Quality Measures of Parameter Tuning for Aggregated Multi-Objective Temporal Planning

Mostepha Redouane Khouadjia, Marc Schoenauer, Vincent Vidal, Johann Dréo, Pierre Savéant

Quality Measures of Parameter Tuning for Aggregated Multi-Objective Temporal Planning

M. R. Khouadjia[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
{mostepha-redouane.khouadjia,marc.schoenauer}@inria.fr,
[2] ONERA-DCSD, Toulouse, France
Vincent.Vidal@onera.fr
[3] THALES Research & Technology, Palaiseau, France**
{johann.dreo,pierre.saveant}@thalesgroup.com

Abstract. Parameter tuning is recognized today as a crucial ingredient when tackling an optimization problem. Several meta-optimization methods have been proposed to find the best parameter set for a given optimization algorithm and (set of) problem instances. When the objective of the optimization is some scalar quality of the solution given by the target algorithm, this quality is also used as the basis for the quality of parameter sets. But in the case of multi-objective optimization by aggregation, the set of solutions is given by several single-objective runs with different weights on the objectives, and it turns out that the hypervolume of the final population of each single-objective run might be a better indicator of the global performance of the aggregation method than the best fitness in its population. This paper discusses this issue on a case study in multi-objective temporal planning using the evolutionary planner DaE$_{\text{YAHSP}}$ and the meta-optimizer ParamILS. The results clearly show how ParamILS makes a difference between both approaches, and demonstrate that indeed, in this context, using the hypervolume indicator as ParamILS target is the best choice. Other issues pertaining to parameter tuning in the proposed context are also discussed.

1 Introduction

Parameter tuning is now well recognized as a mandatory step when attempting to solve a given set of instance of some optimization problem. All optimization algorithms behave very differently on a given problem, depending on their parameter values, and setting the algorithm parameters to the correct value can make the difference between failure and success. This is equally true for deterministic complete algorithms [1] and for stochastic approximate algorithms [2, 3]. Current approaches range from methods issued from racing-like methods [4, 5] to meta-optimization, using Gaussian Processes [6], Evolutionary Algorithms [7] or Iterated Local Search [8]. All these methods repeatedly call the target

** This work is being partially funded by the French National Research Agency under the research contract DESCARWIN (ANR-09-COSI-002).

algorithm and record their performance on the given problem instances.

Quality criteria for parameter sets usually involve the solution quality of the target algorithm and the time complexity of the algorithm, and, in the case of a set of problem instances, statistics of these quantities over the whole set. The present work is concerned with the case of instance-based parameter tuning (i.e. a single instance is considered), and the only goal is the quality of the final solution, for a fixed computational budget. In this context, the objective of the meta-optimizer is generally also directly based on the quality of the solution.

However, things are different in the context of multi-objective optimization, when using an aggregation method, i.e. optimizing several linear combinations of the objectives, gathering all results into one single set, and returning the non-dominated solutions within this set as an approximation of the Pareto front. Indeed, the objective of each single-objective run is the weighted sum of the problem objectives, and using this weighted sum as the objective for parameter tuning seems to be the most straightforward approach. However, the objective of the whole algorithm is to approximate the Pareto front of the multi-objective problem. And the hypervolume indicator [9] has been proved to capture into a single real value the quality of a set as an approximation of the Pareto front. Hence an alternative strategy could be to tune each single-objective run so as to optimize the hypervolume of its final population, as a by-product of optimizing the weighted sum of the problem objectives. This paper presents a case study of the comparison of both parameter-tuning approaches described above for the aggregated multi-objective approach, in the domain of AI planning [10]. This domain is rapidly introduced in Section 2. In particular, MultiZeno, a tunable multi-objective temporal planning benchmark inspired by the well-known `zeno` IPC logistic domain benchmark, is described in detail. Section 3 introduces Divide-and-Evolve ($\text{DaE}_{\text{YAHSP}}$), a single-objective evolutionary AI planning algorithm that has obtained state-of-the-art results on different planning benchmark problems [11], and won the deterministic temporal satisficing track at IPC 2011 competition[4]. Section 4 details the experimental conditions of the forthcoming experiments, introduces the parameters to be optimized, the aggregation method, the meta-optimizer ParamILS, the parameter tuning method that has been chosen here [8], and precisely defines the two quality measures to be used by ParamILS in the experiments: either the best fitness or the global hypervolume of its final population. Section 5 details the experimental results obtained by $\text{DaE}_{\text{YAHSP}}$ for solving MultiZeno instances using these two quality measures. The values of the parameters resulting from the ParamILS runs are discussed, and the quality of the approximations of the Pareto front given by both approaches are compared, and the differences analyzed.

2 AI Planning

An AI Planning problem (see e.g. [10]) is defined by a set of predicates, a set of actions, an initial state and a goal state. A state is a set of non-exclusive

[4] See `http://www.plg.inf.uc3m.es/ipc2011-deterministic`

instantiated predicates, or (Boolean) atoms. An action is defined by a set of *pre-conditions* and a set of *effects*: the action can be executed only if all pre-conditions are true in the current state, and after an action has been executed, the effects of the action modify the state: the system enters a new state. A plan is a sequence of actions, and a *feasible plan* is a plan such that executing each action in turn from the initial state puts the systems into the goal state. The goal of (single objective) AI Planning is to find a feasible plan that minimizes some quantity related to the actions: number of actions for STRIPS problems, sum of action costs in case actions have different costs, or makespan in the case of temporal planning, when actions have a duration and can eventually be executed in parallel. All these problems are P-SPACE.

A simple planning problem in the domain of logistics (inspired by the well-known ZENO problem of IPC series) is given in Figure 1: the problem involves cities, passengers, and planes. Passengers can be transported from one city to another, following the links on the figure. One plane can only carry one passenger at a time from one city to another, and the flight duration (number on the link) is the same whether or not the plane carries a passenger (this defines the *domain* of the problem). In the simplest non-trivial *instance* of such domain, there are 3 passengers and 2 planes. In the initial state, all passengers and planes are in `city 0`, and in the goal state, all passengers must be in `city 4`. The not-so-obvious optimal solution has a total makespan of 8 and is left as a teaser for the reader.

AI Planning is a very active field of research, as witnessed by the success of the ICAPS series of yearly conferences (`http://icaps-conferences.org`), and its biannual competition IPC, where the best planners in the world compete on a set of problems. This competition has lead the researchers to design a common language to describe planning problems, PDDL (Planning Domain Definition Language). Two main categories of planners can be distinguished: *exact planners* are guaranteed to find the optimal solution ... if given enough time; *satisficing planners* give the best possible solution, but with no optimality guarantee.

2.1 Multi-Objective AI Planning

Most existing work in AI Planning involves one single objective, even though real-world problems are generally multi-objective (e.g., optimizing the makespan while minimizing the cost, two contradictory objectives). An obvious approach to Multi-Objective AI planning is to aggregate the different objectives into a single objective, generally a fixed linear combination (weighted sum) of all objectives. The single objective is to be minimized, and the weights have to be positive (resp. negative) for the objectives to be minimized (resp. maximized) in the original problem. The solution of one aggregated problem is Pareto optimal if all weights are non-zero, or the solution is unique [12]. It is also well-known that whatever the weights, the optimal solution of an aggregated problem is always on the convex parts of the Pareto front. However, some adaptive techniques of the aggregation approach have been proposed, that partially address this drawback [13] and are able to identify the whole Pareto front by maintaining an archive of non-dominated solutions ever encountered during the search.

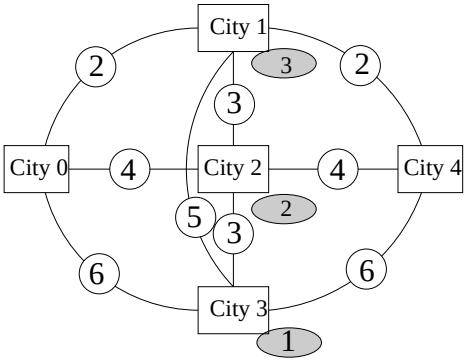


Fig. 1: A schematic view of MultiZeno, a simple benchmark transportation domain: Flight durations of available routes are attached to the corresponding edges, costs are attached to landing in the central cities (in grey circles).

Despite the fact that pure multi-objective approaches like e.g., dominance-based approaches, are able to generate a diverse set of Pareto optimal solutions, which is a serious advantage, aggregation approaches are worth investigating, as they can be implemented seamlessly from almost any single-objective algorithm, and rapidly provide at least part of the Pareto front at a low man-power cost.

This explains why all works in multi-objective AI Planning used objective aggregation, to the best of our knowlege[5]. Early works used some twist in PDDL 2.0 [16–18]. PDDL 3.0, on the other hand, explicitly offered hooks for several objectives [19], and a new track of IPC was dedicated to aggregated multiple objectives: the "net-benefit" track took place in 2006 [20] and 2008 [21], ... but was canceled in 2011 because of a too small number of entries.

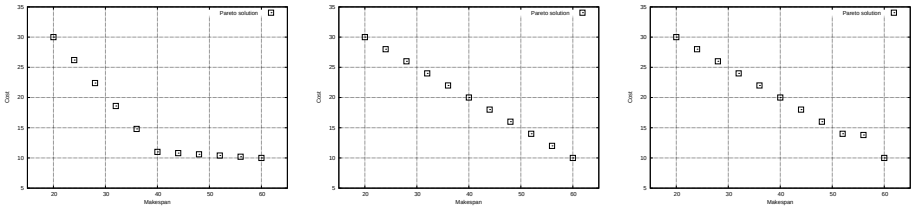
2.2 Tunable Benchmarks for Multi-Objective Temporal Planning

For the sake of understandability, it is important to be able to experiment with instances of tunable complexity for which the exact Pareto fronts are easy to determine, and this is the reason for the design of the MultiZeno benchmark family. The reader will have by now solved the little puzzle illustrated in Figure 1, and found the solution with makespan 8, whose rationale is that no plane ever stays idle. In order to turn this problem into a not-too-unrealistic logistics multi-objective problem, some costs are added to all 3 central cities (1 to 3). This leads to the MultiZeno$_{Cost}$ problems, where the second objective is additive: each plane has to pay the corresponding tax every time it lands in that city[6].

In the simplest instance, MultiZeno3, involving 3 passengers only, there are 3 obvious points that belong to the Pareto Front, using the small trick described

[5] with the exception of an early proof-of-concept for DaEx [14] and its recently accepted follow-up [15].

[6] In the MultiZeno$_{Risk}$ problem, not detailed here, the second objective is the risk: its maximal value ever encountered is to be minimized.



(a) cost(`city2`)=1.1 (b) cost(`city2`)=2 (c) cost(`city2`)=2.9

Fig. 2: The exact Pareto Fronts for the MultiZeno6 problem for different values of cost(`city2`) (all other values as in Figure 1).

above, and going respectively through `city1`, `city 2` or `city 3`. The values of the makespans are respectively 8, 16 and 24, and the values of the costs are, for each solution, 4 times the value of the single landing tax. However, different cities can be used for the different passengers, leading to a Pareto Front made of 5 points, adding points (12,10) and (20,6) to the obvious points (8,12), (16,8), and (24,4).

There are several ways to make this first simple instance more or less complex, by adding passengers, planes and central cities, and by tuning the different values of the makespans and costs. In the present work, only additional bunches of 3 passengers have been considered, in order to be able to easily derive some obvious Pareto-optimal solutions as above, using several times the little trick to avoid leaving any plane idle. This lead to the MultiZeno6, and MultiZeno9 instances, with respectively 6 and 9 passengers. The Pareto front of MultiZeno6 on domain described by Figure 1 can be seen on Figure 2-b. The other direction for complexification that has been investigated in the present work is based on the modification of the cost value for `city 2`, leading to different shapes of the Pareto front, as can be seen on Figure 2-a and 2-c. Further work will investigate other directions of complexification of this very rich benchmark test suite.

3 Divide-and-Evolve

Let $\mathcal{P}_D(I, G)$ denote the planning problem defined on domain D (the predicates, the objects, and the actions), with initial state I and goal state G. In STRIPS representation model [22], a state is a list of Boolean atoms defined using the predicates of the domain, instantiated with the domain objects.

In order to solve $\mathcal{P}_D(I, G)$, the basic idea of DaE$_X$ is to find a sequence of states $S_1, \ldots, S_n$, and to use some embedded planner X to solve the series of planning problems $\mathcal{P}_D(S_k, S_{k+1})$, for $k \in [0, n]$ (with the convention that $S_0 = I$ and $S_{n+1} = G$). The generation and optimization of the sequence of states $(S_i)_{i\in[1,n]}$ is driven by an evolutionary algorithm. After each of the sub-problems $\mathcal{P}_D(S_k, S_{k+1})$ has been solved by the embedded planner, the concatenation of the corresponding plans (possibly compressed to take into account possible parallelism in the case of temporal planning) is a solution of the initial problem.

In case one sub-problem cannot be solved by the embedded solver, the individual is said *unfeasible* and its fitness is highly penalized in order to ensure that feasible individuals always have a better fitness than unfeasible ones, and are selected only when there are not enough feasible individual. A thorough description of DAE$_X$ can be found in [11]. The rest of this section will briefly recall the evolutionary parts of DAE$_X$.

3.1 Representation, Initialization, and Variation Operators

Representation An individual in DAE$_X$ is a variable-length list of states of the given domain. However, the size of the space of lists of complete states rapidly becomes untractable when the number of objects increases. Moreover, goals of planning problems need only to be defined as partial states, involving a subset of the objects, and the aim is to find a state such that all atoms of the goal state are true. An individual in DAE$_X$ is thus a variable-length list of partial states, and a partial state is a variable-length list of atoms (instantiated predicates).
Initialization Previous work with DAE$_X$ on different domains of planning problems from the IPC benchmark series have demonstrated the need for a very careful choice of the atoms that are used to build the partial states [23]. The method that is used today to build the partial states is based on a heuristic estimation, for each atom, of the earliest time from which it can become true [24], and an individual in DAE$_X$ is represented by a variable-length time-consistent sequence of partial states, and each partial state is a variable-length list of atoms that are not pairwise mutually exclusive (aka *mutex*), according to the partial mutex relation computed by the embedded planner.
Crossover and mutation operators: are applied with respective user-defined probabilities `Proba-cross` and `Proba-mut`. They are defined on the DAE$_X$ representation in a straightforward manner - though constrained by the heuristic chronology and the partial mutex relation between atoms. **One-point crossover** is adapted to variable-length representation: both crossover points are independently chosen, uniformly in both parents. Only one offspring is kept, the one that respects the approximate chronological constraint on the successive states.
Four mutation operators are included, and operate either at the individual level, by adding (`addGoal`) or removing (`delGoal`) an intermediate state, or at the state level by adding (`addAtom`) or removing (`delAtom`) some atoms in a uniformly chosen state. The choice among these mutations is made according to user-defined relative weights, named `w-MutationName` - see Table 1.

3.2 Hybridization and Multi-Objectivization

DAE$_X$ uses an external embedded planner to solve in turn the sequence of subproblems defined by the ordered list of partial states. Any existing planner can in theory be used. However, there is no need for an optimality guarantee when solving the intermediate problems in order for DAE$_X$ to obtain good quality results [11]. Hence, and because a very large number of calls to this embedded planner are necessary for a single fitness evaluation, a sub-optimal but fast

planner was found to be the best choice: YAHSP [25] is a lookahead strategy planning system for sub-optimal planning which uses the actions in the relaxed plan to compute reachable states in order to speed up the search process. Because the rationale for DAE$_X$ is that all sub-problems should hopefully be easier than the initial global problem, and for computational performance reason, the search capabilities of the embedded planner YAHSP are limited by setting a maximal number of nodes that it is allowed to expand to solve any of the sub-problems (see again [11] for more details).

However, even though YAHSP, like all known planners to-date, is a single-objective planner, it is nevertheless possible since PDDL 3.0 to add in a PDDL domain file other quantities (aka *Soft Constraints* or *Preferences* [19]) that are simply computed throughout the execution of the final plan, without interfering with the search. Two strategies are then possible for YAHSP in the two-objective context of MULTIZENO: it can optimize either the makespan or the cost, and simply compute the other quantity (cost or makespan) along the solution plan. The corresponding strategie will be referred to as YAHSP$_{makespan}$ and YAHSP$_{cost}$.

In the multi-objective versions of DAE$_{\text{YAHSP}}$ the choice between both strategies is governed by user-defined weights, named respectively W-makespan and W-cost (see table 1). For each individual, the actual strategy is randomly chosen according to those weights, and applied to all subproblems of the individual.

4 Experimental Conditions

The aggregation method for multi-objective optimization runs in turn a series of single-objective problems. The fitness of each of these problems is defined using a single positive parameter α. In the following, F_α will denote $\alpha * \text{makespan} + (1 - \alpha) * \text{cost}$, and DAE$_{\text{YAHSP}}$ run optimizing F_α will be called the α-run. Because the range of the makespan values is approximately twice as large as that of the cost, the following values of α have been used instead of regularly spaced values: $0, 0.05, 0.1, 0.3, 0.5, 0.55, 0.7, 1.0$. One "run" of the aggregation method thus amounts to running the corresponding eight α-runs, and returns as the approximation of the Pareto front the set of non-dominated solutions among the merge of the eight final populations.

ParamILS [8] is used to tune the parameters of DAE$_{\text{YAHSP}}$. PARAMILS uses the simple Iterated Local Search heuristic [26] to optimize parameter configurations, and can be applied to any parameterized algorithm whose parameters can be discretized. PARAMILS repeats local search loops from different random starting points, and during each local search loops, modifies one parameter at a time, runs the target algorithm with the new configuration and computes the quality measure it aims at optimizing, accepting the new configuration if it improves the quality measure over the current one.

The most prominent parameters of DAE$_{\text{YAHSP}}$ that have been subject to optimization can be seen in Table 1.

Quality Measures for ParamILS: The goal of the experiments presented here is to compare the influence of two quality measures of PARAMILS for the

Parameters	Range	Description
W-makespan W-cost	0,1,2,3,4,5	Weighting for optimizing makespan during the search Weighting for optimizing cost during the search
Pop-size	30,50,100,200,300	Population Size
Proba-cross Proba-mut	0.0,0.1,0.2,0.5,0.8,1.0	Probability (at population level) to apply crossover Probability (at population level) to apply one mutation
w-addAtom w-addGoal w-delAtom w-delGoal	0,1,3,5,7,10	Relative weight of the addAtom mutation Relative weight of the addGoal mutation Relative weight of the delAtom mutation Relative weight of the delGoal mutation
Proba-change Proba-delatom	0.0,0.1,0.2,0.5,0.8,1.0	Probability to change an atom in addAtom mutation Average probability to delete an atom in delAtom mutation
Radius	1,3,5,7,10	Number of neighbour goals to consider in addGoal mutation

Table 1: Set of DAE parameters and their discretizations for PARAMILS, leading to approx. $1.5 \cdot 10^9$ possible configurations.

aggregated $\text{DAE}_{\text{YAHSP}}$ on MULTIZENO instances. In $\text{Aggreg}_{Fitness}$, the quality measure used by PARAMILS to tune the α-run of $\text{DAE}_{\text{YAHSP}}$ is F_α, the fitness also used by the target α-run. In Aggreg_{Hyper}, PARAMILS uses, for each of the α-run, the same quality measure, i.e., the unary hypervolume [27] of the final population of the α-run w.r.t. the exact Pareto front of the problem at hand (or its best available approximation when it is not available). The lower the better (a value of 0 indicates that the exact Pareto front has been reached).

Implementation: Algorithms have been implemented within the PARADISEO-MOEO framework[7]. All experiments were performed on the MULTIZENO3, MULTIZENO6, and MULTIZENO9 instances. The first objective is the makespan, and the second objective is the cost. The values of the different flight durations (makespans) and costs are those given on Figure 1 except otherwise stated.

Performance Assessment and Stopping Criterion For all experiments, 11 independent runs were performed. Note that all the performance assessment procedures, including the hypervolume calculations, have been achieved using the PISA performance assessment tool suite[8]. The main quality measure used here to compare Pareto Fronts is, as above, the unary hypervolume I_{H^-} [27] of the set of non-dominated points output by the algorithms with respect to the complete true Pareto front. For aggregated runs, the union of all final populations of the α-runs for the different values of α is considered the output of the complete 'run'.

However, and because the true front is known exactly, and is made of a few scattered points (at most 17 for MULTIZENO9 in this paper), it is also possible to visually monitor, for each point of the front, the ratio of actual runs (out of 11) that discovered it at any given time. This allows some other point of view on the comparison between algorithms, even when none has found the whole Pareto front. Such *hitting plots* will be used in the following, together with more classical plots of hypervolume vs computational effort. In any case,

[7] `http://paradiseo.gforge.inria.fr/`
[8] `http://www.tik.ee.ethz.ch/pisa/`

	Hypervolume								Fitness								IBEA_H
α	0.0	0.05	0.1	0.3	0.5	0.55	0.7	1.0	0.0	0.05	0.1	0.3	0.5	0.55	0.7	1.0	
W-makespan	3	3	3	2	2	2	0	0	0	0	0	0	5	5	1	4	1
W-cost	0	0	0	4	3	3	3	4	2	4	4	2	1	1	0	1	1
Pop-size	100	100	200	200	100	100	200	300	200	300	300	100	100	100	100	100	30
Proba-cross	0.5	1.0	1.0	1.0	1.0	1.0	1.0	0.8	0.8	0.1	0.2	0.2	0.5	0.8	0.2	0.1	0.2
Proba-mut	0.8	0.2	0.2	0.2	0.2	0.2	0.5	1.0	0.5	1.0	0.5	1.0	1.0	1.0	0.8	1.0	0.2
w-addatom	1	1	5	5	5	5	5	5	3	10	3	10	3	5	5	3	7
w-addgoal	5	1	5	7	7	7	0	0	3	7	10	10	10	10	10	10	10
w-delatom	3	3	1	5	10	1	7	0	3	5	10	0	10	10	3	1	5
w-delgoal	5	5	5	7	10	3	7	10	1	7	1	1	0	1	10	1	5
Proba-change	0.5	0.5	0.5	1.0	0.8	0.5	0.5	0.8	0.8	0.8	0.8	0.5	0.0	1.0	0.8	0.5	1.0
Proba-delatom	0.1	0.0	0.0	0.1	0.5	0.5	0.5	0.0	0.1	0.8	0.0	0.8	1.0	0.8	0.5	0.5	1.0
Radius	3	3	10	1	7	7	1	5	3	3	1	3	5	5	10	3	5

Table 2: ParamILS results: Best parameters for $\text{DAE}_{\text{YAHSP}}$ on MultiZeno6

when comparing different approaches, statistical significance tests are made on the hypervolumes, using Wilcoxon signed rank test with 95% confidence level.

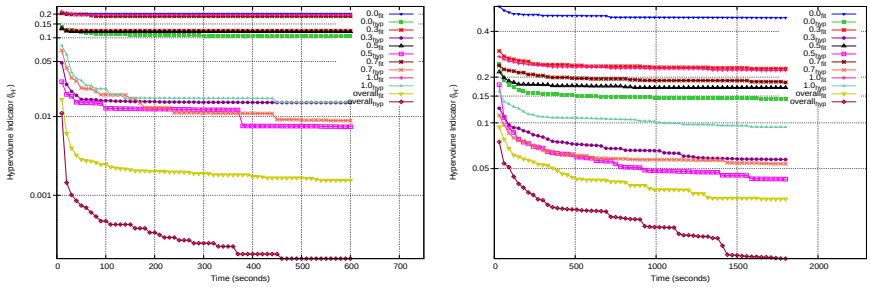
Finally, because different fitness evaluations involve different number calls to YAHSP – and because YAHSP runs can have different computational costs too, depending on the difficulty of the sub-problem being solved – the computational efforts will be measured in terms of CPU time and not number of function evaluations – and that goes for the stopping criterion: The absolute limits in terms of computational efforts were set to 300, 600, and 1800 seconds respectively for MultiZeno3, MultiZeno6, and MultiZeno9. The stopping criterion for ParamILS was likewise set to a fixed wall-clock time: 48h (resp. 72h) for MultiZeno3 and 6 (resp. MultiZeno9), corresponding to 576, 288, and 144 parameter configuration evaluations per value of α for MultiZeno3, 6 and 9 respectively.

5 Experimental Results

5.1 ParamILS Results

Table 2 presents the optimal values for $\text{DAE}_{\text{YAHSP}}$ parameters of Table 1 found by ParamILS in both experiments, for all values of α - as well as for the multi-objective version of $\text{DAE}_{\text{YAHSP}}$ presented in [15] (last column, entitled $IBEA_H$).

The most striking and clear conclusion regards the weights for the choice of YAHSP strategy (see Section 3.2) W-makespan and W-cost. Indeed, for the Aggreg_{Hyper} approach, ParamILS found out that YAHSP should optimize only the makespan (W-cost = 0) for small values of α, and only the cost for large values of α while the exact opposite is true for the $\text{Aggreg}_{Fitness}$ approach. Remember that small (resp. large) values of α correspond to an aggregated fitness having all its weight on the cost (resp. the makespan). Hence, during the 0- or 0.5-runs, the fitness of the corresponding α-run is pulling toward minimizing the cost: but for the Aggreg_{Hyper} approach, the best choice for YAHSP strategy, as identified by ParamILS, is to minimize the makespan (i.e., setting W-cost to 0): as a result, the population has a better chance to remain diverse, and



(a) MultiZeno6 (b) MultiZeno9

Fig. 3: Evolution of the Hypervolume for both approaches, for all α-runs and overall, on MultiZeno instances. Warning: Hypervolume is in log scale, and the X-axis is not the value 0, but 6.7 10^{-5} for MultiZeno6 and 0.0125 for MultiZeno9.

hence to optimize the hypervolume, i.e., ParamILS quality measure. In the same situation (small α), on the opposite, for $\text{Aggreg}_{Fitness}$, ParamILS has identified that the best strategy for YAHSP is to also favor the minimization of the cost, setting W-makespan to zero. The symmetrical reasoning can be applied to the case of large values of α. For the multi-objective version of $\text{DAE}_{\text{YAHSP}}$ (IBEA column in Table 2), the best strategy that ParamILS came up with is a perfect balance between both strategies, setting both weights to 1.

The values returned by ParamILS for the other parameters are more difficult to interpret. It seems that large values of `Proba-mut` are preferable for Aggreg_{Hyper} for α set to 0 or 1, i.e. when the $\text{DAE}_{\text{YAHSP}}$ explores the extreme sides of the objective space – more mutation is needed to depart from the boundary of the objective space and cover more of its volume. Another tendancy is that ParamILS repeatedly found higher values of `Proba-cross` and lower values of `Proba-mut` for Aggreg_{Hyper} than for $\text{Aggreg}_{Fitness}$. Together with large population sizes (compared to the one for IBEA for instance), the 1-point crossover of $\text{DAE}_{\text{YAHSP}}$ remains exploratory for a long time, and leads to viable individuals that can remain in the population even though they don't optimize the α-fitness, thus contributing to the hypervolume. On the opposite, large mutation rate is preferable for $\text{Aggreg}_{Fitness}$ as it increases the chances to hit a better fitness, and otherwise generates likely non- viable individuals that will be quickly eliminated by selection, making $\text{DAE}_{\text{YAHSP}}$ closer from a local search. The values found for IBEA, on the other hand, are rather small – but the small population size also has to be considered here: because it aims at exploring the whole objective space in one go, the most efficient strategy for IBEA is to make more but smaller steps, in all possible directions.

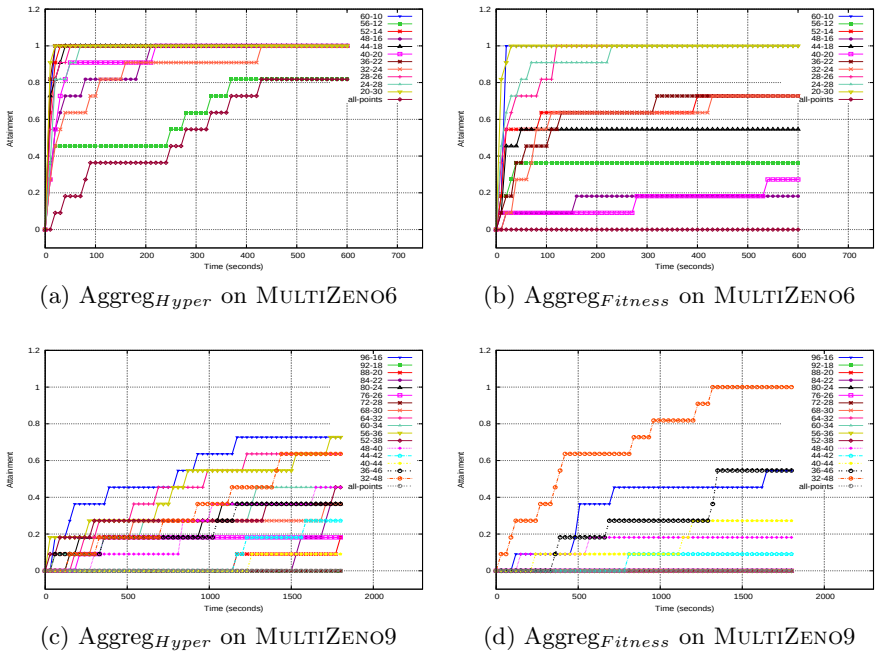
5.2 Comparative Results

Figure 3 represents the evolution during the course of the runs of the hypervolumes (averaged over the 11 independent runs) of some of the (single-objective) α-runs, for both methods together (labelled α_{hyp} or α_{fit}), as well as the evolution of the overall hypervolume, i.e., the hypervolume covered by the union of all populations of the different α-runs as a function of CPU time. Only the results on MultiZeno6 and MultiZeno9 are presented here, but rather similar behaviors can be observed for the two approaches on these two instances, and similar results were obtained on MultiZeno3, though less significantly different.

First of all, Aggreg_{Hyper} appears as a clear winer against $\text{Aggreg}_{Fitness}$, as confirmed by the Wilcoxon test with 95% confidence: On both instances, the two lowest lines are the results of the overall hypervolume for, from bottom to top, Aggreg_{Hyper} and $\text{Aggreg}_{Fitness}$, that reach respectiveley values of $6.7\ 10^{-5}$ and 0.015 on MultiZeno6 and 0.0127 and 0.03155 on MultiZeno9. And for each value of α, a similar difference can be seen. Another remark is that the central values of α (0.5, 0.7 and 0.3, in this order) outperform the extreme values (1 and 0, in this order): this is not really surprising, considering that these runs, that optimize a single objective (makespan or cost), can only spread in one direction, while more 'central' values allow the run to cover more volume around their best solutions. Finally, in all cases, the 0-runs perform significantly worse than the corresponding 1-runs, but this is probably only due to the absence of normalization between both objectives.

Another comparative point of view on the convergence of both aggregation approaches is given by the hitting plots of Figure 4. These plots represent, for each point of the true Pareto front, the ratio along evolution of the runs (remember that one 'run' represent the sum of the eight α-runs, see Section 4) that reached that point, for all three instances MultiZeno{3,6,9}. On MultiZeno3 (results not shown here for space reasons), only one point, $(20, 6)$, is not found by 100% of the runs. But it is found by 10/11 runs by Aggreg_{Hyper} and only by 6/11 runs by $\text{Aggreg}_{Fitness}$. On MultiZeno6, the situation is even clearer in favor of Aggreg_{Hyper}: Most points are found very rapidly by Aggreg_{Hyper}, and only point $(56, 12)$ is not found by 100% of the runs (it is missed by 2 runs); on the other hand, only 4 points are found by all α-runs of $\text{Aggreg}_{Fitness}$, the extreme makespan $(60, 10)$, and the 3 extreme costs $(20, 30)$, $(24, 28)$, and $(28, 26)$. The other points are discovered by different runs ... but overall, not a single run discovers all 11 points. Finally, the situation is even worse in the MultiZeno9 case: only 6 points (out of 17) are ever discovered by $\text{Aggreg}_{Fitness}$, while Aggreg_{Hyper}somehow manages to hit 12 different points. Hence again, no method does identify the full Pareto front.

But take a look at Figure 5, that displays the union of the 11 Pareto front returned by the aggregated runs, for both Aggreg_{Hyper} and $\text{Aggreg}_{Fitness}$. No big difference is observed on MultiZeno6, except maybe a higher diversity away from the Pareto front for Aggreg_{Hyper}. On the other hand, the difference is clear on MultiZeno9, where $\text{Aggreg}_{Fitness}$ completely misses the center of the Pareto-delimited region of the objective space.



(a) Aggreg$_{Hyper}$ on MultiZeno6 (b) Aggreg$_{Fitness}$ on MultiZeno6

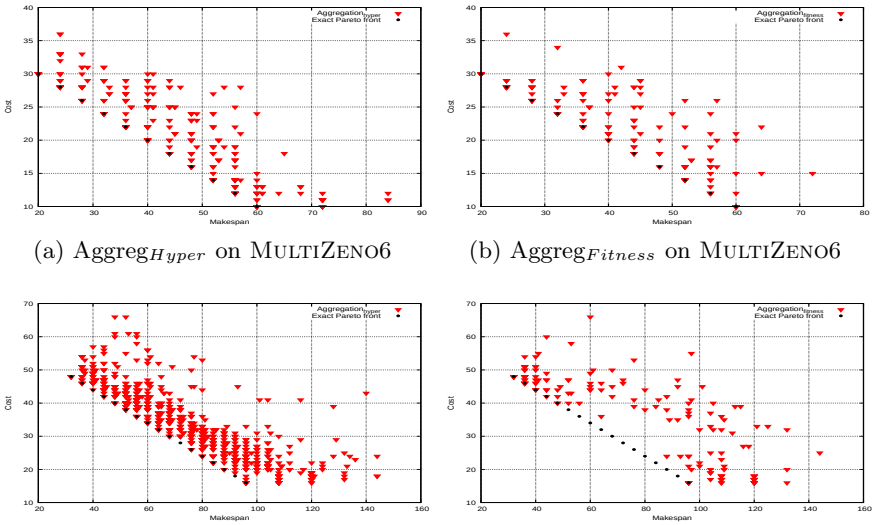
(c) Aggreg$_{Hyper}$ on MultiZeno9 (d) Aggreg$_{Fitness}$ on MultiZeno9

Fig. 4: Hitting plots on the 3 MultiZeno instances.

Preliminary runs have been made with the two other instances presented in Section 2.2, where the costs of `city2` have changed, respectively to 1.1 and 2.9, giving the Pareto fronts that are displayed in Figure 2. However, no specific parameter tuning was done for these instances, and all parameters have been carried on from the ParamILS runs on the corresponding MultiZeno instance where the cost of `city2` is 2. First, it is clear that the overall performance of both aggregation methods is rather poor, as none ever finds the complete Pareto front in the 1.1 case, and only one run out of 11 finds it in the 2.9 case. Here again, only the extreme points are reliably found by both methods. Second, the advantage of Aggreg$_{Hyper}$ over Aggreg$_{Fitness}$ is not clear any more: some points are even found more often by the latter. Finally, and surprisingly, the ankle point in the case 1.1 (Figure 2-a) is not found as easily as it might have seemed; and the point on the concave part of the case 2.9 (point $(56, 22.8)$, see Figure 2-c) is nevertheless found by respectively 9 and 4 runs, whereas aggregation approaches should have difficulties to discover such points.

6 Conclusion and Perspectives

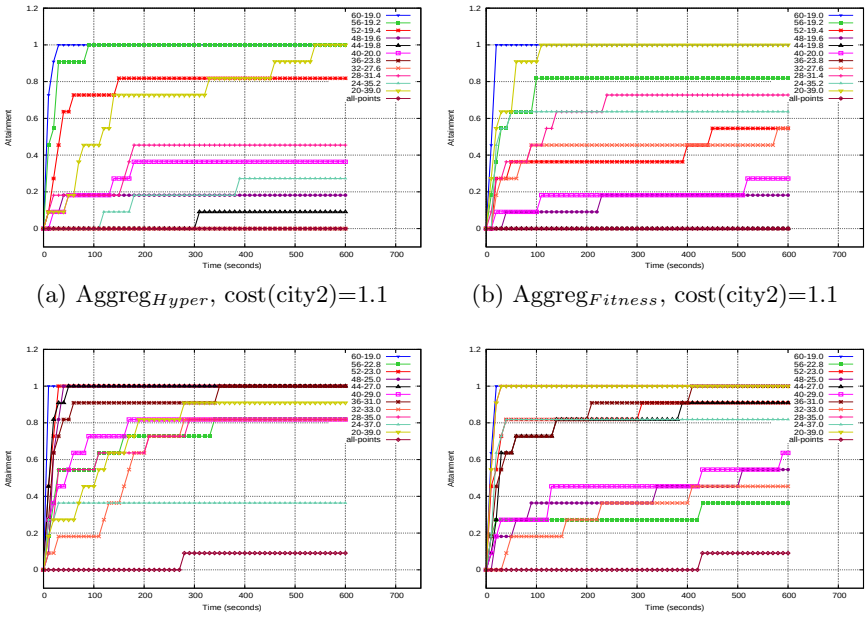
This paper has addressed several issues related to parameter tuning for aggregated approaches to multi-objective optimization. For the specific case study in AI temporal planning presented here, some conclusions can be drawn. First, the



(a) Aggreg$_{Hyper}$ on MultiZeno6 (b) Aggreg$_{Fitness}$ on MultiZeno6

(c) Aggreg$_{Hyper}$ on MultiZeno9 (d) Aggreg$_{Fitness}$ on MultiZeno9

Fig. 5: Pareto fronts on MultiZeno instances.



(a) Aggreg$_{Hyper}$, cost(city2)=1.1 (b) Aggreg$_{Fitness}$, cost(city2)=1.1

(c) Aggreg$_{Hyper}$, cost(city2)=2.9 (d) Aggreg$_{Fitness}$, cost(city2)=2.9

Fig. 6: Hitting plots for different Pareto fronts for MultiZeno6. See Section 2.2 and compare with Figure 4-(a) and (b).

parameter tuning of each single-objective run should be made using the hypervolume (or maybe some other multi-objective indicator) as a quality measure for parameter configurations, rather than the usual fitness of the target algorithm.

Second, the Aggreg_{Hyper} approach seems to obtain better results than the multi-objective DAE$_{\text{YAHSP}}$ presented in [15], in terms of hypervolume, as well as in terms of hitting of the points of the Pareto front. However, such comparison must take into account that one run of the aggregated approach requires eight times the CPU time of one single run: such fair comparison is the topic of ongoing work.

Finally, several specificities of the case study in AI planning make it very hazardous to generalize the results to other problems and algorithms without further investigations: DAE$_{\text{YAHSP}}$ is a hierarchical algorithm, that uses an embedded single objective planner that can only take care of one objective, while the evolutionary part handles the global behavior of the population; and the MULTIZENO instances used here have linear, or quasi-linear Pareto front; ongoing work is concerned with studying other domains along the same lines.

In any case, several issues have been raised by these results, and will be the subject of further work. At the moment, only instance-based parameter tuning was performed – and the preliminary results on the other instances with different Pareto front shapes (see Figure 6) suggest that the best parameter setting is highly instance-dependent (as demonstrated in a similar AI planning context in [28]). But do the conclusions drawn above still apply in the case of class-driven parameter tuning? Another issue that was not discussed here is that of the delicate choice of the values for α. Their proper choice is highly dependent on the scales of the different objectives. Probably some adaptive technique, as proposed by [13], would be a better choice.

References

1. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In A. Lodi et al., ed.: Proc. CPAIOR-10, LNCS 6140, Springer Verlag (2010) 186–202
2. Eiben, A., Michalewicz, Z., Schoenauer, M., Smith, J.: Parameter Control in Evolutionary Algorithms. In F.G.Lobo et al., ed.: Parameter Setting in Evolutionary Algorithms. Volume 54 of Studies Comp. Intel. Springer Verlag (2007) 19–46
3. Zhi, Y., de Oca, M.A.M., Stützle, T., Birattari, M.: Modern continuous optimization algorithms for tuning real and integer algorithm parameters. In M. Dorigo et al., ed.: Proc. ANTS'7, LNCS 6234, Springer Verlag (2010) 203–214
4. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: Automated algorithm tuning using F-Races: Recent developments. In M. Caserta et al., ed.: Proc. MIC'09. University of Hamburg (2009)
5. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: Automatic configuration of state-of-the-art multi-objective optimizers using the TP+PLS framework. In N. Krasnogor and P.-L. Lanzi, ed.: Proc. 13th ACM-GECCO. (2011) 2019–2026
6. Bartz-Beielstein, T., Lasarczyk, C., Preuss, M.: Sequential parameter optimization. In B. McKay et al., ed.: Proc. CEC'05, IEEE (2005) 773–780

7. Nannen, V., Eiben, A.: Relevance estimation and value calibration of evolutionary algorithm parameters. In M. Veloso et al., ed.: Proc. IJCAI'07. (2007) 975–980
8. Hutter, F., Hoos, H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. J. Artif. Intel. Research **36**(1) (2009) 267–306
9. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C., da Fonseca, V.: Performance assessment of multiobjective optimizers: an analysis and review. Evolutionary Computation, IEEE Transactions on **7**(2) (april 2003) 117 – 132
10. Ghallab, M., Nau, D., Traverso, P.: Automated Planning, Theory and Practice. Morgan Kaufmann (2004)
11. 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. 20^{th} ICAPS, AAAI Press (2010) 18–25
12. Miettinen, K.: Nonlinear multiobjective optimization. Volume 12. Springer (1999)
13. Jin, Y., Okabe, T., Sendhoff, B.: Adapting weighted aggregation for multiobjective evolution strategies. In Zitzler, E., Deb, K., Thiele, L., Coello, C.A.C., Corne, D., eds.: Proc. EMO 2001, LNCS 1993, Springer Verlag (2001) 96–110
14. Schoenauer, M., Savéant, P., Vidal, V.: Divide-and-Evolve: a New Memetic Scheme for Domain-Independent Temporal Planning. In Gottlieb, J., Raidl, G., eds.: Proc. 6^{th} EvoCOP, LNCS 3906, Springer (2006) 247–260
15. Khouadjia, M.R., Schoenauer, M., Vidal, V., Dréo, J., Savéant, P.: Multi-objective AI planning: Evaluating DAEYAHSP on a tunable benchmark. In Purshouse, R.C., Fleming, P.J., Fonseca, C.M., eds.: Proc. EMO'2013. (2013) To appear.
16. Do, M., Kambhampati, S.: SAPA: A Multi-Objective Metric Temporal Planner. J. Artif. Intell. Res. (JAIR) **20** (2003) 155–194
17. Refanidis, I., Vlahavas, I.: Multiobjective Heuristic State-Space Planning. Artificial Intelligence **145**(1) (2003) 1–32
18. Gerevini, A., Saetti, A., Serina, I.: An Approach to Efficient Planning with Numerical Fluents and Multi-Criteria Plan Quality. Artificial Intelligence **172(8-9)** (2008) 899–944
19. Gerevini, A., Long, D.: Preferences and Soft Constraints in PDDL3. In: ICAPS Workshop on Planning with Preferences and Soft Constraints. (2006) 46–53
20. Chen, Y., Wah, B., Hsu, C.: Temporal Planning using Subgoal Partitioning and Resolution in SGPlan. J. of Artificial Intelligence Research **26**(1) (2006) 323–369
21. Edelkamp, S., Kissmann, P.: Optimal Symbolic Planning with Action Costs and Preferences. In: Proc. 21^{st} IJCAI. (2009) 1690–1695
22. Fikes, R., Nilsson, N.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence **1** (1971) 27–120
23. Bibaï, J., Savéant, P., Schoenauer, M., Vidal, V.: On the Benefit of Sub-Optimality within the Divide-and-Evolve Scheme. In Cowling, P., Merz, P., eds.: Proc. 10^{th} EvoCOP, LNCS 6022, Springer Verlag (2010) 23–34
24. Haslum, P., Geffner, H.: Admissible Heuristics for Optimal Planning. In: Proc. AIPS-2000. (2000) 70–82
25. Vidal, V.: A Lookahead Strategy for Heuristic Search Planning. In: Proceedings of the 14^{th} ICAPS, AAAI Press (2004) 150–159
26. Lourenço, H., Martin, O., Stützle, T.: Iterated Local Search. Handbook of metaheuristics (2003) 320–353
27. Zitzler, E., Künzli, S.: Indicator-Based Selection in Multiobjective Search. In Xin Yao et al., ed.: Proc. PPSN VIII, LNCS 3242, Springer Verlag (2004) 832–842
28. Bibaï, J., Savéant, P., Schoenauer, M., Vidal, V.: On the Generality of Parameter Tuning in Evolutionary Planning. In: Proc 12^{th} GECCO, ACM (2010) 241–248