OWA-based Search in State Space Graphs with Multiple Cost Functions

Lucie Galand, Olivier Spanjaard

OWA-based Search in State Space Graphs with Multiple Cost Functions

Lucie Galand
LIP6 - University of Paris VI
4 Place Jussieu
75252 Paris Cedex 05, France
lucie.galand@lip6.fr

Olivier Spanjaard
LIP6 - University of Paris VI
4 Place Jussieu
75252 Paris Cedex 05, France
olivier.spanjaard@lip6.fr

Abstract

This paper is devoted to the determination of well-balanced solutions in search problems involving multiple cost functions. After indicating various contexts in which the ordered weighted averaging operator (with decreasing weights) is natural to express the preferences between solutions, we propose a search algorithm to determine the OWA-optimal solution. More precisely, we show how to embed the search for a best solution into the search for the set of Pareto solutions. We provide a sophisticated heuristic evaluation function dedicated to OWA-optimization, and we prove its admissibility. Finally, the numerical performance of our method are presented and discussed.

Introduction

In the heuristic exploration of a state space, the search is usually totally ordered by a scalar and decomposable evaluation function, which makes it possible to explore only a small part of the graph to compute the best solution, as in A* (Hart, Nilsson, & Raphael 1968). However, many real world problems considered in AI involve multiple dimensions, as for instance agents, scenarios or criteria. In the frame of multiobjective search, practitioners have investigated vector-valued extensions of the standard paradigm (one dimension for each criterion). Facing such problems, a popular model to compare solutions consists in using a partial order called *dominance*. A solution is said to dominate another one if its cost vector is at least as "good" on every component, and strictly "better" on at least one component. In this frame, one characterizes interesting solutions as non-dominated solutions (i.e., solution for which there does not exist another solution dominating it), also called Pareto solutions. Since the 1990's there has been a continuous interest in designing algorithms able to cope with multiple objectives, and more precisely to enumerate the whole set of Pareto solutions (Stewart & White 1991; Dasgupta, Chakrabarti, & DeSarkar 1999; Mandow & Pérez-de-la Cruz 2005). These algorithms have been fruitfully applied in various fields such as mobile robot path navigation (Fujimura 1996) or planning (Refanidis & Vlahavas 2003). However, in many contexts, there is no need to determine the entire set of Pareto solutions, but only specific well-balanced solutions among the Pareto solutions. Focusing on well-balanced solutions makes it possible to discriminate between Pareto solutions and thus to reduce the computational effort by adequately pruning the search. For the sake of illustration, we give below three examples of natural problems that boil down to search for well-balanced solutions in state space graphs with multiple cost functions.

Example 1 (Fair allocation) *Consider a* task allocation problem*, where we want to assign n tasks $T_1, \ldots, T_n$ to m agents $a_1, \ldots, a_m$. The agents have different abilities which lead to different times for handling tasks. We denote t_{ij} the time spent by agent a_i to perform task T_j. The state space graph is defined as follows. Each node is characterized by a vector $(s, d_1, \ldots, d_j)$ ($j \in \{1 \ldots n\}$) representing a state in which the decisions on the first j tasks have been made. Component d_k takes value in $\{a_1, \ldots, a_m\}$, with $d_k = a_i$ if task T_k is performed by agent a_i. Starting node s corresponds to the initial state where no decision has been made. Each node of type $(s, d_1, \ldots, d_j)$ has m successors, $(s, d_1, \ldots, d_j, d_{j+1})$ with $d_{j+1} \in \{a_1, \ldots, a_m\}$. A vector is assigned to each arc from node $(s, d_1, \ldots, d_j)$ to node $(s, d_1, \ldots, d_{j+1})$, the i^{th} component of which is $t_{i(j+1)}$ if $d_{j+1} = a_i$, all others being 0. Goals are nodes of type $(s, d_1, \ldots, d_n)$. For each solution-path from s to a goal node, one can consider a vector, the i^{th} component of which represents the global working time of agent a_i (i.e., the sum of the times of the tasks performed by a_i). This is precisely the sum of the vectors along the arcs of the path. We consider here an instance with three tasks and two agents such that $t_{11} = 16$, $t_{12} = 4$, $t_{13} = 14$, $t_{21} = 13$, $t_{22} = 6$, $t_{23} = 11$. There are 8 solution-paths in this problem, named $S_1, \ldots, S_8$. The vectors associated with them are represented on Figure 1. For example, the one associated with path (s, a_1, a_1, a_2) is $(16, 0) + (4, 0) + (0, 11) = (20, 11)$. The aim is to find a* fair and efficient allocation *according to this multidimensional representation.*

Example 2 (Robust optimization) *Consider a* route-finding problem *in the network pictured on Figure 2, where the initial node is 1 and the goal nodes are 6 and 7. In* robust optimization *(Kouvelis & Yu 1997), the costs of paths depend on different possible scenarios (states of the world), or different viewpoints (discordant sources of information). Assume that only two scenarios are relevant here concerning the traffic, yielding two different sets of*

solution	vector
S_1	$(0, 30)$
S_2	$(4, 24)$
S_3	$(14, 19)$
S_4	$(16, 17)$
S_5	$(18, 13)$
S_6	$(20, 11)$
S_7	$(30, 6)$
S_8	$(34, 0)$

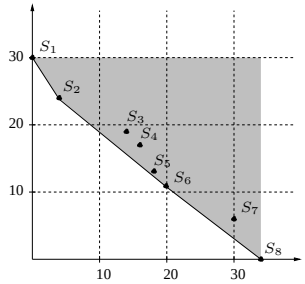

Figure 1: Representation in a multidimensional space.

costs on the network. Hence, to each solution-path S_j from 1 *to* 6 *or* 7 *is associated a vector (the sum of the vectors along its arcs), the i^{th} component of which represents the cost of S_j when the i^{th} scenario occurs. There are again* 8 *solution-paths, named $S_1, \ldots, S_8$. The vectors associated with them are the same as in the previous example (see Figure 1). The aim here is to find a* robust solution *according to this multidimensional representation, i.e. a solution that remains suitable whatever scenario finally occurs.*

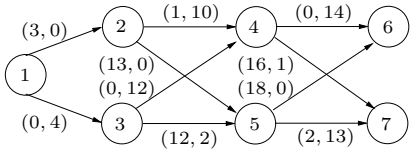

Figure 2: The state space graph.

Example 3 (Multicriteria decision making) *Consider a* robot navigation problem *with a set of states $\{e_1, \ldots, e_7\}$ and a set of actions $\{a_1, \ldots, a_6\}$, where the initial state is e_1 and the goal states are e_6 and e_7. Performing an action in a state yields to another state. Each action is evaluated on two cost criteria (for instance, electric consumption and time). The costs are given on Table 1. In the first column are indicated the states from which the action in the second column can be performed. In the third column are indicated the costs of actions, and in the last column is indicated the resulting state after performing an action. There are 8 possible sequences of actions, named $S_1, \ldots, S_8$, to reach states e_6 and e_7. The vectors associated with them are again the same as in Example 1 (see Figure 1). The aim here is to find a* best compromise solution *according to both criteria.*

state	action	cost	resulting state
e_1	a_1	$(4, 0)$	e_2
	a_2	$(0, 6)$	e_3
e_2, e_3	a_3	$(0, 11)$	e_4
	a_4	$(14, 0)$	e_5
e_4, e_5	a_5	$(0, 13)$	e_6
	a_6	$(16, 0)$	e_7

Table 1: A robot navigation problem.

In these three examples, computing the whole set of Pareto solutions provides no information. Indeed, we can see on Figure 1 that all solutions are Pareto solutions. To overcome this drawback, one can use a disutility function, compatible with the dominance order, to focus on one specific Pareto solution. However, this might raise another difficulties. For instance, using the average of the costs yields solution S_2 which is a very bad solution on the second component (representing an agent, a scenario or a criterion). Performing a weighted sum of the costs does not solve this problem either. Indeed, it can easily be shown that solutions S_3, S_4 and S_5, that seem promising, cannot be obtained by minimizing a weighted sum of costs since they do not belong to the boundary of the convex hull (grey area on Figure 1) of the points representing solutions in the multidimensional space. Finally, focusing only on the worst component (minimax criterion), although it yields to interesting solutions, is a quite conservative approach. For example, solution S_5 cannot be obtained by the minimax criterion, despite its promising costs, due to presence of solution S_4.

These observations show the limitations of standard decision criteria in the context of Examples 1, 2 and 3. Hence, we need to resort to another decision criterion for measuring the quality of a solution in these contexts. The ordered weighted averaging (OWA) operator seems to be a relevant criterion in this concern. Indeed, it avoids the pitfalls of standard decision criteria: unlike the weighted sum, it makes it possible to yield a solution the cost vector of which is not on the boundary of the convex hull; furthermore, this model is less conservative than the minimax one since it takes into account all the components of the vector. The aim of this paper is to design a state space search algorithm able to find a best solution according to this decision criterion.

The paper is organized as follows: in the next section, we introduce preliminary formal material and we present the OWA operator, that reveals an appropriate disutility function for characterizing a fair, robust or best compromise solution. Then, we propose a multiobjective search algorithm for the determination of an OWA-optimal solution. In this framework, we provide a heuristic evaluation function for pruning the search, and we prove its admissibility. Finally, numerical experiments are presented and discussed.

Problem Formulation

Notations and Definitions

We consider a state space graph $G = (N, A)$ where N is a finite set of nodes (possible states), and A is a set of arcs representing feasible transitions between nodes. Formally, we have $A = \{(n, n') : n \in N, n' \in S(n)\}$ where $S(n) \subseteq N$ is the set of all successors of node n (nodes that can be reached from n by a feasible elementary transition). We call *solution-path* a path from s to a goal node $\gamma \in \Gamma$. Throughout the paper, we assume that there exists at least one solution-path.

The three kinds of search problems presented above (multi-agent search, robust search and multi-criteria search) differ from the standard one by their vector-valued cost structure. Indeed, the state space graph is provided with a valuation function $v : A \to \mathbb{N}^m$ which assigns to each arc $a \in A$ a vector $v(a) = (v_1(a), \ldots, v_m(a))$, where $v_i(a)$ denotes the value according to the i^{th} component (i.e., agent, scenario or criterion). The cost-vector $v(P)$ of a path P is then defined as the componentwise sum of the vectors of its arcs. Hence, the comparison of paths reduces to the compar-

ison of their associated cost-vectors. Since we are looking here for well-balanced vectors, we propose to compare paths according to the ordered weighted averaging (OWA) operator (Yager 1988) of their associated cost-vectors.

The OWA Criterion

Definition 1 *Given a vector x, its ordered weighted average is $owa(x)$=$\sum_i w_i x_{(i)}$, where $\sum_i w_i$=1 and $x_{(1)} \geq .. \geq x_{(m)}$ are the components of x sorted by nonincreasing order.*

We consider here the subclass of OWA operators where the weights are ranked in decreasing order: $w_1 \geq \ldots \geq w_m$, which leads to give a greater importance to high-cost components. This preoccupation is natural when looking for well-balanced solutions. Note that criterion max also focuses on well-balanced solutions, but it is very conservative since it only considers the highest cost component. However, there exists a less conservative operator, called leximax, which refines the order induced by max. It consists in comparing two vectors according to their highest component, and then their second highest component in case of equality, and so on... The same order among vectors can be obtained from the OWA criterion by setting a big-stepped distribution of weights (i.e., $w_1 \gg \ldots \gg w_m$).

Coming back to solutions in Figure 1, the ordered weighted average enables to reach all interesting solutions (in particular, solution S_5). Indeed, by varying the weighting vector w, we obtain solution S_2 when $w_1 \in [0.5; 0.6]$ (and therefore $w_2 \in [0.4; 0.5]$), solution S_5 when $w_1 \in [0.6; 0.75]$ and solution S_4 when $w_1 \in [0.75; 1]$. More generally, the OWA criterion proved meaningful in the three contexts mentioned in the introduction:

• *fair allocation:* Ogryczak (2000) imports concepts from inequality measurement in social choice theory to show the interest of the OWA operator in measuring the equity of a cost distribution among agents. In particular, he shows that the OWA operator is consistent with the Pigou-Dalton transfer principle, which says that, if a distribution x can be obtained from a distribution y by a transfer of cost from y_i to y_j ($y_i > y_j$), then distribution x should be preferred to y.

• *robust optimization:* Perny & Spanjaard (2003) provide an axiomatic characterization in a way similar to von Neumann and Morgenstern one. The authors exhibit axioms that are very natural for modelling robustness. They show that these axioms characterize an OWA criterion with strictly positive and strictly decreasing weights.

• *multicriteria decision making:* Yager (1988) introduced the OWA operator for aggregating multiple criteria to form an overall decision function. He emphasizes that this operator makes it possible to model various forms of compromise between min (one compares vectors with respect to their lowest component) and max (one compares vectors with respect to their highest component).

Problem and complexity

We are now able to formulate the OWA search problem:
Given: a vector valued state space graph $G = (N, A)$, with a source node $s \in N$ and a set $\Gamma \subseteq N$ of goal nodes.
Find: a solution path with an OWA-optimal vector in the set of all vectors of solution-paths in G.

This problem is NP-hard. Indeed, choosing $w_1 = 1$ and $w_i = 0\ \forall i = 2, \ldots, m$, we get $owa(x) = x_{(m)} = \max_i x_i$. Hence OWA minimization in a vector graph reduces to the min-max shortest path problem, proved NP-hard by Murthy & Her (1992). However, note that the problem can be solved in polynomial time of the size of the state space for some classes of instances:

• when $w_1 = w_2 = \ldots = w_m$, the optimal path can be found by applying the heuristic search algorithm A* on the state space graph where each arc a is valued by $\sum_i v_i(a)$;

• when there exists a permutation π of $\{1, \ldots, m\}$ such that $v_{\pi(1)}(a) \geq \ldots \geq v_{\pi(m)}(a)$ for every arc a, the optimal path can be found by applying A* on the state space graph where each arc a is valued by $\sum_i w_i v_{\pi(i)}(a)$.

Unfortunately, this kind of instances are quite uncommon. That is why we propose below a new heuristic search algorithm able to solve any OWA search problem (provided the weights are decreasing).

Search for an OWA-optimal Solution

We now show that Bellman's principle does not hold when looking for an OWA optimal solution-path in a state space graph. Consider Example 2 and assume that the weights of the OWA operator are w_1= 0.8 and w_2=0.2. The optimal solution-path is solution S_4= $\langle 1, 3, 4, 7\rangle$ with cost $(16, 17)$ and $owa(16, 17)$=16.8. However, subpath P=$\langle 1, 3, 4\rangle$ from node 1 to node 4 is not optimal since path P'=$\langle 1, 2, 4\rangle$ is better. Indeed we have $v(P)$=$(0, 16)$ with $owa(0, 16)$=12.8, and $v(P')$= $(4, 10)$ with $owa(4, 10)$=8.8. This is a violation of Bellman's principle (any subpath of an optimal path is optimal). It invalidates a direct dynamic programming approach: optimal solution S_4 would be lost during the search if P is pruned at node 4 due to P'.

To overcome the difficulty, one can use the property that any OWA-optimal solution is a Pareto solution. We recall that a vector x is said to *Pareto-dominate* a vector y if $\forall i$=$1, \ldots, m$, $x_i \leq y_i$ and $\exists i$, $x_i < y_i$. A vector is *Pareto optimal* in a set X if no vector y in X Pareto-dominates it. Hence one can resort to multiobjective heuristic search algorithms (the validity of which follows from the compatibility of Pareto optimality with Bellman's principle), like MOA* (Stewart & White 1991) or the new approach to multiobjective A* designed by Mandow & Pérez-de-la Cruz (2005), named NAMOA*. The search can be refined by focusing directly on OWA-optimal solutions during the search. This is related to works of Mandow & Pérez-de-la Cruz (2003) and Galand & Perny (2006), that also used a disutility function to prune large parts of the state space during the search.

We now detail our refinement of NAMOA* called OWA*, for the direct determination of an OWA-optimal solution. As in NAMOA*, OWA* expands vector-valued labels (attached to subpaths) rather than nodes. Note that, unlike the scalar case, there possibly exists several Pareto paths with distinct cost-vectors to reach a given node; hence several labels can be associated to a same node n. At each step of the search, the set of generated labels is divided into two disjoint sets: a set OPEN of not yet expanded labels and a set CLOSED of

already expanded labels. Whenever the label selected for expansion is attached to a solution-path, its OWA value is compared with value $best$ of the best solution-path found so far, and $best$ is updated if necessary. Initially, OPEN contains only the label attached to the empty subpath on node s, CLOSED is empty and $best$ is set to $+\infty$. We describe below the essential features of the OWA* algorithm.

Output: it determines an OWA-optimal solution-path. If several paths have the same OWA value, only one of these paths is stored using standard bookkeeping techniques.
Heuristics: at any node n, we assume we have a heuristic value $h_i(n)$ for each cost function v_i, that underestimates the value of the best path from n to a goal node w.r.t. v_i. Furthermore, we assume we also have a heuristic value $h_S(n)$ that underestimates the value of the best path from n to a goal node w.r.t. $\sum_{i=1}^{m} v_i$. These heuristic values are computed by resorting to a context-dependent scalar heuristic function for the search problem under consideration. It is important to note that function $h_S \neq \sum_{i=1}^{m} h_i$. Indeed, contrarily to h_S, each function h_i underestimates the best value w.r.t. v_i independently from the values on other objectives.
Priority: to direct the search we use a scalar label evaluation $ev(\ell)$ underestimating the OWA value of the best solution-path we can be obtain from ℓ. Hence, OWA* expands in priority the minimal label $\ell \in$ OPEN according to ev. The computation of $ev(\ell)$ is described in the next section.
Pruning: as shown above, the pruning of labels cannot be done directly with the OWA operator. The following pruning rules are used:
RULE 1: at node n, a label $\ell \in$ OPEN is pruned if there exists another label ℓ' at the same node n such that $g(\ell)$ is Pareto-dominated by $g(\ell')$, where $g(\ell)$ denotes the cost-vector $v(P)$ of the subpath P associated with ℓ. This rule is the same as in NAMOA* and is justified by the fact that labels pruned by rule 1 necessarily lead to solution-paths with cost-vectors outside the Pareto set, and there always exists an OWA-optimal vector in this set. Indeed, if a vector x Pareto dominates a vector y, then $owa(x) \leq owa(y)$.
RULE 2: a label $\ell \in$ OPEN is pruned if $ev(\ell) \leq best$, where $ev(\ell)$ is a function underestimating the OWA value of the best solution-path that can be obtained from ℓ. This rule allows an early elimination of uninteresting labels while keeping admissibility of the algorithm.
Termination: the process is kept running until set OPEN becomes empty, i.e. there is no remaining subpath able to reach a solution-path improving $best$. By construction, OWA* explores a subgraph of the one explored by NAMOA* and its termination derives from the one of NAMOA*.

A Heuristic Evaluation Function for OWA

Partitioning the representation space

Given a label ℓ on a node n, we now explain how to compute the value $ev(\ell)$. Let $g_i(\ell)$ denote the i^{th} component of $g(\ell)$. We use value $f_S(\ell) = \sum_i g_i(\ell) + h_S(n)$ and vector $f(\ell)$ defined by $f_i(\ell)$=$g_i(\ell) + h_i(n)$ for i=$1, \ldots, m$. Note that $f_i(\ell)$ underestimates the value of the optimal solution-path that can be obtained from the subpath attached to ℓ, when focusing on the i^{th} component only. The evaluation function we propose consists in solving the following program:

$$(P_\ell) \begin{cases} \min owa(x) & & & \\ \quad x_i & \geq & f_i(\ell) & \forall i = 1..m \\ \sum_{i=1}^{m} x_i & \geq & f_S(\ell) & \\ \quad x & \in & \mathbb{R}^m & \end{cases}$$

When $f_S(\ell) \leq \sum_{i=1}^{m} f_i(\ell)$ (which is very unlikely), the optimal value is obviously $owa(f(\ell))$. In other cases, the solution of (P_ℓ) requires to take into account the way the components of x are ordered. Hence, this problem can be divided into several subproblems, each one focusing on a subspace of $\mathbb{R}^m$ where all vectors are comonotonic, i.e. for any pair x, y of vectors there exists a permutation π of $(1, \ldots, m)$ such that $x_{\pi(1)} \geq \ldots \geq x_{\pi(m)}$ and $y_{\pi(1)} \geq \cdots \geq y_{\pi(m)}$. Within a particular subspace, the OWA operator reduces to a usual weighted sum of the components. The solution of (P_ℓ) reduces therefore to solving each linear program defined by a particular permutation π of $(1, \ldots, m)$:

$$(P_{\ell,\pi}) \begin{cases} \min \sum_{i=1}^{m} w_i x_{\pi(i)} & & & & \\ \quad x_{\pi(i)} & \geq & x_{\pi(i+1)} & \forall i = 1..m-1 & (1.1) \\ \quad x_i & \geq & f_i(\ell) & \forall i = 1..m & (1.2) \\ \sum_{i=1}^{m} x_i & \geq & f_S(\ell) & & (1.3) \\ \quad x & \in & \mathbb{R}^m & & \end{cases}$$

Value $ev(\ell)$ is then defined by $ev(\ell)$=$\min_{\pi \in \Pi} owa(x^*_{\ell,\pi})$ where $x^*_{\ell,\pi}$ denotes the optimal solution to linear program $(P_{\ell,\pi})$ and Π the set of all possible permutations. Note that for m components, there are $|\Pi|$=$m!$ linear programs to solve. However, in practice, it is not necessary to solve the $m!$ linear programs. Indeed, there exists an easily computable permutation π^* for which $ev(\ell)$=$owa(x^*_{\ell,\pi^*})$:

Proposition 1 *Let π^* denote the permutation such that $f_{\pi^*(1)}(\ell) \geq f_{\pi^*(2)}(\ell) \geq \ldots \geq f_{\pi^*(m)}(\ell)$. For all feasible solution x to $(P_{\ell,\pi})$, there exists a feasible solution y to (P_{ℓ,π^*}) such that $owa(y) = owa(x)$.*

Proof. The idea is to determine a feasible solution y to (P_{ℓ,π^*}) such that $y_{(i)}$=$x_{(i)}$ $\forall i$. Indeed, it implies $owa(x)$=$owa(y)$ and the conclusion is then straightforward. In this respect, we construct a sequence $(x^j)_{j=1,\ldots,k}$ of solutions and a sequence $(\pi^j)_{j=1,\ldots,k}$ of permutations s.t. x^j is feasible for P_{ℓ,π^j} (for j=$1, \ldots, k$), with x^1=x, π^1=π, π^k=π^* and $x^1_{(i)}$=$x^2_{(i)}$= $\ldots$ =$x^k_{(i)}$ $\forall i$. Assume that $\exists\ i_0, i_1 \in \{1, \ldots, m\}$ s.t. $i_0 < i_1$ and $f_{\pi^1(i_0)}(\ell) < f_{\pi^1(i_1)}(\ell)$. Let permutation π^2 be defined by $\pi^2(i_0)$=$\pi^1(i_1)$, $\pi^2(i_1)$=$\pi^1(i_0)$, and $\pi^2(i)$=$\pi^1(i)$ $\forall i \neq i_0, i_1$. Let solution x^2 be defined by $x^2_{\pi^2(i)}$=$x^1_{\pi^1(i)}$ for i=$1, \ldots, m$. We now show that x^2 is a feasible solution to (P_{ℓ,π^2}). Note that $x^1_{\pi^1(i_0)} \geq f_{\pi^1(i_0)}(\ell)$, $x^1_{\pi^1(i_1)} \geq f_{\pi^1(i_1)}(\ell)$, $x^1_{\pi^1(i_0)} \geq x^1_{\pi^1(i_1)}$ and $f_{\pi^1(i_1)}(\ell) > f_{\pi^1(i_0)}(\ell)$. Hence, constraints (1.2) are satisfied:
• $x^2_{\pi^2(i_0)}$=$x^1_{\pi^1(i_0)} \geq x^1_{\pi^1(i_1)} \geq f_{\pi^1(i_1)}(\ell)$=$f_{\pi^2(i_0)}(\ell)$,
• $x^2_{\pi^2(i_1)}$=$x^1_{\pi^1(i_1)} \geq f_{\pi^1(i_1)}(\ell) > f_{\pi^1(i_0)}(\ell)$=$f_{\pi^2(i_1)}(\ell)$,
• $x^2_{\pi^2(i)}$=$x^1_{\pi^1(i)} \geq f_{\pi^1(i)}(\ell)$= $f_{\pi^2(i)}(\ell)$ for $i \neq i_0, i_1$.
Constraints (1.1) are also satisfied since $[x^1_{\pi^1(i)} \geq x^1_{\pi^1(i+1)}$ $\forall i] \Rightarrow [x^2_{\pi^2(i)} \geq x^2_{\pi^2(i+1)}$ $\forall i]$. Indeed, $x^1_{\pi^1(i)}$=$x^2_{\pi^2(i)}$ $\forall i$. These equalities imply also that constraint (1.3) is satisfied

and that $x_{(i)} = y_{(i)}\ \forall i$. Solution x^2 is therefore feasible for (P_{ℓ,π^2}) with $x_{(i)}=y_{(i)}\ \forall i$. Since any permutation is the product of elementary permutations, one can always construct in this way a sequence of permutations that leads to π^* (and the corresponding feasible solutions). By setting $y=x^k$, one obtains the desired feasible solution to (P_{ℓ,π^*}). □

An immediate consequence of this result is that $ev(\ell) = owa(x^*_{\ell,\pi^*})$. Thus, the computation of $ev(\ell)$ reduces to solving linear program (P_{ℓ,π^*}). Furthermore, we now show that the solution of this program can be performed in linear time of m without resorting to a linear programming solver.

Resolution of (P_{ℓ,π^*})

For the convenience of the reader, we first explain the principle of the solution procedure thanks to an example with three cost functions. Consider a label ℓ for which $f_S(\ell) = 21$, $f(\ell) = (5, 10, 3)$. Since $f_2(\ell) \geq f_1(\ell) \geq f_3(\ell)$, permutation π^* is defined by $\pi^*(1) = 2$, $\pi^*(2) = 1$ and $\pi^*(3) = 3$. By Proposition 1, value $ev(\ell)$ is obtained by solving:

$$\left\{\begin{array}{rcll} \min & w_1x_2 + w_2x_1 + w_3x_3 & & \\ & x_2 \geq x_1 \geq x_3 & & (2.1)\\ & x_1 \geq 5 & & (2.2)\\ & x_2 \geq 10 & & (2.3)\\ & x_3 \geq 3 & & (2.4)\\ & x_1 + x_2 + x_3 \geq 21 & & (2.5)\\ & x \in \mathbb{R}^3 & & \end{array}\right.$$

The principle of the procedure is the following:
• we set $x=f(\ell)=(5, 10, 3)$: all constraints except (2.5) are thus satisfied. However, constraint (2.5) can be satisfied only if amount $21 - (5 + 10 + 3) = 3$ is added on one or several components of x.
• The component of minimal weight (i.e. x_3) is then increased as much as possible without violating constraints (2.1): we therefore set $x_3 = \min\{x_1; x_3 + 3\}$. We get $x_3 = 5$, which is not enough to satisfy constraint (2.5). This constraint can now be satisfied only if amount $21-(5+10+5) = 1$ is added on one or several components.
• The two components of minimal weights (i.e. x_3 and x_1) are then increased simultaneously as much as possible without violating constraints (2.1): we therefore set $x_3 = x_1 = \min\{x_2; x_1 + \frac{1}{2}\}$ (one distributes amount 1 among both components). We get here $x_1 = x_3 = 5.5$. Constraint (2.5) is now satisfied.
• $x = (5.5, 10, 5.5)$ is a feasible solution to the program: it is the optimum.

This computation can be simplified by directly determining components i for which the value in the optimal solution to (P_{ℓ,π^*}) is distinct from $f_i(\ell)$, and the common value of these components in the optimal solution. We now describe Algorithm 1 which precisely performs these operations. To satisfy constraint (1.3), it is necessary to distribute surplus $\Delta = f_S(\ell) - \sum_{i=1}^m f_i(\ell) > 0$ among the lowest components of the solution under construction.The number of these components is known by determining the lowest component that need not be modified. Its index is $k = \max\{j = 0..m\text{-}1 : \sum_{i=j}^{m-1}(m-i)(f_{\pi^*(i)}(\ell) - f_{\pi^*(i+1)}(\ell)) \geq \Delta\}$, with $f_{\pi^*(0)}(\ell) = +\infty$. Value $f_{\pi^*(i)}(\ell)$ is therefore assigned to component $\pi^*(i)$ for $i = 1, \ldots, k$. Value $r = (\sum_{i=k+1}^m f_{\pi^*(i)}(\ell) + \Delta)/(m-k)$ is then assigned to each of the $(m-k)$ lowest components (i.e. components $\pi^*(i)$ for $i = k+1, \ldots, m$), so as to satisfy constraint (1.3) for a minimum cost.

Algorithm 1: Solution of (P_{ℓ,π^*})

$\Delta \leftarrow f_S(\ell) - \sum_{i=1}^m f_i(\ell)$
$s \leftarrow 0$; $a \leftarrow 0$; $k \leftarrow m$
// a: max amount that can be added to components k+1,. . . , m
while $a < \Delta$ **do**
  $k \leftarrow k-1$; $s \leftarrow s + f_{\pi^*(k+1)}(\ell)$
  if $k = 0$ **then** $a \leftarrow +\infty$
  else $a \leftarrow a + (f_{\pi^*(k)}(\ell) - f_{\pi^*(k+1)}(\ell))(m-k)$
end
$r \leftarrow (s + \Delta)/(m-k)$
for $j = 1$ to k **do** $x^*_{\ell,\pi^*(j)} \leftarrow f_{\pi^*(j)}(\ell)$
for $j = (k+1)$ to m **do** $x^*_{\ell,\pi^*(j)} \leftarrow r$
Output: $owa(x^*_{\ell,\pi^*})$

Proposition 2 establishes the validity of our algorithm:

Proposition 2 *Let $k = max\ \{j = 0..m\text{-}1$: $\sum_{i=j}^{m-1}\ (m-i)\ (f_{\pi^*(i)}(\ell)$ - $f_{\pi^*(i+1)}(\ell)) \geq \Delta\}$ and $r = (\sum_{i=k+1}^m f_{\pi^*(i)}(\ell)\ + \Delta)\ /\ (m-k)$. An optimal solution to (P_{ℓ,π^*}) is $x = (f_{\pi^*(1)}(\ell), \ldots, f_{\pi^*(k)}(\ell)\ , r, \ldots, r)$.*

Proof. W.l.o.g., we assume that $f_1(\ell) \geq .. \geq f_m(\ell)$ (i.e. $\pi^* = id$) for clarity in the following. To simplify the proof presentation, we assume that $f_S(\ell) > \sum_i f_i(\ell)$ and $w_i > w_{i+1} > 0\ \forall i < m$. We first show that any optimal solution saturates constraint (1.3). Since $f_S(\ell) > \sum_i f_i(\ell)$, feasibility implies there exists at least one constraint (1.2) that is not saturated. Consequently if constraint (1.3) is not saturated, it is possible to decrease the value of at least one component while keeping feasibility, and hence to find a better solution. We now show that x is an optimal solution to (P_{ℓ,π^*}). Consider a feasible solution $y \neq x$. We claim that y cannot be optimal. Indeed, there are two cases:
Case 1. $\exists j_0 \leq k$ s.t. $y_{j_0} > f_{j_0}(\ell)$: two subcases:
Case 1.1. $\forall i > j_0, y_i \geq y_{j_0}$: since $y_i \leq y_{j_0}\ \forall i > j_0$ by constraints (1.1), we have therefore $y_i = y_{j_0}\ \forall i > j_0$. Hence $\sum_{i=1}^m y_i = \sum_{i=1}^{j_0} y_i + (m-j_0)y_{j_0}$. However, $y_{j_0} > f_{j_0}(\ell)$ by assumption, $f_{j_0}(\ell) = x_{j_0}$ by definition of x, and $\forall i \geq j_0$ $x_{j_0} \geq x_i$ by constraint (1.1). Therefore $\forall i \geq j_0\ y_{j_0} > x_i$. It implies $\sum_{i=1}^m y_i > \sum_{i=1}^{j_0} f_i(\ell) + \sum_{i=j_0+1}^m x_i = \sum_{i=1}^m x_i$ since $x_i = f_i(\ell)\ \forall i \leq j_0$. Consequently, $\sum_{i=1}^m y_i > \sum_{i=1}^m x_i = f_S(\ell)$. Constraint (1.3) is not saturated and y cannot then be optimal.
Case 1.2. $\exists t > j_0$ s.t. $y_t < y_{j_0}$. Let $t_0 = \min\{i : i > j_0$ and $y_i > y_{i+1}\}$. By definition of t_0 we have $y_{t_0} = y_{j_0}$, which implies $y_{t_0} > f_{j_0}(\ell) \geq f_{t_0}(\ell)$. Let $\varepsilon_1 = y_{t_0} - y_{t_0+1} > 0$, $\varepsilon_2 = y_{t_0} - f_{t_0}(\ell) > 0$ and $\varepsilon = \min\{\varepsilon_1, \varepsilon_2\} > 0$. Consider a solution z to (P_{ℓ,π^*}) defined by $z_{t_0} = y_{t_0} - \varepsilon/2$, $z_{t_0+1} = y_{t_0+1} + \varepsilon/2$ and $z_j = y_j$ otherwise, z is then feasible. Since $w_{t_0} > w_{t_0+1}$, $owa(y) > owa(z)$. Then y is not optimal.
Case 2. $\forall j \leq k$, $y_j = f_j(\ell)$, and $\exists j_0 > k$ s.t. $y_{j_0} \neq r$:

Case 2.1. Assume that $y_{j_0} > r$: if there exists $t > j_0$ s.t. $y_t < y_{j_0}$ then y is not optimal (case 1.2). If $y_{j_0} = y_i$ for all $i > j_0$, then $y_i \geq y_{j_0} > r \; \forall i \in \{k+1,\ldots,j_0\}$ and $y_i = y_{j_0} > r \; \forall i \in \{j_0+1, \ldots, m\}$. Then $\sum_{i=k+1}^{m} y_i > (m-k)r$. Since $\sum_{i=1}^{k} y_i = \sum_{i=1}^{k} x_i$, we have $\sum_{i=1}^{m} y_i > \sum_{i=1}^{m} x_i$ and constraint (1.3) is not saturated. Then y is not optimal.
Case 2.2. Assume that $y_{j_0} < r$: we have then $y_i \leq y_{j_0} < r \; \forall i \in \{j_0, \ldots, m\}$, and therefore $\sum_{i=j_0}^{m} y_i < (m-j_0+1)r$. However $\sum_{i=k+1}^{m} y_i = (m-k)r$ when constraint (1.3) is saturated. Then $\sum_{i=k+1}^{j_0-1} y_i > (m-k)r - (m-j_0+1)r = (j_0-k-1)r$. It implies there exists i_0 in $\{k+1, \ldots, j_0-1\}$ s.t. $y_{i_0} > r$, and thus y cannot be optimal (see case 2.1). □

Numerical Tests

Algorithms were implemented in C++. The computational experiments were carried out with a Pentium IV 3.6Ghz PC. Table 2 summarizes the average performances of OWA* when the evaluation function presented in the previous section is used (sharp approach SA). We compare it with the results obtained with a more naïve approach (NA), where the evaluation function consists in computing the OWA value of vector $f(\ell)$. The same priority rule is used in both approaches. The tests have been performed for different classes of graphs $G_{i,j}$, characterized by their number i of thousands of nodes and j of cost functions. The number of nodes range from 1,000 (with 190,000 arcs) to 3,000 (with 2,000,000 arcs). Cost vectors are integers randomly drawn within [0,100] for each arc. For each class and each approach, we give three average time performances over 50 different instances, depending on weight vectors that yield a gradation of OWA operators from close to max (weights close to $(1, 0, \ldots, 0)$) to close to average (weights close to $(\frac{1}{m}, \ldots, \frac{1}{m})$). For every class, we give the average percentage of nodes generated by NA that have not been generated by SA ($\%_n$). Admissible heuristic functions (i.e., underestimating the values of the cheapest paths) were generated by setting $h_i(n) = \alpha h_i^*(n)$ and $h_S(n) = \alpha h_S^*(n)$, where α denotes a random value within [0.8,1) for each node and $h_i^*(n)$ (resp. $h_S^*(n)$) the perfectly informed heuristic for component i (resp. for the sum of components).

	close to max			between max & avg			close to avg		
	NA	SA	$\%_n$	NA	SA	$\%_n$	NA	SA	$\%_n$
$G_{1,3}$	0.2	0.1	67	0.2	0.1	74	0.2	0.1	77
$G_{2,3}$	0.7	0.3	67	0.8	0.3	74	0.8	0.2	74
$G_{3,3}$	1.5	0.7	61	1.9	0.6	76	2.1	0.6	77
$G_{1,5}$	0.3	0.1	71	0.3	0.1	82	0.5	0.1	87
$G_{2,5}$	1.2	0.4	73	1.4	0.3	83	1.7	0.3	87
$G_{3,5}$	2.7	0.9	70	2.9	0.7	78	3.4	0.6	85
$G_{1,10}$	1.4	0.5	64	2.0	0.2	92	2.1	0.2	92
$G_{2,10}$	6.3	2.2	66	9.0	0.7	92	11.1	0.7	94
$G_{3,10}$	15.7	5.0	69	21.4	1.7	92	23.0	1.4	94

Table 2: OWA-based search (time in seconds).

These results show that SA generates an optimal solution significantly faster than NA. The saving in the number of generated nodes is all the more so significant as the size of the search space increases. Moreover, SA is much more robust to weights changes than NA, thanks to the use of $f_S(\ell)$.

Conclusion

We have presented an efficient way to seek for well-balanced solutions in search problems with multiple cost functions. We have justified the use of the OWA operator to compare solutions in the representation space. We have then provided a new multiobjective search algorithm, named OWA*, to efficiently determine an OWA-optimal solution-path in a state space graph. The efficiency of OWA* strongly relies on a procedure that estimates the value of an OWA-optimal solution by combining two types of scalar heuristic informations: a value for each cost function, and a value taking into account tradeoffs between components. In the future, it should be worth studying decision criteria able to express interactions between components by weighting not only individual components, but also groups of components, such as *Choquet integral* (of which the OWA operators are particular instances) or *Sugeno integral* (e.g., Grabisch 1996).

Acknowledgements

This work has been supported by the ANR project PHAC which is gratefully acknowledged. We would like to thank Patrice Perny and Francis Sourd for their useful comments.

References

Dasgupta, P.; Chakrabarti, P.; and DeSarkar, S. 1999. *Multiobjective Heuristic Search*. Vieweg&Son/Morg. Kauf.

Fujimura, K. 1996. Path planning with multiple objectives. *IEEE Robotics and Automation Magazine* 3(1):33–38.

Galand, L., and Perny, P. 2006. Search for compromise solutions in multiobjective state space graphs. In *17th European Conference on Artificial Intelligence*, 93–97.

Grabisch, M. 1996. The application of fuzzy integrals in multicriteria decision making. *EJOR* 89:445–456.

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

Kouvelis, P., and Yu, G. 1997. *Robust discrete optimization and its applications*. Kluwer Academic Publisher.

Mandow, L., and Pérez-de-la Cruz, J.-L. 2003. Multicriteria heuristic search. *EJOR* 150(2):253–280.

Mandow, L., and Pérez-de-la Cruz, J.-L. 2005. A new approach to multiobjective A* search. In *IJCAI*, 218–223.

Murthy, I., and Her, S. 1992. Solving min-max shortest-path problems on a network. *Nav. Res. Log.* 39:669–683.

Ogryczak, W. 2000. Inequality measures and equitable approaches to location problems. *EJOR* 122(2):374–391.

Perny, P., and Spanjaard, O. 2003. An axiomatic approach to robustness in search problems with multiple scenarios. In *19th UAI*, 469–476.

Refanidis, I., and Vlahavas, I. 2003. Multiobjective heuristic state-space planning. *Artif. Intell.* 145(1-2):1–32.

Stewart, B., and White, C. 1991. Multiobjective A*. *JACM* 38(4):775–814.

Yager, R. 1988. On ordered weighted averaging aggregation operators in multi-criteria decision making. *IEEE Trans. on Sys., Man and Cyb.* 18:183–190.