Pareto Fronts for MultiZenoTravel Benchmarks

A. Quemy[1], M. Schoenauer[1], V. Vidal[2], J. Dréo[3], and P. Savéant[3]

[1] TAO Project, INRIA Saclay & LRI Paris-Sud University, Orsay, France
{alexandre.quemy, marc.schoenaue}@inria.fr,
[2] ONERA-DCSD, Toulouse, France
Vincent.Vidal@onera.fr
[3] THALES Research & Technology, Palaiseau, France
{johann.dreo, pierre.saveant@thalesgroup.com}

Abstract. Contrary to single objective planning, multi-objective planning suffers from a lack of benchmarks with a known Pareto Front. As a result, testing and comparing algorithms is almost impossible. A method to generate various size tunable benchmarks for multi-objective planning with a known Pareto Front is proposed in order to provide a wide range of Pareto front shapes and different magnitudes of difficulty. The performances of the implementation are shown and some large instances with singular Pareto front shapes are proposed. A first attempt to solve some of them using multi-objective DaE$_{\text{YAHSP}}$ solver is provided.

1 Introduction

Many benchmark suites exist for continuous multi-objective optimization (the famous ZDT, IHR, etc, proposed two decades ago), for which the exact Pareto front can be analytically computed, and with know difficulties (e.g. dimensionality, shape of the Pareto fronts, existence of local Pareto-optima, ...). For combinatorial optimization, the situation is not yet so clear, and whereas there exist known benchmark problems of all sizes, their Pareto fronts are generally not precisely known except the simplest ones (see e.g., MOCOLIB[4], offering several instances of several well-known combinatorial benchmark problems).

The context of the proposed benchmark suite is that of AI planning: a planning domain is defined by a set of predicates, that define the state of the system, a set of possible actions that can be triggered in states where their pre-conditions are satisfied, resulting in a new state. A planning problem instance is defined on a given planning domain by a list of objects, used to instantiate the predicates to define the state, an initial and a target state. The goal is to some up with a *feasible plan*, i.e., a set of actions that, when applied in turn to the initial state, lead the system to the goal state.

A simple planning problem in the domain of logistics, inspired by the well-known ZenoTravel problem of IPC series (see Figure 1) involves cities, passengers, and planes. Planes can fly from one city to another when a link exist, taking a given duration to do so (number on the link). Planes fly either empty, or carrying a unique passenger. An instance is defined by the number of cities and

[4] http://www.mcdmsociety.org/MCDMlib.html

the links between them, a number of passengers and a number of planes. The goal of the (single-objective) planning problem is to carry all passengers from city I to city G in the minimum *makespan* (total duration of all flights). Because all planes can of course fly in parallel, this benchmark pertains to temporal planning. Previous work [3] proposed a multi-objective version of these benchmarks called MultiZenoTravel, by adding a *cost* for landing in some cities: the second objective is to minimize the *total cost* of the plan. This work demonstrated the possibility for such problems to provide Pareto Front of various shapes and difficulties, but only provided the exact Pareto front for small instances.

This paper analyzes the MultiZenoTravel benchmarks, providing an algorithm to compute the true Pareto Front even for very large instances. Beyond providing a generic way to generate Pareto Fronts of diverse complexities for AI Planning, the proposed MultiZenoTravel benchmarks will allow testing different generic decomposition methods (weighted sum aggregation, Tchebycheff decomposition, Boundary Intersection approach – see e.g., [8]) on large, concave, non-uniform, ... benchmarks of Combinatorial Multi-objective Problems of for which the Pareto Front is exactly known.

The paper is organized as follows: Section 2 formally presents the MultiZenoTravel benchmark, proving some properties of their Pareto optimal plans. Building on these properties, Section 3 proposes the ZenoSolver algorithm to actually derive the true Pareto front for these instances. Sample experimental results demonstrate the diversity of Pareto fronts that can be obtained, and gives performance measurements of its complexity on large instances. The end of the paper presents experiments on some of the large MultiZenoTravel instances obtained by Divide-and-Evolve, the only Pareto-based evolutionary AI planner to-date [4], which is rapidly recalled in Section 4, before some comparison with its single-objective version using the weighted sum aggregation on problems with non-convex fronts are given and discussed in Section 4.

2 MultiZenoTravel problem

2.1 Instances

Let us introduce some notations related to the planning problems introduced in Section 1: a MultiZenoTravel instance (Figure 1) is defined by: n 'central' cities, with costs $c_1, \ldots, c_n$[5], plus the initial and goal cities c_I and c_G; t persons and p planes, initially in c_I. Each central city provides a flight to the initial and final cities, with same flight durations $d_1, \ldots, d_n$. In addition, central cities form a clique, with durations $(d_{ij})_{i,j}, \forall (i,j) \in [1,n]^2$. The goal is to carry all persons to c_G, minimizing the makespan and the total cost of the plan. Let us analyze some properties of the Pareto fronts of such problems, that will allow us to design an efficient algorithm to actually compute them in a reasonable time.

[5] by simplicity, c_i will denote both the city and the cost of landing in that city.

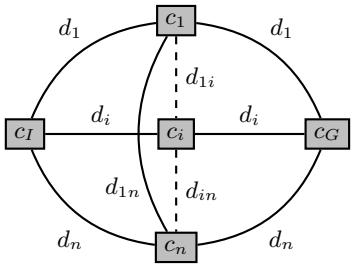

Fig. 1: A schematic view of a general MULTIZENOTRAVEL problem.

2.2 Pareto Optimal Plans

Hypothesis: Assume that, for all pair of cities (i, j), $d_i + d_j < d_{ij}$[6]. Then the following holds.
Proposition: Pareto-optimal plans are plans where exactly $2t - p$ central cities are reached by a flight.
Proof: Consider a plan where a person flies from c_i to c_j. Using the same plane, the same person could fly instead from c_i to c_G, and the plane would return empty to c_j. The plan could continue unchanged from thereon: because of the hypothesis on makespans, the needed resource would be in c_j on time. Moreover, the total cost is unchanged, and the total makespan is lower or equal to the original one: the new plan thus Pareto-dominates the original one.

Iterating the same reasoning for each person, and each empty plane, we conclude that there are no flights between central cities in Pareto-optimal plans. Thus bringing the t persons from c_I to c_G will amount to carry each person through one central city: t flights will be needed from c_I to one c_i, then t flights from c_i to c_G. Of course, $t - p$ flights back empty will be needed for some planes, possibly through some different central cities– hence the result. □

Counting and Pruning Possibly Pareto-optimal Plans. A Possibly Pareto-optimal Plan (PPP) is thus defined by 2 tuples, namely $e \in [1, n]^t$ for cities involved in eastbound flights, and $w \in [1, n]^{t-p}$ for westbound flights, and the optimal schedule for the p planes along the corresponding $4t - 2p$ arcs. However, such definition contains redundancies, that are removed by ordering the indices:

Definition: An *admissible PPP* is a pair of $E \times W$, where $E = \{e \in [1, n]^t; \forall i \in [1, t-1], d_{e_i} \geq d_{e_{i+1}}\}$ and $W = \{w \in [1, n]^{t-p}; \forall i \in [1, t-p-1], d_{w_i} \geq d_{w_{i+1}}\}$.

Number of admissible PPPs: Let K_k^m be the set of k-multicombinations (or multi-subset of size k) with elements in a set of of size m. The cardinality of K_k^m is $\Gamma_k^m = \binom{m+k-1}{k}$. As E is in bijection with K_t^n, and K_{t-p}^n with W, the number

[6] Even though this might look unrealistic in real-world logistic domain. However, we hypothesize that the result still holds with the weaker condition that for any cities c_i, c_j, c_k ($c_0 = I$ and $c_{n+1} = G$), $d_{ik} \leq d_{ij} + d_{jk}$ (triangular inequality).

of PPP is $\Gamma_t^n \Gamma_{t-p}^n$, i.e., $\binom{n+t-1}{t}\binom{n+(t-p)-1}{t-p}$.

Cost of a PPP: Given the PPP $(e, w) \in E \times W$, the cost of **any** plan using only the cities in e and w is $\text{Cost}(C) = \sum_{e_i \in e} c_{e_i} + \sum_{w_i \in w} c_{w_i}$.

Makespan of a PPP: The makespan of a PPP is thus that of the shortest schedule that uses exactly the $4t - 2p$ arcs in a feasible way. Trivial upper and lower bounds for the shortest makespan of PPP C are respectively $M_S(C)$, the makespan of the sequential plan (i.e., that of the plan for a single plane that would carry all persons one by one), and $M_L(C)$, the makespan of the perfect plan where none of the p planes would ever stay idle. These bounds will be useful to prune PPP.

$$M_S(C) = 2(\sum_{e_i \in e} d_{e_i} + \sum_{w_i \in w} d_{w_i}) \qquad\qquad M_L(C) = \frac{M_S(C)}{p}$$

Greedy (Pareto) domination: Given two PPP C and C', C *greedily dominates* C' if $M_S(C) \leq M_L(C')$ and $\text{Cost}(C) \leq \text{Cost}(C')$.

2.3 Computing the Shortest Makespan

Clearly, all possible plane moves during the execution of a PPP can be categorized into the following 3 patterns:
P1: plane leaves I (with passenger), flies eastward to city c_i, and goes on to G;
P2: plane leaves G (empty), flies westward to city c_i, and goes on to G;
P3: two planes are involved here; first plane leaves I (with passenger), flies to city c_i, and goes back empty to I; second plane leaves G empty, flies to c_i, and flies back with the passenger to G. Note that there can be some delay between the drop-off of the person at the central city, and the arrival of the second plane.

Let α_E, α_W, and β be the numbers of patterns P1, P2, and P3 respectively. Clearly, β entirely determines α_E and α_W, as $\alpha_W = t - p - \beta$ and $\alpha_E = t - \beta$.

Notice that, given β, there are multiple choices for the cities involved in the different P3. Each choice (list of cities) will be called a β_{set}, and the set of all possible β_{set} is called the β-PowerSet.

The optimal makespan of a given admissible PPP C for a given β is the lowest makespan obtained among all $\beta_{set} \in \beta$-PowerSet. For each β_{set}, the idea is to first solve the subproblem obtained by removing all P3s, and then to account for them in a second step. We shall prove that the obtained makespan is optimal.

First Step: The greedy Algorithm 1 returns the optimal makespan for a PPP without any P3 (i.e., with $\beta = 0$), allocating the longest durations first. Consider now a PPP $C = (e, w)$ and β_{set}, let $e' = e \backslash \beta_{set}$ and $w' = w \backslash \beta_{set}$ (obtained by removing the elements of β_{set} from e and w). The new PPP $C' = (e', w')$ has by definition no P3, and it can be applied Algorithm 1. As we need them for the second step, the algorithm return durations $(D_i)_i$ for all planes. The optimal makespan of C' is obviously $\max_i(D_i)$.

Algorithm 1 Computing the optimal makespan when $\beta = 0$

$maxE \leftarrow 1$; $maxW \leftarrow t + 1$ {Start with longest durations both sides}
$D_i \leftarrow 0 \; \forall i = 1, \ldots, p$ {'Private' makespan for plane i}
$S_i \leftarrow EAST \; \forall i = 1, \ldots, p$ {Direction for next flight of plane i}
while $maxW \leq 2t - p$ **do**
  $k \leftarrow \text{ArgMin}_i(D_i)$ {Schedule next in line (in c_I or c_G)}
  if $S_k = EAST$ **then**
    $D_k \leftarrow D_k + 2d_{e_{maxE}}$
    $S_k \leftarrow WEST$; $maxE \leftarrow maxE + 1$;
  else
    $D_k \leftarrow D_k + 2d_{w_{maxW}}$
    $S_k \leftarrow EAST$; $maxW \leftarrow maxW + 1$;
  end if
end while
while $maxE \leq t$ **do**
  $k \leftarrow \underset{i, S_i = EAST}{\text{ArgMin}} (D_i)$
  $D_k \leftarrow D_k + 2d_{c_{maxE}}$
  $maxE \leftarrow maxE + 1$;
end while
return $(D_i)_i$ {All private makespans are needed for the second step}

Second step: This step dispatches the different P3 flights among the planes according to their own makespan $(D_i)_i$ of the first step, by sequentially assigning the longest P3 flight to the two planes with the smallest current flight times.

However, in P3, if the second plane lands in the central city before the person has been brought by the first plane, it has to wait, and it is then possible that the makespan of the plan is not simply a greedy distribution of all durations.

Consider a pattern P3 performed by planes p_1 et p_2, through city c_i. Their schedules will look like:

$$p_1 : c_I \rightarrow \ldots \rightarrow c_I \rightarrow \overset{\Diamond}{c_i} \rightarrow c_I \rightarrow \ldots \rightarrow c_G$$
$$p_2 : c_I \rightarrow \ldots \rightarrow c_G \rightarrow \overset{\blacklozenge}{c_i} \rightarrow c_G \rightarrow \ldots \rightarrow c_G$$

where $\Diamond$ the moment where p_1 drops the person in city c_i and $\blacklozenge$ is a possible waiting point for p_2. The idea of Step 2 is to place all $\Diamond$'s as early as possible in the plan, by scheduling the western parts of all P3s as soon as possible, and to place $\blacklozenge$ as late as possible, by scheduling the eastern parts of the P3s as late as possible. Let us consider only the planes that have to perform at least one P3.

1. For each plane, select the one with the maximum number of P3s to be performed. In case of tie, select the plane with longest P3 duration, or the plane with the latest current makespan.
2. Construct a partial schedule with only P1s and P2s.
3. For every 'not already started' P3, add its eastern part at the end of the schedule by descending order of durations.
4. For every 'already started' P3, add its western part at the beginning of the schedule by ascending order of durations.

Example: $t = 7$, $p = 3$, $d = (2, 4, 6), C = (3, 3, 2, 2, 2, 1, 1)(3, 3, 2, 1), 0 \leq \beta \leq 3$. Choosing $\beta_{set} = \{3, 2, 1\}$ leads to $C' = (3, 2, 2, 1)(3)$ with $t' = 4$. Solving C' (Step 1) gives the 'private' makespans D_i^1 in the table below. Adding the P3s gives the complete schedule, with 'private' makespans D_i^2.

p_i	D_i^1	D_i^2	Pat. 3
p_1	12	32	2
p_2	24	28	1
p_3	8	32	3

$p_3 : c_I \to c_2 \to c_G \to \overset{\blacklozenge 3}{c_3} \to c_G \to \overset{\blacklozenge 2}{c_2} \to c_G \to \overset{\blacklozenge 1}{c_1} \to c_G$

$p_2 : c_I \to \overset{\diamondsuit 1}{c_1} \to c_I \to c_2 \to c_G \to c_3 \to c_I \to c_2 \to c_G$

$p_3 : c_I \to \overset{\diamondsuit 3}{c_3} \to c_I \to \overset{\diamondsuit 2}{c_2} \to c_I \to c_3 \to c_G$

Hence, there is no waiting time within P3s, and the optimal makespan is 32. □

Proposition: The makespan returned by the algorithm above is optimal.

Proof: The incompressible time to transport all passengers, according to a given β_{set} is $T = 4 \sum_{i \in \beta_{set}} d_i + 2 \sum_{i \in \{e', w'\}} d_i$. A theoretical optimal plan with this pattern repartition is a plan without any waiting point for any plane. The above algorithm gives the optimal distribution of the set of times into p. Then, if a plan can be constructed with such a makespan, it is optimal for the PPP and the repartition of patterns. As it exists a method to construct such a plan, we can conclude that the algorithm is optimal for the PPP C and β_{set}. □

Complexity: In the worst case, for a given PPP, $w \subset e$ and $w_i \neq w_j$ if $i \neq j$. Hence, for each value of β there are $\binom{t-p}{\beta}$ possible β_{set}. As $0 \leq \beta \leq t - p$, we will perform 2^{t-p} *iterations* (i.e. determining the optimal makespan given a PPP and a β_{set}). The total worst-case complexity is hence $2^{t-p}\binom{n+t-1}{t}\binom{n+(t-p)-1}{t-p}$.

3 ZenoSolver

ZenoSolver is a C++11 software dedicated to generate and solve MultiZenoTravel instances. Firstly, it allows to tune every parameters in order to adjust the difficulty or to obtain different shapes for Pareto Front. In particular, vectors c and d are generated using two user-defined functions, f and g, such that $c_i = x_c f(i) + y_c$ and $d_i = x_d g(n - i) + y_d$, insuring that both objectives are conflicting. Second, ZenoSolver outputs the corresponding PDDL file, that can be directly used by any standard AI planner. Finally, ZenoSolver computes the true Pareto front using the algorithm described above, iterating over $E \times W$, storing, for each value of the total cost, the PPP with best makespan to date, but without explicitly constructing the set of admissible PPPs. Using the Greedy domination, ZenoSolver implements a pruning method that checks if the current PPP is dominated by any other PPP already stored. As the the optimal makespan is lower or equal to the upper bound M_S, this leads to a more efficient pruning. Indeed, as PPP are generated in an approximated increasing order, this avoid to iterate over the whole PPP set to check domination.

Determining if the current PPP is dominated can be performed in $O(h)$ where h is the number of different total costs. An obvious upper-bound on h is given

Table 1: Sample Large Instances: parameters & generation statistics.

Inst.	Cities	Pass.	Planes	Generating functions		Pareto#	h	PPP(k)	Iter.(k)	Time
1	20	6	2	$\frac{5}{2}i+\frac{(i \bmod 2)}{10}$	$\frac{5}{2}i+\frac{(i \bmod 2)}{10}$	409	4015	1568220	3317140	16h46
2	3	21	2			61	861	53	233	2006s
3	10	10	3			383				
4	200	3	2	$\sqrt{i}$	$\sqrt{i}$	538	4963	270680	3906	1845s
5	200	3	2	$\sqrt{i}$	i	399	797	270680	32066	1666s
6	8	26	25	$\sqrt{i}$	i	15	190	34176	60457	4240s

by $(2t-p)(\max_i(c_i)-\min_i(c_i))$. The memory requirement is also in $O(h)$, which is approximately the size of the Pareto Front (see Table 1).

3.1 Empirical Performances

Empirical Complexity: The number of iterations (Section 2.3) has worst case complexity $2^{t-p}\binom{n+t-1}{t}\binom{n+(t-p)-1}{t-p}$. Practically however, it is influenced not only by the number of PPPs but also by their structures: increasing n does not in general significantly increase the number of iterations (the upper-bound per PPP is 2^{t-p}, independent of n). However, increasing t increases that upper-bound, and dramatically increases the actual average number of iterations per PPP.

Pruning or not pruning? The efficiency of the pruning method seems to be instance-dependent. Fixing n, t and p, different generating functions result in different numbers of iterations and CPU times. The benefits of the pruning method strongly rely on the average number of iterations per PPP and thus, is more efficient while increasing t. Furthermore, increasing n while using pruning can degrade the performance, even if there are less iterations than PPPs.

There are however some clear cases for pruning, e.g. with $n = t = 9$: ZenoSolver requires 1.26 10^9 iterations and 2222 seconds without pruning. Adding pruning, for $f(i) = \sqrt{i}$ and $g(i) = i$ it requires only 119000 iterations and 26 seconds, but 36000 iterations but 53 seconds for $f(i) = log(i)$ and $g(i) = \sqrt{i}$.

3.2 Sample Large Instances

Figure 2 displays the Pareto Fronts of six large instances of various shapes and complexities (with zooms on interesting parts of the front), while Table 1 gives the parameters used by ZenoSolver to build them, as well as some statistics about their complexity (defined above). None is linear (though some seem to be at large scale). Also note the non-uniform distribution of the points in instances 3 and 4, and the few Pareto points of instance 6 in spite of the complexity of the instance (26 persons), due to the small ratio $\frac{p}{t}$.

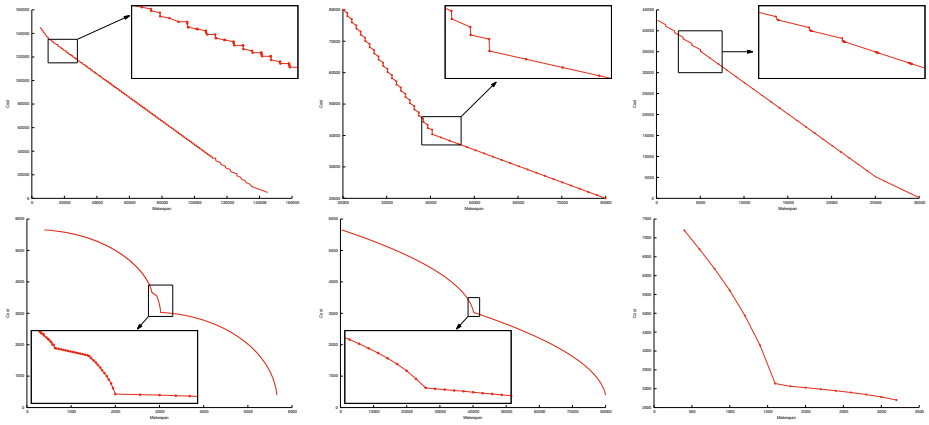

Fig. 2: True Pareto Fronts for instances 1 to 6 of Table 1.

4 Multi-objective Experiments

4.1 Divide-and-Evolve

Based on the Divide-and-Conquer paradigm, Divide-and-Evolve (DAE) is a generic hybrid evolutionary originally introduced in [5]. It splices the problem into a sequence of sub-problems that are hopefully easier to solve by some classical AI planer. Given a planning problem $\mathcal{P}_D(O, G)$ (on domain D with initial state O and target state T), the goal is to find an optimal sequence of states $S_0 = O, S_1, \ldots, S_n, S_{n+1} \subseteq T$ such that when solving the series of planning problems $\mathcal{P}_D(S_k, S_{k+1})$ $(k \in [0, n])$ with some given embedded planer, the sequential concatenation of all sub-plans is an optimal plan for the original problem $(G \subseteq T)$. An evolutionary algorithm using specific variation operators is used to evolve the sequence $(S_i)_{i\in[1,n]}$. The current version (see [1] for details) embeds the sub-optimal but very fast planner YAHSP [7].

4.2 Experimental Conditions

Previous work [3] showed that IBEA_{H^-} performed best among several competitors as the evolutionary engine for $\text{DAE}_{\text{YAHSP}}$. Hence all experiments have been conducted with IBEA_{H^-}. $\text{DAE}_{\text{YASHP}}$ internal parameters have been tuned thanks to PARAMILS using an **Hypervolume Difference Indicator** metric H^- as detailed in [2]. For each instance, 20 independent runs limited to 5400 seconds (1800 for instance 6) have been run. All performance assessments and comparisons have been done using PISA[7].

4.3 $\text{DAE}_{\text{YAHSP}}$ on Large Instances

The attainment surface on Instance 1 (Figure 3-left) shows a good fit to the true Pareto front, even though very few Pareto optima were actually reached

[7] http://www.tik.ee.ethz.ch/pisa/

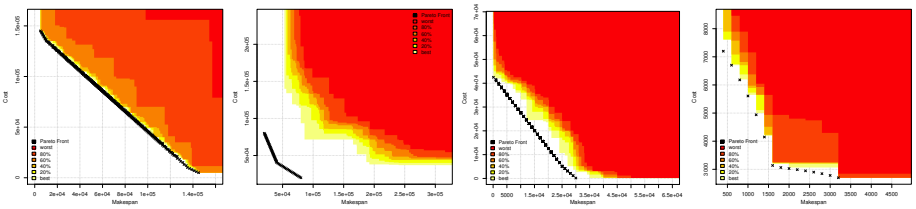

Fig. 3: Attainment Surfaces and true Pareto fronts for Instances 1, 2, 3 and 6.

(the darker, the larger the probability to reach that region - full white meaning that none of the 20 runs ever reached it). However, the attainment surfaces are uniformly distributed over the front. Surfaces on Instance 2 and 3 are far from the exact front and only 2 points are found for Instance 3 out of 383. On the opposite, even if with smaller budget, most of the actual Pareto optima are found for Instance 6, except on the most concave part.

We can notice that, even if n is higher for the Instance 6 than for the Instance 2, more planes results in a easier front to reach. This is quite surprising since the search space for DAE$_{\text{YAHSP}}$ is growing up with p. A reason could be the maximum number of states in a decomposition that has been fixed at 20 and could be too low to obtain easy-to-solve sub-problems, resulting on poor results on the Instance 2 and 3.

4.4 Pareto vs Weighted Sum Aggregation

Finally, let us have a quick look at some comparative results between the multi-objective version of DAE$_{\text{YAHSP}}$ and the single-objective version of DAE$_{\text{YAHSP}}$ on a weighted sum of the objectives. The chosen instance is a concave instance similar to Instance 4, but with only 30 cities (resulting in a Pareto Front made of 66 points). All experimental conditions are the same than in [2]. One aggregated run amounts to 11 independent runs, the weight parameter α taking values from 0 to 1 by step of 0.1. However, whereas each Pareto run had a budget of 5400s, the same budget was given to each of the α-runs (resulting in a total budget 11 times larger for each aggregated run).

Despite this big advantage, the attainment surfaces (Figure 4) show that in the case of Pareto approach, the exact Pareto Front is already delineated after 900s, even considering only the worst of the 20 runs. On the opposite, even the best of the 20 runs is still far from Pareto Front apart from a few points that lie in the convex parts of the front.

5 Discussion and Perspectives

The proposed benchmark suite opens the floor to deeper comparative experiments in a combinatorial domain where no ground truth existed for large instances as far as we know. In particular, beyond the basic weighted sum aggregation used here as a baseline (all α-runs are independent), deeper experiments

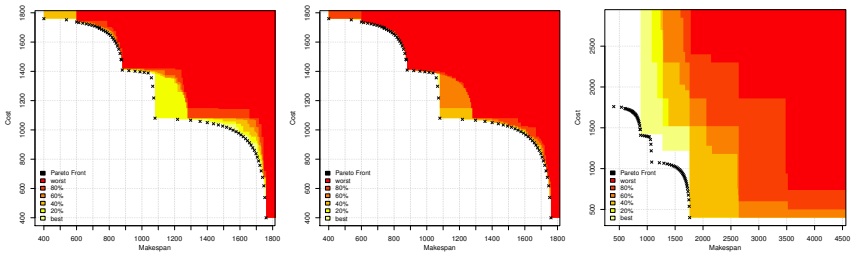

Fig. 4: Attainment surface for Pareto approach after 900s and 5400s (left, center) and for aggregation after 5400s (right) on (a small version of) Instance 4.

should be made with state-of-the-art decomposition methods in which the different components of the decomposition cooperate (e.g., from the MOEA/D family [8]). Thanks to the proposed MultiZenoTravel benchmark and ZenoSolver algorithm (available at `https://descarwin.lri.fr`), such experiments can now be made with the knowledge of the ground truth (the Pareto front). In the AI planning domain, this will hopefully lead to sound comparisons between Pareto and non-Pareto planners (see e.g., [6]).

References

1. Bibaï, J., Savéant, P., Schoenauer, M., Vidal, V.: An Evolutionary Metaheuristic Based on State Decomposition for Domain-Independent Satisficing Planning. In: R. Brafman et al. (ed.) Proc. of the 14^{th} ICAPS. pp. 18–25. AAAI Press (2010)
2. Khouadjia, M.R., Schoenauer, M., Vidal, V., Dréo, J., Savéant, P.: Multi-Objective AI Planning: Comparing Aggregation and Pareto Approaches. In: Middendorf, M., Blum, C. (eds.) Proc. EvoCOP. pp. 202–213. LNCS 7832, Springer Verlag (2013)
3. Khouadjia, M.R., Schoenauer, M., Vidal, V., Dréo, J., Savéant, P.: Multi-Objective AI Planning: Evaluating DAE-YAHSP on a Tunable Benchmark. In: R.C. Purshouse et al. (ed.) Proc. EMO. pp. 36–50. LNCS 7811, Springer Verlag (2013)
4. Khouadjia, M.R., Schoenauer, M., Vidal, V., Dréo, J., Savéant, P.: Pareto-Based Multiobjective AI Planning. In: Rossi, F. (ed.) Proc. IJCAI. AAAI Press (2013)
5. Schoenauer, M., Savéant, P., Vidal, V.: Divide-and-Evolve: a New Memetic Scheme for Domain-Independent Temporal Planning. In: Gottlieb, J., Raidl, G.R. (eds.) Proc. EvoCOP. pp. 247–260. LNCS 3906, Springer Verlag (2006)
6. Sroka, M., Long, D.: Exploring Metric Sensitivity of Planners for Generation of Pareto Frontiers. In: Kersting, K., Toussaint, M. (eds.) The Sixth "Starting Artificial Intelligence Research" Symposium (STAIRS 2012). pp. 306–317. IOS Press (2012)
7. Vidal, V.: A Lookahead Strategy for Heuristic Search Planning. In: R. Brafman et al. (ed.) Proc. of the 14^{th} ICAPS. pp. 150–159. AAAI Press (2004)
8. Zhang, Q., Li, H.: A multi-objective evolutionary algorithm based on decomposition. IEEE Transactions on Evolutionary Computation 11(6), 712–731 (2007)