linear aggregation order selects first the path which aggregation of both c1 and c2 values is minimum. Thus, the label selection policy leads NAMOA* to explore the search space in a different order.

Label selection policy

NAMOA* can be used with any path selection policy that assures the best label according to that policy is a non-dominated label.

- Lexicographic order
- Linear aggregation order

NAMOA* uses a priority queue to sort the open paths, like A^* selects the minimum f-value from the queue, NAMOA* uses any label selection policy that can assure the best label according to that policy is always a non-dominated label. In this work, we have considered two different orders, lexicographic and linear aggregation. The lexicographic order selects the path with minimum c1 and to break ties would use c2, meanwhile, the linear aggregation order selects first the path which aggregation of both c1 and c2 values is minimum. Thus, the label selection policy leads NAMOA* to explore the search space in a different order.

Label selection policy

NAMOA* can be used with any path selection policy that assures the best label according to that policy is a non-dominated label.

- Lexicographic order

$$\vec{y} \prec_L \vec{y'} \quad \Leftrightarrow \quad \exists j \; (1 \leq \; i \leq \; q) \; y_j < y'_j \; \wedge \; \forall i < j \;\; y_i = y'_i.$$

- Linear aggregation order

NAMOA* uses a priority queue to sort the open paths, like A^* selects the minimum f-value from the queue, NAMOA* uses any label selection policy that can assure the best label according to that policy is always a non-dominated label. In this work, we have considered two different orders, lexicographic and linear aggregation. The lexicographic order selects the path with minimum c1 and to break ties would use c2, meanwhile, the linear aggregation order selects first the path which aggregation of both c1 and c2 values is minimum. Thus, the label selection policy leads NAMOA* to explore the search space in a different order.

Label selection policy

NAMOA* can be used with any path selection policy that assures the best label according to that policy is a non-dominated label.

- Lexicographic order

$$\vec{y} \prec_L \vec{y'} \quad \Leftrightarrow \quad \exists j \; (1 \leq \; i \leq \; q) \; y_j < y'_j \; \wedge \; \forall i < j \;\; y_i = y'_i.$$

- Linear aggregation order

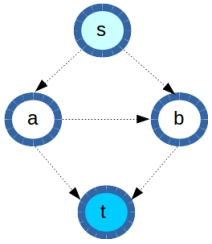
$$\vec{y} \prec_{lin} \vec{y'} \; \Leftrightarrow \; \sum_i y_i \; < \; \sum_i y'_i \;\; , \; 1 \; \leq \; i \; \leq \; q$$

NAMOA* uses a priority queue to sort the open paths, like A^* selects the minimum f-value from the queue, NAMOA* uses any label selection policy that can assure the best label according to that policy is always a non-dominated label. In this work, we have considered two different orders, lexicographic and linear aggregation. The lexicographic order selects the path with minimum c1 and to break ties would use c2, meanwhile, the linear aggregation order selects first the path which aggregation of both c1 and c2 values is minimum. Thus, the label selection policy leads NAMOA* to explore the search space in a different order.

Discarding dominated paths on NAMOA*

NAMOA* discards early in the search those paths that will not lead to non-dominated solutions.

- Op-pruning
- Cl-pruning
- Filtering

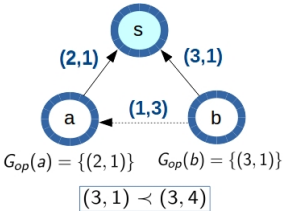


As we said, NAMOA* discards dominated labels by three procedures: Let's see them through different examples: there is a new path found to n' with cost (3,6). This path is discarded by a path with cost (3,4) already closed in n'. Let's now analyze another new path with cost (5,3), this will be pruned by (5,2), an open path already found in n'. Finally, let's assume a path to be expanded in n, which f-value is (9,9), this path will be filtered by either (8,8), or (9,7), solutions path already found, due to this path can not lead to a non-dominated solution.

Discarding dominated paths on NAMOA*

NAMOA* discards early in the search those paths that will not lead to non-dominated solutions.

- Op-pruning
- Cl-pruning
- Filtering

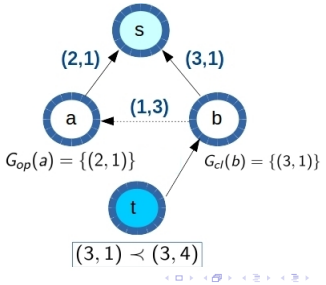


As we said, NAMOA* discards dominated labels by three procedures: Let's see them through different examples: there is a new path found to n' with cost (3,6). This path is discarded by a path with cost (3,4) already closed in n'. Let's now analyze another new path with cost (5,3), this will be pruned by (5,2), an open path already found in n'. Finally, let's assume a path to be expanded in n, which f-value is (9,9), this path will be filtered by either (8,8), or (9,7), solutions path already found, due to this path can not lead to a non-dominated solution.

Discarding dominated paths on NAMOA*

NAMOA* discards early in the search those paths that will not lead to non-dominated solutions.

- Op-pruning
- Cl-pruning
- Filtering

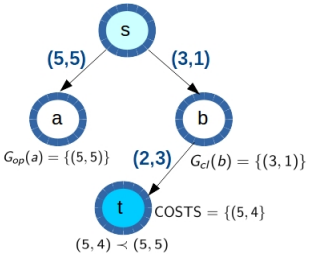


As we said, NAMOA* discards dominated labels by three procedures: Let's see them through different examples: there is a new path found to n' with cost (3,6). This path is discarded by a path with cost (3,4) already closed in n'. Let's now analyze another new path with cost (5,3), this will be pruned by (5,2), an open path already found in n'. Finally, let's assume a path to be expanded in n, which f-value is (9,9), this path will be filtered by either (8,8), or (9,7), solutions path already found, due to this path can not lead to a non-dominated solution.

Discarding dominated paths on NAMOA*

NAMOA* discards early in the search those paths that will not lead to non-dominated solutions.

- Op-pruning
- Cl-pruning
- Filtering



As we said, NAMOA* discards dominated labels by three procedures: Let's see them through different examples: there is a new path found to n' with cost (3,6). This path is discarded by a path with cost (3,4) already closed in n'. Let's now analyze another new path with cost (5,3), this will be pruned by (5,2), an open path already found in n'. Finally, let's assume a path to be expanded in n, which f-value is (9,9), this path will be filtered by either (8,8), or (9,7), solutions path already found, due to this path can not lead to a non-dominated solution.

Outline

A priori algorithms

Is there any specifically devised algorithm for lexicographic goals?

METAL-A* (Mandow & Pérez de la Cruz, 2001)
Based on MOA* $\longrightarrow$ pathological behaviour

Now we turn our attention to the second possibility we named, a priori algorithms. The state of the art here was METAL-A* which was based on MOA*. According to recent studies, we can consider this algorithm deprecated and hence, we will propose a new approach based on NAMOA*.

A priori algorithms

Is there any specifically devised algorithm for lexicographic goals?

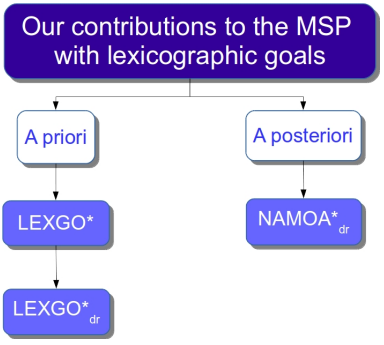
METAL-A* (Mandow & Pérez de la Cruz, 2001)
Based on MOA* $\longrightarrow$ pathological behaviour

$\longrightarrow$ Then, let's propose a new approach based on NAMOA*

Now we turn our attention to the second possibility we named, a priori algorithms. The state of the art here was METAL-A* which was based on MOA*. According to recent studies, we can consider this algorithm deprecated and hence, we will propose a new approach based on NAMOA*.

Outline

Algorithmic contributions



Outline

Lexicographic goal preferences

Important!

Bellman's Principle of Optimality, **an optimal path is made up of suboptimal paths**, holds for multiobjective search problems, but it does not hold for lexicographic goal preferences!

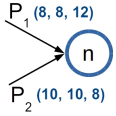
Let's remark a fact, the principle of optimality does hold for MSP but it does not hold for lexicographic goals. We need to keep that in mind when we present the new discarding rules for these problems.

Lexicographic goal preferences

Example

Two paths P_1 and P_2 reach some node n with costs $\vec{g}(P_1) = (8, 8, 12)$ and $\vec{g}(P_2) = (10, 10, 8)$, respectively. The user goals are defined as follows:

$$\begin{array}{lll} \text{Level 1:} & g_1 \leq 10, & w_1 = 0.5 \\ & g_2 \leq 10, & w_2 = 0.5 \\ \text{Level 2:} & g_3 \leq 10, & w_3 = 1. \end{array}$$



Example

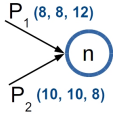
The deviation from goals for P_1 and P_2 is $\vec{d}(P_1) = (0, 2)$ and $\vec{d}(P_2) = (0, 0)$, respectively. Thus, $P_2 \prec_G P_1$.

Lexicographic goal preferences

Example

Two paths P_1 and P_2 reach some node n with costs $\vec{g}(P_1) = (8, 8, 12)$ and $\vec{g}(P_2) = (10, 10, 8)$, respectively. The user goals are defined as follows:

$$\begin{array}{lll} \text{Level 1:} & g_1 \leq 10, & w_1 = 0.5 \\ & g_2 \leq 10, & w_2 = 0.5 \\ \text{Level 2:} & g_3 \leq 10, & w_3 = 1. \end{array}$$



Example

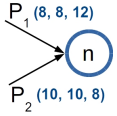
The deviation from goals for P_1 and P_2 is $\vec{d}(P_1) = (0, 2)$ and $\vec{d}(P_2) = (0, 0)$, respectively. Thus, $P_2 \prec_G P_1$.

Lexicographic goal preferences

Example

Two paths P_1 and P_2 reach some node n with costs $\vec{g}(P_1) = (8, 8, 12)$ and $\vec{g}(P_2) = (10, 10, 8)$, respectively. The user goals are defined as follows:

$$\begin{array}{lll} \text{Level 1:} & g_1 \leq 10, & w_1 = 0.5 \\ & g_2 \leq 10, & w_2 = 0.5 \\ \text{Level 2:} & g_3 \leq 10, & w_3 = 1. \end{array}$$



Example

The deviation from goals for P_1 and P_2 is $\vec{d}(P_1) = (0, 2)$ and $\vec{d}(P_2) = (0, 0)$, respectively. Thus, $P_2 \prec_G P_1$.

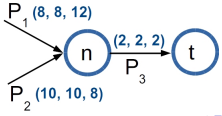
Lexicographic goal preferences

Example

Let's now consider there is only one additional path P_3 to the destination node t, such as $\vec{g}(P_3)) = (2, 2, 2)$.
Therefore,

- $\vec{g}(P_1P_3)) = (10, 10, 14)$ and $\vec{g}(P_2P_3)) = (12, 12, 10)$.
- $\vec{d}(P_1P_3) = (0, 4)$ and $\vec{d}(P_2P_3) = (2, 0)$.
- Now, $P_1P_3 \prec_G P_2P_3$

We have pruned an optimal path!



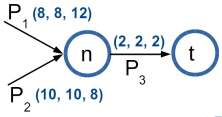
Lexicographic goal preferences

Example

Let's now consider there is only one additional path P_3 to the destination node t, such as $\vec{g}(P_3)) = (2, 2, 2)$.
Therefore,

- $\vec{g}(P_1P_3)) = (10, 10, 14)$ and $\vec{g}(P_2P_3)) = (12, 12, 10)$.
- $\vec{d}(P_1P_3) = (0, 4)$ and $\vec{d}(P_2P_3) = (2, 0)$.
- Now, $P_1P_3 \prec_G P_2P_3$

We have pruned an optimal path!



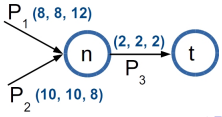
Lexicographic goal preferences

Example

Let's now consider there is only one additional path P_3 to the destination node t, such as $\vec{g}(P_3)) = (2, 2, 2)$.
Therefore,

- $\vec{g}(P_1P_3)) = (10, 10, 14)$ and $\vec{g}(P_2P_3)) = (12, 12, 10)$.
- $\vec{d}(P_1P_3) = (0, 4)$ and $\vec{d}(P_2P_3) = (2, 0)$.
- Now, $P_1P_3 \prec_G P_2P_3$

We have pruned an optimal path!

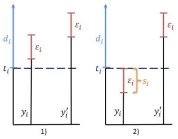


Lexicographic goal preferences

Example

Let's now assume paths P_1 and P_2 have costs $\vec{g}(P_1) = (12, 8, 16)$ and $\vec{g}(P_2) = (12, 12, 6)$, respectively. The user goals are now *different* and defined as follows:

$$\begin{array}{lll} \text{Level 1:} & g_1 \leq 10, & w_1 = 1 \\ \text{Level 2:} & g_2 \leq 10, & w_2 = 0.5 \\ & g_3 \leq 10, & w_3 = 0.5. \end{array}$$



Example

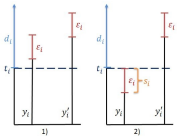
The deviation from goals for P_1 and P_2 is now $\vec{d}(P_1) = (2, 3)$ and $\vec{d}(P_2) = (2, 1)$, respectively. $P_2 \prec_G P_1$, but, can P_1 be safely pruned?

Lexicographic goal preferences

Example

Let's now assume paths P_1 and P_2 have costs $\vec{g}(P_1) = (12, 8, 16)$ and $\vec{g}(P_2) = (12, 12, 6)$, respectively. The user goals are now *different* and defined as follows:

$$\begin{array}{lll} \text{Level 1:} & g_1 \leq 10, & w_1 = 1 \\ \text{Level 2:} & g_2 \leq 10, & w_2 = 0.5 \\ & g_3 \leq 10, & w_3 = 0.5. \end{array}$$



Example

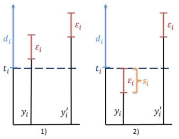
The deviation from goals for P_1 and P_2 is now $\vec{d}(P_1) = (2, 3)$ and $\vec{d}(P_2) = (2, 1)$, respectively. $P_2 \prec_G P_1$, but, can P_1 be safely pruned?

Lexicographic goal preferences

Example

Let's now assume paths P_1 and P_2 have costs $\vec{g}(P_1) = (12, 8, 16)$ and $\vec{g}(P_2) = (12, 12, 6)$, respectively. The user goals are now *different* and defined as follows:

$$\begin{array}{lll} \text{Level 1:} & g_1 \leq 10, & w_1 = 1 \\ \text{Level 2:} & g_2 \leq 10, & w_2 = 0.5 \\ & g_3 \leq 10, & w_3 = 0.5. \end{array}$$



Example

The deviation from goals for P_1 and P_2 is now $\vec{d}(P_1) = (2, 3)$ and $\vec{d}(P_2) = (2, 1)$, respectively. $P_2 \prec_G P_1$, but, can P_1 be safely pruned?

Discarding labels by deviation

Deviation-based pruning ($\prec_P$)

Slack variable

$$s_k = \max(0, t_k - y_k)$$

Cross slacks

$$\delta_j(\vec{y}, \vec{y'}) = \sum_{k \in I_j} w_k \times \max(0, s'_k - s_k)$$

Example

Given $\vec{g}(P_1) = (12, 8, 16)$ and $\vec{g}(P_2) = (12, 12, 6)$,
where $\vec{d}(P_1) = (2, 3)$ and $\vec{d}(P_2) = (2, 1)$:

- $s_2(P_1) = (10 - 8) \times 0.5 = 1.$
- $s_3(P_2) = (10 - 6) \times 0.5 = 2.$
- $\delta_2(P_1, P_2) = 2 - 1 = 1,$

Finally, $d_2(P_2) - d_2(P_1) = 3 - 1 = 2 > 1.$

Discarding labels by deviation

Deviation-based pruning ($\prec_P$)

Slack variable

$s_k = \max(0, t_k - y_k)$

Cross slacks

$\delta_j(\vec{y}, \vec{y'}) = \sum_{k \in I_j} w_k \times \max(0, s'_k - s_k)$

Example

Given $\vec{g}(P_1) = (12, 8, 16)$ and $\vec{g}(P_2) = (12, 12, 6)$,
where $\vec{d}(P_1) = (2, 3)$ and $\vec{d}(P_2) = (2, 1)$:

- $s_2(P_1) = (10 - 8) \times 0.5 = 1.$
- $s_3(P_2) = (10 - 6) \times 0.5 = 2.$
- $\delta_2(P_1, P_2) = 2 - 1 = 1,$

Finally, $d_2(P_2) - d_2(P_1) = 3 - 1 = 2 > 1.$

Discarding labels by deviation

Deviation-based pruning ($\prec_P$)

Slack variable

$$s_k = \max(0, t_k - y_k)$$

Cross slacks

$$\delta_j(\vec{y}, \vec{y'}) = \sum_{k \in I_j} w_k \times \max(0, s'_k - s_k)$$

Example

Given $\vec{g}(P_1) = (12, 8, 16)$ and $\vec{g}(P_2) = (12, 12, 6)$,
where $\vec{d}(P_1) = (2, 3)$ and $\vec{d}(P_2) = (2, 1)$:

- $s_2(P_1) = (10 - 8) \times 0.5 = 1.$
- $s_3(P_2) = (10 - 6) \times 0.5 = 2.$
- $\delta_2(P_1, P_2) = 2 - 1 = 1,$

Finally, $d_2(P_2) - d_2(P_1) = 3 - 1 = 2 > 1.$

Discarding labels by deviation

Deviation-based pruning ($\prec_P$)

Slack variable

$s_k = \max(0, t_k - y_k)$

Cross slacks

$\delta_j(\vec{y}, \vec{y'}) = \sum_{k \in I_j} w_k \times \max(0, s'_k - s_k)$

Example

Given $\vec{g}(P_1) = (12, 8, 16)$ and $\vec{g}(P_2) = (12, 12, 6)$,
where $\vec{d}(P_1) = (2, 3)$ and $\vec{d}(P_2) = (2, 1)$:

- $s_2(P_1) = (10 - 8) \times 0.5 = 1$.
- $s_3(P_2) = (10 - 6) \times 0.5 = 2$.
- $\delta_2(P_1, P_2) = 2 - 1 = 1$,

Finally, $d_2(P_2) - d_2(P_1) = 3 - 1 = 2 > 1$.

Discarding labels by deviation

Deviation-based pruning ($\prec_P$)

Slack variable

$$s_k = \max(0, t_k - y_k)$$

Cross slacks

$$\delta_j(\vec{y}, \vec{y'}) = \sum_{k \in I_j} w_k \times \max(0, s'_k - s_k)$$

Example

Given $\vec{g}(P_1) = (12, 8, 16)$ and $\vec{g}(P_2) = (12, 12, 6)$,
where $\vec{d}(P_1) = (2, 3)$ and $\vec{d}(P_2) = (2, 1)$:

- $s_2(P_1) = (10 - 8) \times 0.5 = 1.$
- $s_3(P_2) = (10 - 6) \times 0.5 = 2.$
- $\delta_2(P_1, P_2) = 2 - 1 = 1,$

Finally, $d_2(P_2) - d_2(P_1) = 3 - 1 = 2 > 1.$

Discarding labels by deviation

Deviation-based pruning ($\prec_P$)

Slack variable

$$s_k = \max(0, t_k - y_k)$$

Cross slacks

$$\delta_j(\vec{y}, \vec{y'}) = \sum_{k \in I_j} w_k \times \max(0, s'_k - s_k)$$

Example

Given $\vec{g}(P_1) = (12, 8, 16)$ and $\vec{g}(P_2) = (12, 12, 6)$,
where $\vec{d}(P_1) = (2, 3)$ and $\vec{d}(P_2) = (2, 1)$:

- $s_2(P_1) = (10 - 8) \times 0.5 = 1$.
- $s_3(P_2) = (10 - 6) \times 0.5 = 2$.
- $\delta_2(P_1, P_2) = 2 - 1 = 1$,

Finally, $d_2(P_2) - d_2(P_1) = 3 - 1 = 2 > 1$.

Discarding labels by deviation

Deviation-based pruning ($\prec_P$)

$$\vec{y} \prec_P \vec{y'} \quad \Leftrightarrow \quad \forall i < j \;\; (d_i(\vec{y}) = d_i(\vec{y'}) \quad \wedge \quad \delta_i(\vec{y}, \vec{y'}) = 0)) \quad \wedge$$
$$\exists j \; (d_j(\vec{y}) < d_j(\vec{y'}) \quad \wedge \quad \delta_j(\vec{y}, \vec{y'}) < d_j(\vec{y'}) - d_j(\vec{y})$$

Deviation-based filtering

$$\vec{d_B} \prec_L \vec{d}$$

Discarding labels by deviation

Deviation-based pruning ($\prec_P$)

$$\vec{y} \prec_P \vec{y'} \quad \Leftrightarrow \quad \forall i < j \;\; (d_i(\vec{y}) = d_i(\vec{y'}) \quad \wedge \quad \delta_i(\vec{y}, \vec{y'}) = 0)) \quad \wedge$$
$$\exists j \; (d_j(\vec{y}) < d_j(\vec{y'}) \quad \wedge \quad \delta_j(\vec{y}, \vec{y'}) < d_j(\vec{y'}) - d_j(\vec{y})$$

Deviation-based filtering

$$\vec{d_B} \prec_L \vec{d}$$

Discarding labels by deviation

Deviation-based pruning ($\prec_P$)

$$\vec{y} \prec_P \vec{y'} \quad \Leftrightarrow \quad \forall i < j \;\; (d_i(\vec{y}) = d_i(\vec{y'}) \quad \wedge \quad \delta_i(\vec{y}, \vec{y'}) = 0)) \quad \wedge$$
$$\exists j \; (d_j(\vec{y}) < d_j(\vec{y'}) \quad \wedge \quad \delta_j(\vec{y}, \vec{y'}) < d_j(\vec{y'}) - d_j(\vec{y})$$

Deviation-based filtering

$$\vec{d_B} \prec_L \vec{d}$$

LEXGO*

In addition to the pruning and filtering rules of NAMOA*, LEXGO* also uses its own rules to prune and filter paths by deviation.

LEXGO*

LEXGO* shares NAMOA* formal properties

- The new deviation-based pruning and filtering rules do not discard goal-optimal solutions.
- LEXGO* always returns the set of all goal-optimal solutions.
- LEXGO* expands a subset of the labels expanded by NAMOA*.

Outline

T-discarding

Limiting factor in MSP

Runtime, rather than memory, is the limiting factor in multicriteria search algorithms performance, as well as in the size of problems that can be practically solved.

Limiting factor in MSP (II)

The majority of the algorithm runtime can be attributed to dominance checks against G_{op}, G_{cl} and COSTS to discard dominated paths.

T-discarding arose due to time rather memory is nowadays the limiting factor in multicriteria search algorithms performance. This runtime can be mostly attributed to dominance checks to discard paths. In this manner, if we speed up these checks, we will speed up the algorithms.

T-discarding

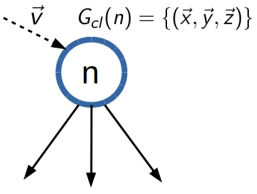
Let's suppose a typical scenario where a new path with cost $\vec{v}$ reaches some node n. This node has already three closed paths with costs $(\vec{x}, \vec{y}, \vec{z})$. Let's now check if $\vec{v}$ is dominated by any vector in $G_{cl}(n)$:

Example

- $\vec{x} \prec \vec{v}$? $\rightarrow$ $x_1 \leq v_1$, $x_2 \leq v_2$ ...
- $\vec{y} \prec \vec{v}$? $\rightarrow$ $y_1 \leq v_1$, $y_2 \leq v_2$ ...
- $\vec{z} \prec \vec{v}$? $\rightarrow$ $z_1 \leq v_1$, $z_2 \leq v_2$ ...

Worst-case scenario:

- 3 vector comparisons
- 9 scalar comparisons



T-discarding

Definition

T-discarding (*truncated discarding*) is a dimensionality reduction technique to speed up dominance checks.

This is the definition of our proposal, called t-discarding, or truncated distarding. This proposal is based on a dimensionality reduction technique to speed up dominance checks.

T-discarding

Assumptions

To apply t-discarding an algorithm must have:

- Lexicographic order as label selection policy.
- One consistent heuristic function as lower bound.

Do we have some assumptions before applying this technique? Yes, two assumptions, the first one is to use a lexicographic order as label selection policy, and the second one, is to use a consistent heuristic function as lower bound. We can depict here the main idea. This is the representation of 3 vectors x, y, and z. What happens if we ignore the first objective and we represent the three truncated vectors only according to the second and third objectives? We observe that the truncated vector x dominates the other two...

T-discarding

Assumptions

To apply t-discarding an algorithm must have:

- Lexicographic order as label selection policy.
- One consistent heuristic function as lower bound.

Example

When lexicographic order is used to select open paths:

- $\rightarrow (3, 5, 7)$
- $\rightarrow (4, 6, 4)$
- $\rightarrow (4, 6, 7)$
- $\rightarrow (6, 2, 4)$

Do we have some assumptions before applying this technique? Yes, two assumptions, the first one is to use a lexicographic order as label selection policy, and the second one, is to use a consistent heuristic function as lower bound. We can depict here the main idea. This is the representation of 3 vectors x, y, and z. What happens if we ignore the first objective and we represent the three truncated vectors only according to the second and third objectives? We observe that the truncated vector x dominates the other two...

T-discarding

Assumptions

To apply t-discarding an algorithm must have:

- Lexicographic order as label selection policy.
- One consistent heuristic function as lower bound.

Example

When lexicographic order is used to select open paths:

- → (3, -, -)
- → (4, -, -)
- → (4, -, -)
- → (6, -, -)

Do we have some assumptions before applying this technique? Yes, two assumptions, the first one is to use a lexicographic order as label selection policy, and the second one, is to use a consistent heuristic function as lower bound. We can depict here the main idea. This is the representation of 3 vectors x, y, and z. What happens if we ignore the first objective and we represent the three truncated vectors only according to the second and third objectives? We observe that the truncated vector x dominates the other two...

T-discarding

Assumptions

To apply t-discarding an algorithm must have:

- Lexicographic order as label selection policy.
- One consistent heuristic function as lower bound.

Property

Assuming the last selected open path had cost $\vec{c} = c_1, c_2, ..., c_q$, the next path selected to be open with cost $\vec{c'} = c'_1, c'_2, ..., c'_q$ must have $c'_1 \geq c_1$!

Do we have some assumptions before applying this technique? Yes, two assumptions, the first one is to use a lexicographic order as label selection policy, and the second one, is to use a consistent heuristic function as lower bound. We can depict here the main idea. This is the representation of 3 vectors x, y, and z. What happens if we ignore the first objective and we represent the three truncated vectors only according to the second and third objectives? We observe that the truncated vector x dominates the other two...

T-discarding

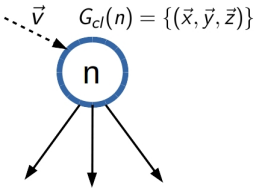
Let's suppose a typical scenario where a new path with cost $\vec{v}$ reaches some node n. This node has already three closed paths with costs $(\vec{x}, \vec{y}, \vec{z})$. Let's now check if $\vec{v}$ is dominated by any vector in $G_{cl}(n)$:

Example

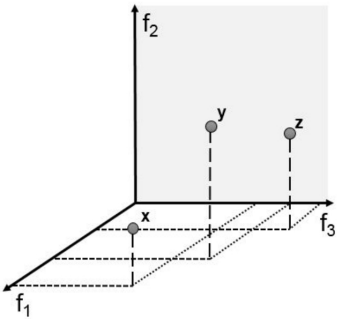
- $\vec{x} \prec \vec{v}$? $\rightarrow$ $x_1 \leq v_1$, $x_2 \leq v_2$ ...
- $\vec{y} \prec \vec{v}$? $\rightarrow$ $y_1 \leq v_1$, $y_2 \leq v_2$ ...
- $\vec{z} \prec \vec{v}$? $\rightarrow$ $z_1 \leq v_1$, $z_2 \leq v_2$ ...

Worst-case scenario:

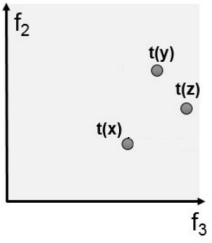
- 3 vector comparisons
- 9 scalar comparisons



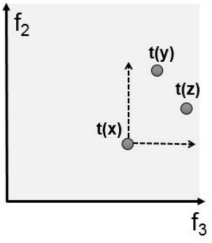
T-discarding



T-discarding



T-discarding



T-discarding

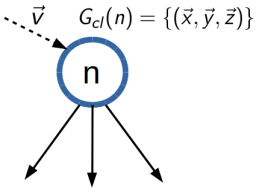
Let's suppose a typical scenario where a new path with cost $\vec{v}$ reaches some node n. This node has already three closed paths with costs $(\vec{x}, \vec{y}, \vec{z})$. Let's now check if $\vec{v}$ is dominated by any vector in $G_{cl}(n)$:

Example

- $\vec{x} \prec \vec{v}$? $\rightarrow$ $x_1 \leq v_1$, $x_2 \leq v_2$ ...
- $\vec{y} \prec \vec{v}$? $\rightarrow$ $y_1 \leq v_1$, $y_2 \leq v_2$ ...
- $\vec{z} \prec \vec{v}$? $\rightarrow$ $z_1 \leq v_1$, $z_2 \leq v_2$ ...

Worst-case scenario:

- 3 vector comparisons
- 9 scalar comparisons



T-discarding

Is a vector $\vec{v}$ dominated by any vector in the previous set $(\vec{x}, \vec{y}, \vec{z})$?

Standard discarding

$\vec{v}$ must be compared to $(\vec{x}, \vec{y}, \vec{z})$ to check if it is dominated.

T-discarding

The truncated vector $t(\vec{v})$ must be compared only to $t(\vec{x})$ to check if it is dominated.

T-discarding

Benefits of t-discarding

- Dominance checks can be done with vectors reduced in one dimension.
- The size of sets $T(G_{cl}(n))$ and $T(COSTS)$ is also reduced.

Important!

T-discarding only implies checking dominance against different sets:

- Very simple implementation.
- Any label discarded by standard pruning $\Longleftrightarrow$ is discarded by t-discarding.
- Does not affect any formal property of the algorithm.

Outline

NAMOA$^*_{dr}$

The applicability of t-discarding to NAMOA* is full and straightforward:

Cl-pruning and filtering are now performed against:

- $T(G_{cl}(n))$
- T(COSTS)

Outline

$\text{LEXGO}^*_{\text{dr}}$

The applicability of t-discarding to LEXGO* is partial:

Cl-pruning and filtering are now performed against:

- $T(G_{cl}(n))$
- T(COSTS)

but only when $\vec{d} = 0$.

Summary

	NAMOA*	LEXGO*	NAMOA$^*_{dr}$	LEXGO$^*_{dr}$
Op-pruning	$G_{op}(n)$	$G_{op}(n)$	$G_{op}(n)$	$G_{op}(n)$
Cl-pruning	$G_{cl}(n)$	$G_{cl}(n)$	$T(G_{cl}(n))$	$T(G_{cl}(n))$*
Filtering	COSTS	COSTS	T(COSTS)	T(COSTS)*

Outline

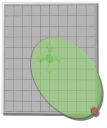
Empirical analyses

- Are these new algorithms effective?
- Will LEXGO* become faster when goals get more restrictive?
- Have we achieved improvements over NAMOA*? To what extent? Why?

Empirical analyses

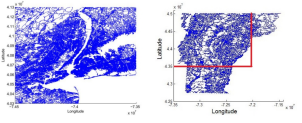
Random grids

- 100 × 100
- Solution depth from 20 to 100
- Random costs in range [1,10]



Realistic Road maps

- 20 random problems over New York City & Vermont State from 9th DIMACS Challenge
- Three integer costs
  - Distance
  - Travel time
  - Economic cost



This research work employed two different benchmarks, random grids and realistic road maps. The first experiments were performed on 100 times 100 grids with the start node located in the center of the grid and the target node at different solutions depths from 20 to 100 to the bottom right corner. Horizontal and vertical movements are possible in the grid, with arcs costs randomly generated from one to ten for each of the q criteria.

We also tested the algorithms in two maps from the 9th DIMACS Shortest Path Challenge, with three metrics, distance and travel time, and the economic cost, calculated as a function of fuel consumption, road types and tolls.

Outline

A priori

LEXGO* vs NAMOA*

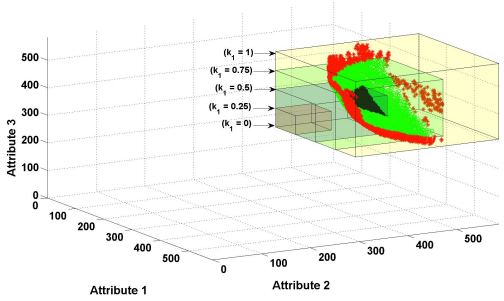


Figure: 3D Pareto frontier according to goal satisfiability.

In this figure we see the three dimensional Pareto frontier obtained by NAMOA* for a grid problem. Each of these cubes represent the goals imposed to LEXGO* in Class I experiments, from the ideal point, located on k_1 equals to zero, to the yellow cube that corresponds to goals situated in the nadir point. We can observe that in 0.5 a good percentage of the Pareto frontier would be returned by LEXGO*, and in 0.75 the majority of the non-dominated solutions

LEXGO* vs NAMOA*

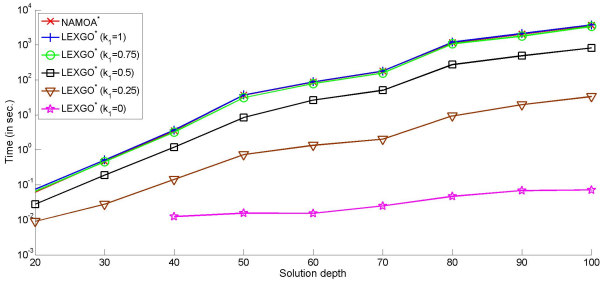


Figure: Runtimes of NAMOA* and LEXGO* in seconds (logarithmic scale).

LEXGO* vs NAMOA*

Table: Runtimes for $d = 100$ experiments.

	LEXGO$^*_{lex}$					
NAMOA$^*_{lex}$	1	0.75	0.5	0.25	0	k_1
Runtime (s)	%	%	%	%	%	
3,662.9	102.7	92.3	22.3	0.9	0.001	

Let's see now the runtime requirements. We also observe here the effectiveness of LEXGO*, 5 orders of magnitude faster for k1 equals to 0, two orders of magnitude for k1 in 0.25, 5 times faster for 0.5, slightly faster for 0.75 and with a little overhead when LEXGO* returns the whole Pareto frontier. In Class experiments we observe a similar behaviour, LEXGO* is somewhat around forty to fifty times faster than NAMOA* in these cases (0.75-0.1875, 0.5,0.125)

A priori

LEXGO*	vs	NAMOA*
$\text{LEXGO}^*_{\text{dr}}$	vs	LEXGO*

LEXGO$^*_{dr}$ vs LEXGO*

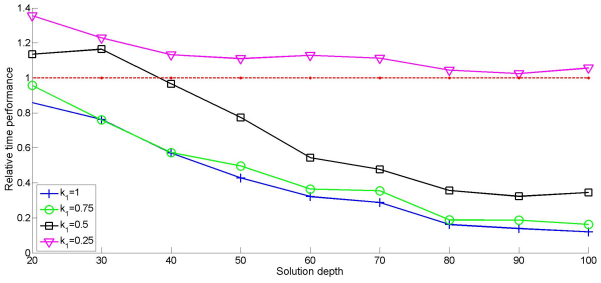


Figure: Relative runtime performance of LEXGO$^*_{dr}$ over LEXGO*.

LEXGO$^*_{dr}$ vs LEXGO*

Table: Runtimes in seconds for $d = 100$ experiments.

k_1	LEXGO$^*_{lex}$	LEXGO$^*_{dr}$	Speedup
1	3,763.78	254.40	14.81
0.75	3,381.33	300.86	11.27
0.5	819.28	282.49	2.90
0.25	33.47	35.37	0.94
0	0.07	0.10	0.7

Is t-discarding as effective in LEXGO* as it was in NAMOA*? To some extent, we see speed ups of 2 to 8 when goals can be satisfied, and obviously nothing we goals can not be satisfied, since t-discarding is not applied then.

A posteriori

$NAMOA^*_{dr}$ vs $NAMOA^*$

NAMOA$^*_{dr}$ vs NAMOA*

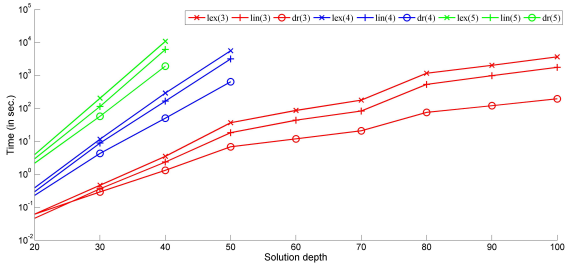


Figure: Runtimes for NAMOA$^*_{lex}$, NAMOA$^*_{lin}$, and NAMOA$^*_{dr}$ with $q = \{3, 4, 5\}$

We see these improvements here. For three, four or five objectives, the runtimes are greatly reduced and the improvement even grows with problem difficulty. In particular, for three objectives we see here a comparison between the new NAMOA* and both versions of standard NAMOA* with lexicographic and linear selection orders. I'd like to remark that like in other studies, NAMOA$^*_{lin}$ is somewhat twice as fast as NAMOA$^*_{lex}$, but both are meaningfully slower than NAMOA$^*_{dr}$.

A posteriori vs A priori

$\text{NAMOA}^*_{\text{dr}}$ vs $\text{LEXGO}^*_{\text{dr}}$

LEXGO$^*_{dr}$ vs NAMOA$^*_{dr}$

Table: Runtimes of LEXGO$^*_{dr}$ and NAMOA$^*_{dr}$.

	LEXGO$^*_{dr}$				
NAMOA$^*_{dr}$ Runtime (s)	($k_1 = 1$)	($k_1 = 0.75$)	($k_1 = 0.5$)	($k_1 = 0.25$)	($k_1 = 0$)
100%	129.7%	153.4%	144%	18%	0.04%

What about the final comparison between the two best alternatives? NAMOA$^*_{dr}$ is preferable whenever a big amount of non-dominated paths satisfy the goals, and LEXGO$^*_{dr}$ when a small number satisfy the goals or they can not be satisfied at all.

Outline

A priori

LEXGO* vs NAMOA*

LEXGO* vs NAMOA*

Table: Experiments for Vermont State map.

	LEXGO*$_{lex}$					
NAMOA*$_{lex}$	1	0.75	0.5	0.25	0	k_1
Avg. runtime (s)	%	%	%	%	%	
5,048.47	100.39	74.67	29.06	3.51	0.01	

The runtime requirements are also equivalent to the relative performance shown in grids, with greater advantage of LEXGO* when goals are more restrictive and less non-dominated paths satisfy the goals.

A priori

LEXGO*	vs	NAMOA*
$\text{LEXGO}^*_{\text{dr}}$	vs	LEXGO*

LEXGO$^*_{dr}$ vs LEXGO*

Table: Runtimes in seconds for Vermont State map.

	1	0.75	0.5	0.25	0	k_1
LEXGO$^*_{lex}$	5,068.00	3,769.51	1,467.00	177.33	0.03	
LEXGO$^*_{dr}$	94.90	83.59	57.38	157.02	0.03	

The same impressive improvements can be seen when we apply t-discarding to LEXGO* and goals can be satisfied, with 0.5, 0.75 and 1.

A posteriori

$NAMOA^*_{dr}$ vs $NAMOA^*$

NAMOA$^*_{dr}$ vs NAMOA*

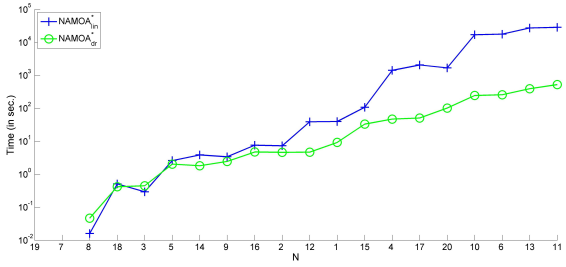


Figure: Runtimes of NAMOA$^*_{lin}$ and NAMOA$^*_{dr}$ for Vermont map problems.

A posteriori vs A priori

$NAMOA^*_{dr}$ vs $LEXGO^*_{dr}$

LEXGO$^*_{dr}$ vs NAMOA$^*_{dr}$

Table: Runtimes in seconds of LEXGO$^*_{dr}$ and NAMOA$^*_{dr}$ in Vermont State map problems.

	LEXGO$^*_{dr}$					
NAMOA$^*_{dr}$	1	0.75	0.5	0.25	0	k_1
84.66	94.89	83.59	57.38	157.02	0.03	

and the final comparison between the two best algorithmic alternatives gives us better results for NAMOA$^*_{dr}$ than we achieved in random grids, and the particular case here with 0.25. Even though the number of labels expanded is greatly reduced, t-discarding cannot be applied in that case, and therefore, LEXGO$^*_{dr}$ runtime is almost two times slower.

Why do the new algorithms have such a runtime performance?

LEXGO* vs NAMOA*

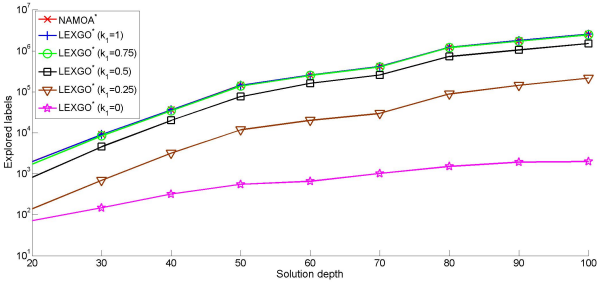


Figure: Explored labels by NAMOA* and LEXGO* in grids.

We can observe a reduction of expanded labels of several orders of magnitude for LEXGO* when k1 is equal to 0 or 0.25. The more restrictive the goals are the greater is that reduction. We can also observe that in the second class of experiments when the second level gets more restrictive.

Deviation-based pruning

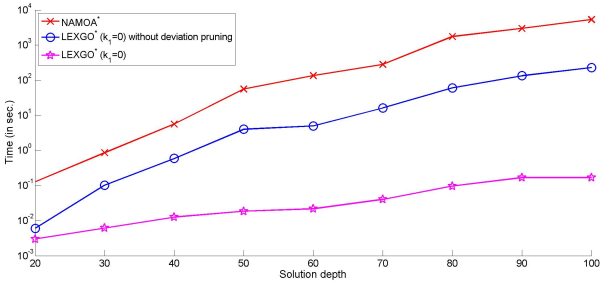


Figure: Impact of deviation pruning in $k_1 = 0$ experiments.

In this graphic we see the runtimes of NAMOA* and LEXGO* with goals in the ideal point with and without using the deviation-based pruning. For the most difficult problems, the runtimes without this rule vary from minutes to tenths of a second.

NAMOA$^*_{dr}$ vs NAMOA*

Table: Set sizes for Vermont State map.

Map	$(\frac{\sum T(G_{cl})}{\sum G_{cl}})\%$	$(\frac{\sum T(C^*)}{\sum C^*})\%$
VT_{cut}	3.43	2.45

The relative size of the truncated sets is also reduced to 2 to 3% of the size of the standard sets Gcl and COSTS.

NAMOA$^*_{dr}$ vs NAMOA*

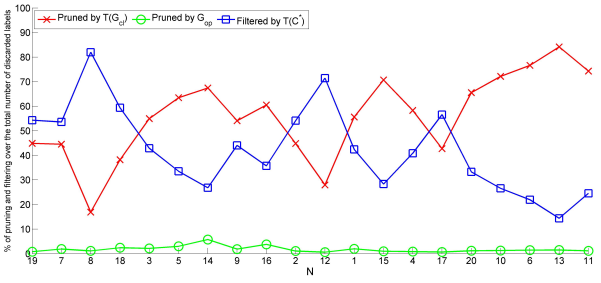


Figure: Discarded labels in Vermont problems.

The percentage of labels pruned by Gop is even smaller for road maps, so that, we can expect even bigger runtime improvements in this domain.

New York experiments

Now, we can even solve more problems!

Problem #	NAMOA$^*_{dr}$	NAMOA$^*_{lin}$
#11	30 min.	31 hours
#6	3 hours	25 days

As an example, we are going to show now the runtimes of NAMOA$^*_{dr}$ and NAMOA$^*_{lin}$ sorted by increasing difficulty. The easiest problem of New York was solved by both algorithms really fast. The second problem was solved by NAMOA$^*_{dr}$ twice as fast as NAMOA$^*_{lin}$. The third problem, number 16, was solved around 6 times faster. The fourth problem was solved by NAMOA$^*_{dr}$ in two minutes and half and more than an hour by the standard version of NAMOA*. The fifth problem by difficulty was solved by NAMOA$^*_{dr}$ in less than half an hour, and it tood 31 hours to be solved by NAMOA$^*_{lin}$, in other words, a speed up of more than sixty. Problem six, was solved in about three hours by NAMOA$^*_{dr}$ and the impressive amount of 25 days by NAMOA$^*_{lin}$. We can easily see where this is going if we wanted to keep solving problems with NAMOA$^*_{lin}$.

Outline

Outline

Conclusions

1. We have tackled the MSP with lexicographic preferences from two approaches: a priori and a posteriori.
2. On the a priori approach, we have contributed two new algorithms: LEXGO* and LEXGO$^*_{dr}$.
3. On the a posteriori approach: NAMOA$^*_{dr}$ based on NAMOA*.
4. All algorithmic approaches have been formally proved to be admissible.
5. The contributions of this thesis outperform the state of the art in an order of magnitude for medium size problems and two orders of magnitude for the hardest problems.
6. NAMOA$^*_{dr}$ LEXGO$^*_{dr}$ represent the new state of the art in MSP with lexicographic goals.

Outline

Future work

1. Investigate more formal proofs about LEXGO*, concerning the optimality in its class of algorithms and the expansion of labels when using more informed lower bounds.
2. Extend LEXGO* to be used with other GP models or other formulas to measure the deviation from goals.
3. Research the applicability of t-discarding to other multiobjective or multicriteria search algorithms with lower bounds.
4. Bring multicriteria search algorithms to real route planning applications.

Thank you for your attention !!!

Gracias por su atención !!!

New Techniques and Algorithms for Multiobjective and Lexicographic Goal-Based Shortest Path Problems

PhD Dissertation

Francisco J. Pulido Arrebola
Supervised by: Dr. Lawrence Mandow

Dept. Lenguajes y Ciencias de la Computación
E.T.S. Ingeniería Informática
Universidad de Málaga

July 7, 2015