ADR-EDA: A Reliability-Aware Controller for Dynamic Multiobjective Estimation of Distribution Algorithms

Jesús Samuel García Carballo[1]*, Concha Bielza Lozoya[1], and Pedro Larrañaga Múgica[1]

Departamento de Inteligencia Artificial, Escuela Técnica Superior de Ingenieros Informáticos, Universidad Politécnica de Madrid, 28660 Boadilla del Monte, Madrid, Spain
jesussamuel.garciacarballo@alumnos.upm.es
mcbielza@fi.upm.es, pedro.larranaga@fi.upm.es

Abstract. Dynamic estimation of distribution algorithms (EDAs) reuse probabilistic search information learned under previous objective, constraint, or operational settings. This reuse can accelerate adaptation when consecutive environments remain related, but outdated search state may misguide the search after substantial changes. We propose Adaptive Dynamic Response Estimation of Distribution Algorithm (ADR-EDA), a controller-oriented dynamic EDA that decides how much of the learned model, elite set, archive, and population should still guide search after a detected change. Rather than applying a fixed repair operator, ADR-EDA estimates change severity and assembles the next working population from preserved, predicted, exploratory, and locally refined candidates; a quality gate rejects detrimental responses before the probabilistic model is updated. ADR-EDA is evaluated on a 14-problem dynamic benchmark against eight multiobjective baselines and on an electric car-sharing study with nominal, long-horizon, and demand-stress rolling-horizon validations. Results show robust global tracking on the benchmark and a useful deployment-oriented behavior in the application, where ADR-EDA improves operating margin and reduces deadhead while accepting far fewer policy refreshes.

Keywords: Dynamic multiobjective optimization · Estimation of distribution algorithms · Reliability-aware control · Adaptive dynamic response · Electric car-sharing

1 Introduction

In dynamic multiobjective optimization problems (DMOPs) [1], candidate solutions must remain effective while objective functions, constraints, or operating conditions change over time.

* Corresponding author.

DMOPs are difficult to solve because the search information collected under one regime may become only partially reusable after a change. As a consequence, the Pareto-optimal set may move in the decision space, the Pareto front may evolve in the objective space, and feasible regions may shift when constraints change. This creates a tension between recovering high-quality solutions after a change and tracking the moving Pareto-optimal set and front over time: methods that react too weakly may fail to recover good solutions after abrupt changes, whereas methods that react too aggressively may discard useful structure and lose continuity when tracking related regimes.

This difficulty is especially acute for estimation of distribution algorithms (EDAs) [5]. Because they learn probabilistic models from promising solutions, EDAs can exploit structure in the search space more explicitly than the rest of evolutionary algorithms based mainly on mutation and recombination. In dynamic environments, however, the probabilistic model and the search structures accumulated during optimization, such as the current population, elite set, and archive, become part of the optimizer's state: if they remain informative, they can accelerate adaptation after change. If they are preserved too rigidly, however, they can bias the search toward an outdated region of the search space. For dynamic EDAs [7], fitting a model is only part of the problem: the EDA backbone estimates and samples a distribution, whereas a separate response mechanism is needed to decide how the carried-over search state should be reused after the change. This includes deciding how much of the previous population and probabilistic model should be preserved, when promising solutions should be shifted according to the observed movement of the front, how much diversity should be reintroduced, and when local refinement around the current front is worth the effort.

Existing dynamic multiobjective evolutionary algorithms address this challenge with several complementary mechanisms. Diversity injection restores exploration after change by introducing or maintaining search material [4]. Memory reuse preserves information from previous environments for later use [11]. Transfer methods exploit similarity between consecutive environments to reuse information in the current search [6]. Prediction methods estimate the movement of the Pareto set or front and use that estimate to relocate existing solutions or generate new candidates [14]. Adaptive controllers select or weight response mechanisms according to detected dynamic conditions, rather than applying the same reaction after every change [3,10,13]. However, these controllers are not designed for EDAs, where post-change adaptation must also decide how the learned probabilistic model, population, elite set, and archive should influence the next model update.

This motivates Adaptive Dynamic Response Estimation of Distribution Algorithm (ADR-EDA), whose ADR layer assesses the reliability of the probabilistic model, population, elite set, and archive inherited from the previous problem setting. ADR-EDA then sets the response level, builds the next working population from preserved, predicted, exploratory, and locally refined candidates, and applies a quality gate before updating the probabilistic model. We evaluate it

on a 14-problem dynamic benchmark and a rolling-horizon electric car-sharing simulator.

2 Related Work

EDAs replace explicit recombination with probabilistic models learned from promising regions of the decision space [5]. A standard EDA cycle selects promising solutions, learns a probabilistic model, samples new candidates, and updates the population. EDA variants differ in the structure of this model and in the domain representation: univariate EDAs assume independent variables, bivariate and multivariate EDAs model dependencies between variables, and both combinatorial and continuous optimization variants have been developed.

Classical mechanisms for handling environmental changes in DMOPs address different ways of rebuilding or redirecting the population after a change. Several baselines follow the NSGA-II evolutionary scheme combining multiobjective population ranking with evolutionary operators that generate new candidate solutions. DNSGA-II-B injects mutated individuals after change [4], MNSGA-II augments NSGA-II with external memory [11], and Tr-NSGA-II combines NSGA-II with transfer learning across environments [6]. SGEA uses a hybrid steady-state and generational evolutionary search and reuses outdated but well-distributed solutions [9]. PFPA is a prediction-oriented evolutionary method that fits polynomial models to past Pareto-set or Pareto-front movements and uses them to guide population movement [14]. These mechanisms address different types of environmental change: diversity-oriented responses are useful after abrupt environmental changes, whereas memory, transfer, and prediction are generally more useful when successive environments remain related.

Recent work therefore shifts attention from individual mechanisms to adaptive controllers that decide how to combine them. TDA-DMOEA selects response mechanisms according to dynamic conditions [3], whereas AB-DMOEA reweights adaptive responses according to the detected change [10]. ADR-DMOEA is the closest conceptual neighbor to the present work because it allocates response effort over MOEA/D subpopulations after changes [13]. The key difference is not the presence of response mechanisms, but the object through which adaptation is applied: ADR-DMOEA controls the distribution of responses over MOEA/D subpopulations, whereas the EDA case considered here builds a post-change working population that is subsequently used to update the probabilistic model, elite set, and archive. This distinction matters because, in EDAs, the population accepted after a change determines which carried-over search information remains active in the next model update.

This leaves a specific gap: the reliability-aware integration of dynamic responses within an EDA-based architecture. Existing methods provide useful post-change mechanisms, but they do not explicitly address how much carried-over search information should still guide the assembly of the next working population before the probabilistic model is updated.

3 Method

ADR-EDA is organized as a persistent EDA backbone plus an ADR controller. The backbone learns a probabilistic model, samples candidate solutions, and carries population-level search state across environments. The controller is invoked when this carried-over state may no longer be reliable. Its role is not to edit the SPBN parameters directly, but to decide which candidates should enter the next working population. The SPBN is then re-estimated from the accepted working population.

3.1 Persistent EDA Backbone

The backbone of ADR-EDA uses a very flexible probabilistic model, the semi-parametric Bayesian network (SPBN), i.e., it is a SPEDA [12]. In this SPBN, local conditional distributions may be represented by either Gaussian densities or kernel-density estimators. The EDA carries the search state

$$S_t = (P_t, E_t, A_t, \boldsymbol{\theta}_t),$$

where P_t, E_t, A_t, and $\boldsymbol{\theta}_t$ denote the current population, elite set, bounded archive, and probabilistic-model parameters, respectively. In an ordinary EDA step, promising solutions update the elite set, archive, and model parameters; the updated model then samples the next population. After an environmental change, the same state may be partly useful and partly obsolete. ADR-EDA therefore intervenes before the model update: it constructs the working population from which the next model will be learned.

The SPBN sampling distribution factorizes according to the network structure. For a decision vector $\mathbf{X} = (X_1, \ldots, X_n)$, where n is the number of variables, new candidates are sampled from $p_t(\mathbf{X} \mid \boldsymbol{\theta}_t)$:

$$p_t(\mathbf{X} \mid \boldsymbol{\theta}_t) = \prod_{i=1}^{n} p_t(X_i \mid \mathbf{X}_{\mathrm{Pa}(X_i)}; \boldsymbol{\theta}_{i,t}),$$

where $\mathrm{Pa}(X_i)$ denotes the parent variables of X_i in the network and $\boldsymbol{\theta}_{i,t}$ contains the local conditional parameters. Thus, $\boldsymbol{\theta}_t$ stores the dependencies and local distributions learned before the current change. If the environment changes, this learned model may become unreliable, which motivates the ADR control layer below.

3.2 Reliability-Aware ADR Control

ADR-EDA separates *change evidence* from *response control*, as summarized in Fig. 1. Evidence assesses whether the inherited state remains compatible with the new environment. Response control does not overwrite the SPBN parameters directly. Instead, it decides which solutions should form the next working

population; only after the quality gate accepts that population is the SPBN re-estimated from it. When activation conditions are not met, the ordinary EDA update continues. Otherwise, ADR proceeds in four steps: it filters the evidence, maps it to a response level and context, allocates a replacement budget across four candidate-generation blocks, and assembles a tentative working population.

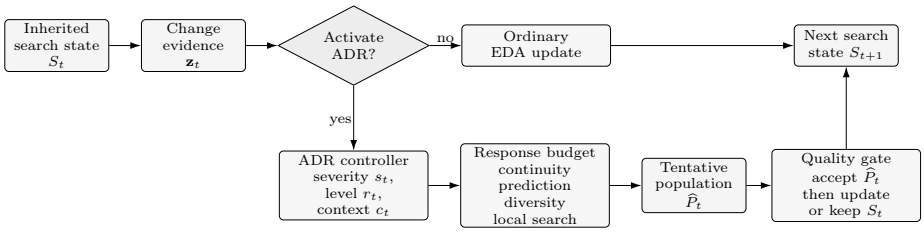


Fig. 1. ADR-EDA workflow. When ADR is active, the controller allocates a response budget, builds a tentative working population, and updates the SPBN only if the quality gate accepts it; otherwise the inherited state is kept. Here S_t denotes the EDA search state at time t.

Evidence construction produces a vector $\mathbf{z}_t$ whose components summarize how strongly the carried-over state may no longer be compatible with the current problem setting. In the benchmark profile, $\mathbf{z}_t$ is built from a bounded probe set $Q_t = \{(\mathbf{X}_q, \mathbf{F}_q^{\text{old}})\}_{q=1}^{m_q}$ of representative decision vectors kept only for change checking. Their stored objective values $\mathbf{F}_q^{\text{old}}$ are compared with $\mathbf{F}_q^{\text{new}}$ after reevaluating the same vectors, so the evidence measures the same decisions before and after the change.

From the old and new objective-value matrices, ADR-EDA computes two scalar drift scores that summarize complementary changes in the probe responses. The mean-drift score d_t^{μ} measures displacement of the objective-value center and is implemented as a Hotelling T^2 statistic. The covariance-drift score d_t^{Σ} measures changes in the objective-value dispersion and dependence structure and is implemented as a Stein divergence between regularized covariance matrices. Each score is then smoothed and normalized by its detection threshold:

$$\bar{d}_t^k = \lambda_{k,t} d_t^k + (1 - \lambda_{k,t}) \bar{d}_{t-1}^k, \qquad k \in \{\mu, \Sigma\},$$
$$\mathbf{z}_t \equiv \mathbf{z}_t^{\text{bench}} = \left(\bar{d}_t^{\mu} / \tau_t^{\mu}, \bar{d}_t^{\Sigma} / \tau_t^{\Sigma} \right).$$

Here, $\lambda_{k,t} \in [0, 1]$ controls the trade-off between the current drift score d_t^k and the previous smoothed value $\bar{d}_{t-1}^k$: larger values react faster, whereas smaller values give more inertia to the past. This recursive average reduces the effect of isolated one-step fluctuations while preserving persistent changes. The thresholds τ_t^{μ} and τ_t^{Σ} make the two drift scores comparable: τ_t^{μ} is derived from the Hotelling threshold, whereas τ_t^{Σ} is estimated from recent covariance-drift scores. After filtering, three safeguards reduce unstable triggering: hysteresis uses separate activation and deactivation thresholds, consecutive-alert logic requires persistence across checks, and cooldown prevents repeated strong responses over very short intervals. In domain-specific settings, $\mathbf{z}_t$ may instead contain problem-aware de-

scriptors, but the same ADR control steps are applied to the resulting evidence vector.

The filtered evidence is first summarized into a scalar severity score s_t. In the benchmark profile, s_t combines the strongest filtered drift ratio with detector-history information, such as the number of checks since the previous ADR response. A fixed threshold rule maps this score to a response level $r_t \in \{\text{mild, medium, severe}\}$: mild responses mostly preserve inherited state, medium responses mix reuse and correction, and severe responses emphasize exploratory repair.

The response level r_t controls how much of the population may be changed. A second label, the response context c_t, controls how this change is distributed across mechanisms. ADR-EDA classifies c_t as stable, uncertain, or abrupt using detector-history summaries: alert regularity, alert persistence, and whether covariance drift dominates mean drift. Stable contexts favor continuity and prediction, uncertain contexts use a balanced allocation, and abrupt contexts give more weight to diversity. The fixed profile $W(r_t, c_t)$ then assigns one weight to each candidate-generation block:

$$\mathbf{w}_t = W(r_t, c_t) = (w_t^c, w_t^p, w_t^d, w_t^\ell), \qquad w_t^c + w_t^p + w_t^d + w_t^\ell = 1.$$

Let $N = |P_t|$. The response level r_t determines a replacement budget $B_t \leq N$, i.e., the maximum number of current candidates that ADR may replace. The remaining $N - B_t$ candidates stay in the population. The weights $\mathbf{w}_t$ split this budget among the four candidate-generation blocks. For block $j \in \{c, p, d, \ell\}$, a provisional count is

$$\tilde{n}_t^j = \lfloor w_t^j B_t \rfloor.$$

The remaining slots are assigned to the largest fractional remainders of $w_t^j B_t$, so that the final counts satisfy $n_t^c + n_t^p + n_t^d + n_t^\ell = B_t$. The four candidate-generation blocks then build a response pool $R_t = P_t^c \cup P_t^p \cup P_t^d \cup P_t^\ell$. The subsets correspond to continuity, prediction, diversity injection, and local search. P_t^c reuses reliable candidate solutions from P_t, E_t, or A_t, optionally with small jitter. P_t^p shifts representative elite/front solutions according to their recent displacement. P_t^d broadens exploration by sampling away from the elite neighborhood or by applying stronger random perturbations. P_t^ℓ applies bounded neighborhood moves around current non-dominated solutions. The response pool replaces B_t selected candidates of P_t; the remaining $N - B_t$ candidates are carried over unchanged, yielding a full tentative working population $\widehat{P}_t$ of size N.

The quality gate is the final safeguard. The tentative population is evaluated to obtain a front, but the resulting update is not accepted into the stored EDA state until this gate is passed. The tentative front $\widetilde{F}_{t+1}$ is compared with the pre-response front F_t. The fixed profile checks four front-quality indicators: best summed objective value, mean summed objective value, componentwise ideal sum, and front size. The response is rejected only when the componentwise ideal sum worsens beyond the fixed tolerance and either the best and mean summed objective values also worsen or the tentative front becomes too small. If rejected,

all tentative updates are discarded, $\widehat{P}_t$ is not used for SPBN learning, and the previous state S_t is kept.

4 Experimental Setup

4.1 Benchmark Protocol

The benchmark validation uses 14 problems belonging to two families. The first family contains eight canonical DMOP constructions [1] covering step and sinusoidal shifts, shape and shock changes, constraints, and a three-objective DTLZ-style case. The second family contains six instances from the Dynamic Functions benchmark suite, using the frequent explicit-change setting ($T = 10$) [8]. Thus, the canonical family tests heterogeneous dynamic regimes, whereas the Dynamic Functions suite stresses repeated post-change response under a widely used dynamic-function protocol.

ADR-EDA is compared against DNSGA-II-B [4], MNSGA-II [11], SGEA [9], Tr-NSGA-II [6], PFPA [14], TDA-DMOEA [3], AB-DMOEA [10], and ADR-DMOEA [13] under the same budget: 15 seeds, 60 environmental time steps per run, population size of 40, eight decision variables, and one generation per step. The eight-variable setting is a compact low-budget profile intended to compare dynamic response behavior under the same online evaluation budget, not a scalability claim; Sect. 5 therefore also reports a focused 20-variable sensitivity probe. SGEA and AB-DMOEA were cross-checked against released author code, whereas the remaining baselines were reimplemented in our experimental framework following the algorithmic descriptions in the corresponding publications and run under the same budget. The primary indicators are mean inverted generational distance (MIGD), the average distance to the time-varying reference front, and mean hypervolume (MHV), the dominated objective-space volume over time. On explicit-change problems, post-change IGD/HV measure immediate recovery after changes, and recovery time measures how quickly pre-change quality is recovered, following standard dynamic benchmark practice [1].

The evaluated ADR-EDA configuration is fixed for this 14-problem benchmark and uses the SPBN-based profile described in Sect. 3. ADR-DMOEA is included as the closest reliability-aware response competitor through an independent paper-core implementation; no author-provided implementation was available to us, so its row is marked accordingly and should not be read as an official-code execution. Statistical tests use paired Wilcoxon signed-rank comparisons over matched problem-seed runs. The release package contains the per-run CSVs, configuration files, simulator source, and table-regeneration scripts.

4.2 Car-Sharing Setting

The car-sharing study is the main applied validation of the paper. It tests whether benchmark-side ADR transfers to a rolling-horizon free-floating electric car-sharing setting. The problem is an online fleet-management task over

a Madrid-inspired service area: the operator must decide how a fleet of electric vehicles should be charged, repositioned, and reserved while chargers and relocation staff are limited. The operational goal is not only to serve as many customers as possible, but also to improve the episodic economic margin and avoid policies that degrade battery health. In this domain, refreshing a policy too often can be harmful: each accepted update may induce additional relocation, deadhead travel, staff actions, and charging operations. The setting is motivated by optimization-simulation work on electric car-sharing relocation, personnel operations, and reservations [2]. Our simulator is not a replication of those frameworks, but follows the same type of operational coupling between fleet availability, charging, staff resources, and reservation pressure.

We introduce the simulator used in this study. At each decision epoch, it provides the state used by the policy-update method: vehicle state of charge and health (SOC/SOH), charger occupancy, operational-resource availability, active reservations, pending requests, recent operating indicators, and temporal context. Reservations are advance bookings that must be protected before their pickup time, whereas requests are new trip demands arriving during the simulated horizon. The operating indicators summarize recent service and rejection rates, relocation and deadhead effort, charger use, and the revenue and cost terms used in the margin objective; temporal context identifies the current decision time.

The car-sharing environment uses 220 vehicles, 22 chargers, and 26 operational resources. Every 5 minutes, the system observes the current operational state and may update the high-level policy used over a 60-minute rolling horizon. The applied validation includes 60-minute and 8-hour online simulations, with nominal and demand-stress blocks; the stress block uses demand multipliers 1.00, 1.15, and 1.30. Three policy-update strategies are compared: `full_static`, which refreshes the policy at every decision epoch; ADR-EDA, which refreshes selectively through the ADR control layer; and `fast_event`, a selective event-driven baseline that uses the same problem-aware descriptors with fixed trigger thresholds and fixed minimum and maximum waiting times between policy refreshes, but without ADR-EDA's block allocation and quality gate. Whenever a strategy evaluates a policy update, it uses the same online optimization budget: two EDA update cycles per decision epoch, each evaluating 24 candidate policies.

Candidate solutions represent high-level operating policies, not explicit vehicle routes. In the applied instance, $\mathbf{x} = (x_1, \ldots, x_{20}) \in \mathbb{R}^{20}$: the first six components encode charging trigger SOC, charging target SOC, relocation intensity, minimum SOC for relocation, demand weight in repositioning, and reservation pre-assignment intensity. The remaining 14 components are multiplicative zone-level demand biases that increase or decrease the attractiveness of sending idle vehicles to each zone. Thus, ADR responses modify policy vectors, not the number of vehicles, chargers, or operational resources.

Each candidate policy is decoded according to this representation and evaluated over the rolling-horizon window that starts from the current operating

state. The simulator returns a tri-objective value $\mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x}))$, where all objectives should be maximized: service quality, normalized operating margin, and fleet health. The objectives are defined as

$$
\begin{aligned}
f_1(\mathbf{x}) &= s(\mathbf{x})\big(w_{\mathrm{sr}} + w_{\mathrm{walk}} a(\mathbf{x}) + w_{\mathrm{res}} r(\mathbf{x})\big), \\
f_2(\mathbf{x}) &= \tfrac{1}{2}\left(\tanh(\mathit{\Pi}(\mathbf{x})/\alpha) + 1\right), \quad \text{with} \\
\mathit{\Pi}(\mathbf{x}) &= \mathrm{Rev} + \mathrm{Cred} - (\mathrm{Cost}^{\mathrm{rel}} + \mathrm{Cost}^{\mathrm{ene}} + \mathrm{Cost}^{\mathrm{ops}} + \mathrm{Cost}^{\mathrm{dead}} + \mathrm{Cost}^{\mathrm{pen}}), \\
f_3(\mathbf{x}) &= w_{\mathrm{use}} H_{\mathrm{use}}(\mathbf{x}) + w_{\mathrm{loss}} H_{\mathrm{loss}}(\mathbf{x}) + w_{\mathrm{tail}} H_{\mathrm{tail}}(\mathbf{x}).
\end{aligned}
$$

The service objective f_1 rewards policies that serve more requests, reduce walking inconvenience, and protect reservations. Here, $s(\mathbf{x})$ is the served-request fraction, $a(\mathbf{x})$ decreases when users must walk farther to reach a vehicle, and $r(\mathbf{x})$ penalizes advance bookings that cannot be served at pickup time. If no request becomes due in the simulated window, f_1 is set to 1. The margin objective f_2 maps the episodic operating margin $\mathit{\Pi}(\mathbf{x})$, accumulated over the same window, to $[0, 1]$; α controls the scale of this normalization. In $\mathit{\Pi}(\mathbf{x})$, Rev denotes trip revenue, whereas Cred represents an internal credit for operational actions that relocate vehicles to chargers. The cost terms correspond to relocation distance, charging energy, staff actions, non-revenue deadhead travel, and penalties for rejected requests. Finally, f_3 evaluates the effect of the policy on fleet condition at the end of the window. The score H_{use} discourages intensive battery throughput, while H_{loss} and H_{tail} account for SOH loss during the window and low final SOC levels. Weights in f_1 and f_3 are normalized.

For ADR control, the benchmark evidence vector $\mathbf{z}_t$ is replaced by a problem-aware vector $\mathbf{z}_t^{cs} \in \mathbb{R}^7$. Its seven components are demand pressure, reservation pressure, low-SOC share, charger pressure, spatial imbalance, coverage gap, and recent rejection rate. These signals feed the same ADR decision layer defined in Sect. 3. Benchmark and application therefore differ in evidence semantics, not in the evidence-to-response scheme.

5 Results

5.1 Benchmark Results

Table 1 evaluates the fixed ADR-EDA profile under the unified benchmark protocol. It reports means and standard deviations over 210 runs for the global metrics (14 problems $\times$ 15 seeds); the post-change columns use the explicit-change subset of 120 runs. ADR-EDA obtains the lowest MIGD and the highest mean hypervolume.

The closest competitors are SGEA in hypervolume and Tr-NSGA-II in MIGD and post-change indicators. Table 2 shows that ADR-EDA remains best in both benchmark families for global MIGD and hypervolume averages: SGEA is closest on the canonical suite, whereas Tr-NSGA-II is closest on DF-T10. Paired Wilcoxon tests support the aggregate comparison ($p < 0.05$ against every baseline in MIGD and MHV). Against the closest response-control comparator, ADR-

Table 1. Overall benchmark results (mean ± standard deviation) on the 14-problem suite. Post-change metrics use the explicit-change subset (120 runs). †Independent paper-core implementation.

Algorithm	MIGD ↓	MHV ↑	postIGD ↓	postHV ↑	Time (s) ↓
ADR-EDA	**0.3748** ± 0.6824	**5.3607** ± 3.5307	0.2780 ± 0.5266	4.5904 ± 4.4231	204.3 ± 116.7
ADR-DMOEA†	1.0066 ± 1.7754	4.1112 ± 2.8282	1.3915 ± 2.6384	3.5148 ± 3.5976	**1.7** ± 2.6
DNSGA-II-B	1.1113 ± 2.9730	4.9528 ± 3.4255	1.5801 ± 3.9008	4.3584 ± 4.4953	23.8 ± 4.6
MNSGA-II	1.1731 ± 2.7596	4.6152 ± 3.3129	1.4922 ± 3.5353	4.3665 ± 4.5051	25.6 ± 5.3
SGEA	0.6829 ± 1.6395	5.1786 ± 3.5289	0.9610 ± 2.1825	4.4672 ± 4.5471	7.6 ± 5.5
Tr-NSGA-II	0.6112 ± 0.8874	4.7203 ± 3.1768	**0.2192** ± 0.3410	**4.5953** ± 4.2818	28.6 ± 8.2
PFPA	1.2935 ± 2.8458	4.1596 ± 3.0690	1.3091 ± 3.0276	4.1196 ± 4.2258	4.7 ± 2.3
TDA-DMOEA	1.2624 ± 3.0784	4.3670 ± 3.0725	1.9564 ± 4.6893	3.8424 ± 4.0035	2.1 ± 2.2
AB-DMOEA	1.1306 ± 2.9748	4.7017 ± 3.2665	1.6677 ± 4.3032	4.3137 ± 4.3895	3.6 ± 3.0

Table 2. Family-level means on the canonical suite and DF-T10.

Algorithm	Canonical		DF-T10	
	MIGD ↓	MHV ↑	MIGD ↓	MHV ↑
ADR-EDA	**0.1936**	**7.3321**	**0.6164**	**2.7323**
ADR-DMOEA†	0.5954	5.5020	1.5550	2.2570
DNSGA-II-B	0.2854	6.8336	2.2126	2.4450
MNSGA-II	0.4575	6.2345	2.1273	2.4561
SGEA	0.1973	7.1017	1.3305	2.6145
Tr-NSGA-II	0.4593	6.2326	0.8138	2.7039
PFPA	0.6621	5.3387	2.1353	2.5875
TDA-DMOEA	0.5023	5.8423	2.2758	2.3999
AB-DMOEA	0.4045	6.2671	2.0987	2.6145

DMOEA, ADR-EDA wins 204/210 MIGD, 200/210 MHV, 120/120 postIGD, and 111/120 postHV pairs ($p < 10^{-20}$), while ADR-DMOEA is much faster.

The short-term adaptation picture is more nuanced. Against Tr-NSGA-II on the explicit-change subset, ADR-EDA loses significantly in post-change IGD, while post-change HV and recovery time remain statistically indistinguishable (Table 3).

Table 3. Post-change means, runtime, and complementary Wilcoxon signed-rank tests against Tr-NSGA-II. Post-change metrics use the explicit-change subset (120 paired runs); runtime uses all 210 paired runs.

Metric	ADR-EDA	Tr-NSGA-II	Wilcoxon p
Post-change IGD ↓	0.2780	0.2192	8.85e−04
Post-change HV ↑	4.5904	4.5953	0.1450
Recovery time	2.8028	2.9228	0.9570
Time (s) ↓	204.3	28.6	$< 10^{-16}$

Overall, ADR-EDA leads on global tracking, whereas Tr-NSGA-II remains the main immediate-response competitor: its post-change IGD advantage is significant, its small post-change HV advantage is not, and it is significantly faster. ADR-EDA's overhead mainly comes from SPBN maintenance, evidence probes,

response-pool generation, local front refinement, and the quality gate. A five-seed variant screen found a faster ADR-EDA profile with similar global quality, but no tested variant improved MIGD and recovery time simultaneously. The quality gains should therefore be read as a robustness-versus-cost trade-off, not a free improvement; a full component ablation remains future work. A focused 20-variable probe on three representative problems supports the same bounded reading: ADR-EDA obtained the best aggregate MIGD (3.626), MHV (2.936), postIGD (1.940), and postHV (3.996), but it is a dimensional-sensitivity probe rather than a full large-scale benchmark.

5.2 Car-Sharing Results

The applied study tests whether ADR-EDA improves operating outcomes in the electric car-sharing simulator by refreshing the deployed policy only when the evidence justifies it. Table 4 compares ADR-EDA with the always-reoptimize controller across the validation blocks defined in Sect. 4. In both applied tables, service deltas are percentage points, refreshes are mean accepted policy updates, and margin intervals are 95% CI half-widths.

Table 4. Applied validation against the always-reoptimize `full_static` controller. Delta columns report ADR-EDA minus `full_static`.

Study	n	Δ margin (EUR)	Δ service (pp)	Δ deadhead (km)	Refreshes full/ADR	Wins margin/deadhead
Nominal 60 min	10	$+133.23 \pm 60.05$	0.00	-92.24	12.00/1.30	8/10, 8/10
Nominal 8 h	8	$+453.06 \pm 401.51$	-0.40	-140.74	96.00/5.75	7/8, 8/8
Stress 60 min	24	$+144.77 \pm 34.36$	0.00	-104.05	12.00/1.21	23/24, 23/24

Across all three applied blocks, ADR-EDA improves mean episodic margin and reduces deadhead while accepting far fewer policy updates. Service remains identical in the 60-minute blocks and slightly lower in the 8-hour block, so the correct reading is not uniform dominance but a better operating trade-off. Exact paired signed-rank tests confirm these margin and deadhead improvements against `full_static` in all blocks ($p \leq 0.016$, two-sided).

Table 5 then compares ADR-EDA with `fast_event`, a selective event-driven controller, to separate the effect of refreshing less often from the quality of the accepted policy.

Table 5. Mechanism comparison against the selective `fast_event` baseline. Delta columns report ADR-EDA minus `fast_event`.

Study	n	Δ margin (EUR)	Δ service (pp)	Δ deadhead (km)	Refreshes fast/ADR	Wins margin/deadhead
Nominal 60 min	10	$+51.24 \pm 33.48$	0.00	-18.88	1.70/1.30	8/10, 6/10
Nominal 8 h	8	$+87.08 \pm 84.55$	+0.20	-25.33	8.88/5.75	6/8, 6/8
Stress 60 min	24	$+25.77 \pm 33.55$	0.00	-18.18	1.83/1.21	14/24, 12/24

The mechanism comparison shows that `fast_event` is a competitive selective control. ADR-EDA still gives the better aggregate trade-off, although the stress-block wins are not absolute; hence the advantage beyond selectivity is real but smaller than the advantage over always refreshing.

6 Conclusion

This paper presented ADR-EDA as a reliability-aware response controller for dynamic multiobjective EDAs. It assembles each post-change working population from preserved, predicted, exploratory, and locally refined candidates before updating the probabilistic model, thereby deciding how much inherited search state should remain active. On the 14-problem benchmark, ADR-EDA obtains the best global averages, while Tr-NSGA-II remains stronger on immediate post-change IGD and ADR-DMOEA exposes the quality-versus-runtime trade-off. In the car-sharing study, ADR-EDA improves margin, lowers deadhead, and reduces policy refreshes; the selective baseline shows that this is not merely a refresh-frequency effect. The limits are also clear: ADR-EDA is costly, Tr-NSGA-II remains stronger in post-change IGD, and the current ablation evidence is only a variant screen. Future work should improve immediate recovery, cheaper evaluation, systematic ablations, larger decision spaces, and broader car-sharing regimes.

Acknowledgments. This work was partially supported by the Comunidad de Madrid through the IDEA-CM-TFM predoctoral research contract at Universidad Politécnica de Madrid.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Azzouz, R., Bechikh, S., Said, L.B.: Dynamic multi-objective optimization using evolutionary algorithms: A survey. In: Recent Advances in Evolutionary Multi-objective Optimization, pp. 31–70. Springer (2017).
2. Boyaci, B., Zografos, K.G., Geroliminis, N.: An integrated optimization-simulation framework for vehicle and personnel relocations of electric carsharing systems with reservations. Transportation Research Part B: Methodological **95**, 214–237 (2017).
3. Cai, X., Wu, L., Zhao, T., Wu, D., Zhang, W., Chen, J.: Dynamic adaptive multi-objective optimization algorithm based on type detection. Information Sciences **654**, 119867 (2024).
4. Deb, K., Rao, N.U.B., Karthik, S.: Dynamic multi-objective optimization and decision-making using modified NSGA-II: A case study on hydro-thermal power scheduling. In: Evolutionary Multi-Criterion Optimization. Lecture Notes in Computer Science, vol. 4403, pp. 803–817. Springer (2007).
5. Hauschild, M., Pelikan, M.: An introduction and survey of estimation of distribution algorithms. Swarm and Evolutionary Computation **1**(3), 111–128 (2011).
6. Jiang, M., Huang, Z., Qiu, L., Huang, W., Yen, G.G.: Transfer learning-based dynamic multiobjective optimization algorithms. IEEE Transactions on Evolutionary Computation **22**(4), 501–514 (2018).
7. Jiang, M., Qiu, L., Huang, Z., Yen, G.G.: Dynamic multi-objective estimation of distribution algorithm based on domain adaptation and nonparametric estimation. Information Sciences **435**, 203–223 (2018).
8. Jiang, S., Kaiser, M., Yang, S., Kollias, S., Krasnogor, N.: A scalable test suite for continuous dynamic multiobjective optimization. IEEE Transactions on Cybernetics **50**(6), 2814–2826 (2020).
9. Jiang, S., Yang, S.: A steady-state and generational evolutionary algorithm for dynamic multiobjective optimization. IEEE Transactions on Evolutionary Computation **21**(1), 65–82 (2017).
10. Peng, H., Xiong, J., Pi, C., Zhou, X., Wu, Z.: A dynamic multi-objective optimization evolutionary algorithm with adaptive boosting. Swarm and Evolutionary Computation **89**, 101621 (2024).
11. Sahmoud, S., Topcuoglu, H.R.: A memory-based NSGA-II algorithm for dynamic multi-objective optimization problems. In: Applications of Evolutionary Computation. Lecture Notes in Computer Science, vol. 9597, pp. 296–310. Springer (2016).
12. Soloviev, V.P., Bielza, C., Larrañaga, P.: Semiparametric estimation of distribution algorithms for continuous optimization. IEEE Transactions on Evolutionary Computation **28**(4), 1069–1083 (2024).
13. Wang, Y., Zhou, P., Yang, S., Chai, T.: ADR-DMOEA: A dynamic multiobjective optimization evolutionary algorithm based on adaptive dynamic response strategy. IEEE Transactions on Cybernetics (2026). , early access article
14. Zhang, Q., He, X., Yang, S., Dong, Y., Song, H., Jiang, S.: Solving dynamic multi-objective problems using polynomial fitting-based prediction algorithm. Information Sciences **610**, 868–886 (2022).