Advancing Dynamic Evolutionary Optimization Using In-Memory Database Technology

Julia Jordan[1,2], Wei Cheng[3], and Bernd Scheuermann[1](✉)

[1] Hochschule Karlsruhe, University of Applied Sciences, Karlsruhe, Germany
bernd.scheuermann@hs-karlsruhe.de
[2] CAS Software AG, Karlsruhe, Germany
[3] SAP Innovation Center Network, Potsdam, Germany

Abstract. This paper reports on IMDEA (In-Memory database Dynamic Evolutionary Algorithm), an approach to dynamic evolutionary optimization exploiting in-memory database (IMDB) technology to expedite the search process subject to change events arising at runtime. The implemented system benefits from optimization knowledge persisted on an IMDB serving as associative memory to better guide the optimizer through changing environments. For this, specific strategies for knowledge processing, extraction and injection are developed and evaluated. Moreover, prediction methods are embedded and empirical studies outline to which extent these methods are able to anticipate forthcoming dynamic change events by evaluating historical records of previous changes and other optimization knowledge managed by the IMDB.

Keywords: Dynamic evolutionary algorithm · Associative memory · Prediction · In-memory databases

1 Introduction

For decades *Evolutionary Algorithms (EA)* [1] have been established heuristics to tackle NP-hard optimization problems which are inherent to countless industrial applications. Typically, the search for good solutions to such problems can consume up to several hours or even days. The hitherto best solution found, e.g. a production schedule, would then be used for planning and executing operations. In practice, however, several aspects like the objective function, the size of the problem instance or constraints may be subject to changes. In such *dynamic optimization* scenarios, it is essential that every relevant change of the optimization problem is taken into account. However, calculation time is commonly restricted and usually one cannot afford to restart optimization from scratch. Instead it is often advisable to exploit existing optimization knowledge from the running optimization to quickly react to and to recover from dynamic changes arriving.

This paper reports on the implementation and on the empirical evaluation of *IMDEA (In-Memory database Dynamic Evolutionary Algorithm)*, a dynamic

EA which interfaces with an in-memory database (IMDB) [2] and exploits its strengths to expedite the search process in dynamically changing environments. It is shown, how in-memory databases can be used as a large capacity knowledge store that embodies a persistent associative memory to the optimization algorithm. Such knowledge includes, e.g. historical logs of visited search areas, environmental data, and recorded change events. In contrast to previous work, where a memory is usually implemented by storing knowledge in data objects inside the EA, it is examined to which extent in-memory database technology can help increase and manage the amount of stored knowledge in order to better guide the optimization process. Furthermore, the database is employed as a storage for predictive knowledge that is accessed and analyzed to make the optimizer better prepared for prospected changes, to quickly respond to such changes and to easier recover from their impact.

The remainder of the paper is structured as follows: Sect. 2 introduces to dynamic evolutionary computing. Section 3 briefly reviews related work and outlines how IMDEA contributes to progress beyond. Section 4 describes the architecture and the methods of the system implemented. Subsequently, a sample of results from extensive experiments are presented in Sect. 5. Concluding remarks are provided in Sect. 6.

2 Dynamic Evolutionary Optimization

Prior to introducing the EA, the static Knapsack Problem shall be defined and henceforth be used to exemplify the strategies proposed in this paper. In its static variant, the 0/1 Knapsack Problem [3] is described by a set of n items of weight w_j and value v_j where $j \in \{1, \ldots, n\}$. A candidate solution $X = (x_1, \ldots, x_n)$ represents a subset of all items, with $x_j \in \{0, 1\}$ indicating if item j is included in the knapsack which has a capacity of C. The goal is to maximize the total value of items included in the knapsack such that the sum of their weights is less or equal to the knapsack capacity: Maximize

$$f(X) = \sum_{j=1}^{n} v_j x_j \text{ subject to } \sum_{j=1}^{n} w_j x_j \leq C, x_j \in \{0, 1\}. \tag{1}$$

As the Knapsack Problem is known to be $NP-hard$, EA [1] are one possible heuristic to search for near optimal solutions. Inspired by the principles of natural evolution, the main idea behind evolutionary optimization is to represent solutions of an optimization problem as a set of individuals called population. The size of the population shall be denoted as p. An individual $i \in \{1, \ldots, p\}$ is encoded in a chromosome X_i representing the individual's genotype. In the case of the Knapsack Problem, individual i is encoded as n-bit chromosome $X_i = (x_{i,1}, \ldots x_{i,n})$ with $x_{i,j} \in \{0, 1\}$, where $x_{i,j} = 1$ means that item number j is contained in the knapsack of individual i, and $x_{i,j} = 0$ otherwise. The *dynamic* Knapsack Problem introduces time-dependent variance: capacity $C(t)$, weights $w_j(t)$ and values $v_j(t)$ are considered dynamic over time t.

The goal of an EA is to incrementally improve the *fitness* of the best individual, which represents its solution quality, by mimicking the principles of natural selection, recombination, mutation and survival of the fittest (cf. [1] for more details). Overweight individuals are invalid but in dynamic problems it is more advisable to reduce their fitness by a penalty cost term than considering them as completely unsuitable, since a change event could lead an invalid solution to become valid or even the best. Hence, IMDEA calculates the fitness of individual i with genotype $X_i(t)$ as

$$fit\,(X_i(t)) = \sum_{j=1}^{n} (v_j(t)x_{ji}(t)) \cdot \left(1 - \max\left\{0; \frac{\sum_{j=1}^{n} (w_j(t)x_{ji}(t))}{C(t)} - 1\right\}^{\lambda}\right) \quad (2)$$

with external parameter λ representing the penalty weight.

For dynamic optimization the goal is not to localize a global stationary optimum but to track moving optima [4]. It is assumed that the problem instances before and after a change are related to each other, thus reusing prior optimization knowledge is more beneficial than a restart [5]. If prior solutions are intended to be reused, good individuals will have to be stored in a so-called *direct* memory. A memory that additionally stores information on the corresponding problem instance is called *associative* memory [6,7]. It allows to reuse individuals that had been successful under similar circumstances. Predictive analysis can be used for tracking optima by calculating the prospected path of an optimum through the solution space or by anticipating the nature of the next change [8,9]. A successful dynamic EA should include a *memory* and a *predictive* component and it should maintain *diversity* throughout the run because a diverse population can better react to a change than a converged one [4].

3 Related Work and Progress Beyond

Related work started with early contributions by Fogel et al. [10] and Goldberg [1]. A recent state of the art survey by Nguyen et al. [4] summarizes papers on evolutionary dynamic optimization of the past 20 years, benchmark generators and performance measures. Cruz et al. [11] provide a list of about 40 artificial and real world problems as well as papers addressing them.

Hatzakis et al. [12] integrate auto-regression and moving average analysis into a multiobjective EA to forecast optimal regions. Rossi et al. [8] use an EA that learns the movement of the optimum and adjusts the fitness function accordingly to force the population into promising areas of the solution space. Simões and Costa [9,13–15] published several papers on linear and nonlinear regression to predict the generation of the next change. They use *Marcov Chains* to anticipate the nature of the next change and they include a direct memory.

Fitness Sharing [16] is a widely used diversity management technique. Fitness sharing calculates the so-called *shared fitness* for each individual i depending on its distance $d\,(i, k)$ (e.g. hamming distance) to all other individuals k:

$$fit^*\,(X_i(t)) = \frac{fit\,(X_i(t))}{\sum_{k=1}^{p} \left(1 - \min\left\{1; \frac{d\,(i,k)}{\sigma_s}\right\}^{\alpha}\right)}. \quad (3)$$

Parameter α is a constant which defines the shape of the sharing function and is commonly set to 1 [17]. Further strategies to maintain diversity are, e.g., *Deterministic Crowding Selection* [18] and *Mating Restricted Tournament (MRT)* [19].

Grefenstette et al. [7] published one of the early papers on an associative memory. Branke [5] worked on direct memory and suggests to compute an *importance value* for each individual to decide which individuals to store in the memory. Yang introduced EA with direct [20] and with associative [21,22] memory.

Previous approaches to memory-extended EAs appear to be implemented as data objects within the algorithm. Online analytical processing using IMDB was explored by Plattner [23], however an extensive review of prior work yields that no previous publication on evolutionary computation has ever used database technology as knowledge store. Hence, this paper introduces IMDEA, an approach to dynamic evolutionary optimization exploiting IMDB technology as storage for associative memory and for prediction knowledge.

4 The Dynamic In-Memory Database Evolutionary Algorithm (IMDEA)

4.1 In-Memory Databases

Plattner [2] introduces the characteristics and advantages of IMDB: The rapid decline of prices for RAM storage during the last decade comes along with an increase in chip capacities and expedited access times. IMDBs provide a vast amount of storage capacity to the database residing entirely in main memory. And compared to disc-resident databases, data access times are reduced dramatically. This can also be attributed to flexible table encoding (row-store or column-store), as well as improved data compression and partitioning techniques. Therefore IMDBs suggest themselves as a technology to support dynamic evolutionary computing. This paper proposes to use the IMDB to store and efficiently maintain optimization knowledge in terms of an associative memory and prediction data. The storage volume required depends on (1) the optimization problem instance, (2) the extraction and replacement strategy, and (3) the number of generations executed. A set of preliminary experiments indicates that such storage requirements can easily exceed 1 GB of data, effectively managed by an IMDB. The product chosen for this paper is SAP HANA (High Performance Analytics Appliance) [24], which is the market-leading IMDB [25], and it has shown to excel in many application scenarios [25] with practical relevance. The IMDEA system is implemented in *SAP HANA Extended Application Services* [26], which is the common approach for native HANA applications.

4.2 Architectural Overview

Figure 1 visualizes the architecture of the IMDEA. The core algorithm, based on the dynamic EA approach introduced in Sect. 2, is started reading a set of parameters and the problem definition. Since the problem is considered dynamic, a simulator continuously adapts this problem definition, whereupon the changes are propagated to the core algorithm. A set of individuals is perpetually extracted from the current population and persisted into an associative memory held in an IMDB. Whenever necessary this memory is queried for suitable individuals which are injected into the core algorithm. A predictive analytics component processes the knowledge stored in the IMDB to better prepare the algorithm for forthcoming dynamic changes.

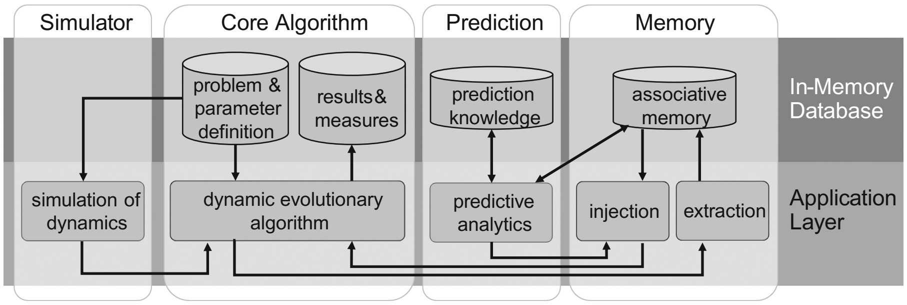


Fig. 1. Architectural overview

4.3 Simulator

For simulating a dynamically changing environment, several approaches are proposed in literature. Simulators like the *Moving Peaks Benchmark* by Branke [5] are not suitable for binary encoded problems like the Knapsack Problem as they assume real-valued search spaces. The *XOR Generator* by Yang et al. [22] is designed for binary encoded problems but creates dynamics by manipulating genotypes. A better approach would be to adjust the problem definition itself. The framework proposed by Li and Yang [27] appears to be an adequate approach and therefore inspired the implementation of the IMDEA simulator. This approach is closer to real world dynamics, avoids manipulating the population and leads to measurable dynamic environments which are crucial for an associative memory that stores environmental information.

The simulator creates environments $e(t)$, which are considered to represent the definition of the optimization problem at time t, where the time is supposed to be the generation number. The environment is constituted by a tuple of problem parameters that are subject to change. In the case of the knapsack problem, the environment $e(t) = (C(t), v(t), w(t))$ shall be signified by the knapsack capacity $C(t)$, its item weights $w(t) = (w_1(t), \ldots, w_n(t))$ and item values

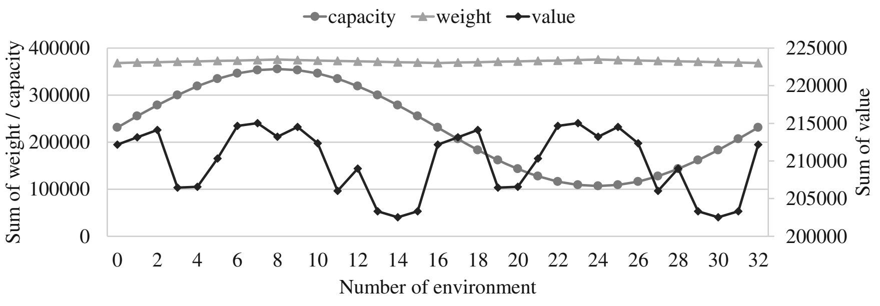


Fig. 2. Sample scenario simulating a dynamic knapsack problem, cycle length $L = 32$.

$v(t) = (v_1(t), \ldots, v_n(t))$. The simulator is implemented to change the environment at a constant frequency of T generations. Furthermore, it is assumed that the environment changes in a cyclic manner at a cycle length of L. Hence any environment will recur every $L{\cdot}T$ generations. The specific environment sequence used in this paper is visualized in Fig. 2.

4.4 Pseudocode for IMDEA

Listing 1.1 illustrates the IMDEA as pseudocode. The functions concerning memory and prediction are described in Sects. 4.5 and 4.6. After preparing the IMDB for the upcoming optimization and loading the problem definition and several parameters (line 1) the IMDEA initializes all necessary variables (line 2–6). There are population-objects for temporary saving parents, children, individuals from the memory and individuals provided by the predictive component (line 3–4).

The population is initialized (line 7), whereupon the population size is defined by one of the loaded parameters and the IMDEA enters its main while-loop. If a change is predicted for the current generation t, IMDEA will update the population with the individuals provided by the predictive component (line 9). In the first generation this condition is false. Changes are simulated every T generations (line 10) and the fitness of all individuals is evaluated (line 13).

Lines 12–22 are entered every time a change occured: If the accuracy of the predictive component for the current change lies below a threshold, that means that the change was anticipated badly and there is a high possibility that the individuals, which had been provided by the predictive component to prepare the EA for this change, are not suitable for the new environment. Thus the memory will be searched for individuals from a similar environment to update the population (line 13–17). Otherwise no measures will be taken.

Lines 18–21 illustrate the prediction cycle that is run after every change: The new change is saved in the IMDB (line 18). Using this prediction knowledge the IMDEA anticipates the generation of the next change and the expected environment (line 19–20). Afterwards IMDEA searches the associative memory for individuals that had been successful in an environment similar to the expected one (line 21, cf. Sect. 4.6 for more detail). These individuals will be used to update the population before the next change (line 9).

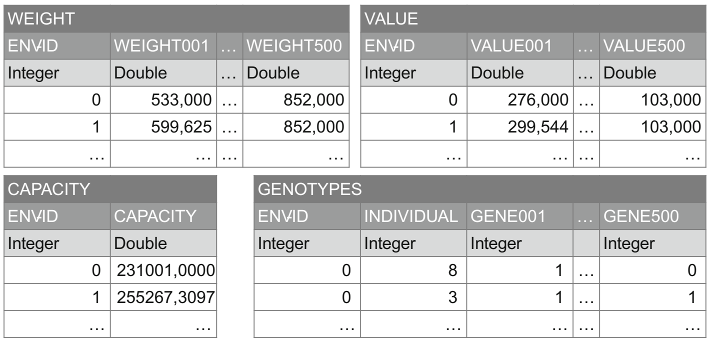
Lines 23–26 execute the standard functions of an EA, namely selection of at least two parents, recombination, mutation and population update. Extraction of good individuals to the memory is performed in line 27 (cf. Sect. 4.5 for more details). Finally the algorithm enters the next generation unless the stopping condition is met.

```
1  $initializeIMDB()$; $env = loadProblem()$; $par = loadParameters()$;
2  $pop = newPopulation()$; $fitness = newFitnessArray()$;
3  $parents = newPopulation()$; $children = newPopulation()$;
4  $memoryPop = newPopulation()$; $predictionPop = newPopulation()$;
5  $acc = 0.0$; $predictedGen = -1$; $predictedEnv = newEnvironment()$;
6  $t = 0$;
7  $pop = initializePopulation(par.populationSize)$;
8  while (stopping condition not met)
9      if ($t == predictedGen$) $pop = updatePopulation(predictionPop)$; endif
10     if ($(t \bmod par.T) == 0$) $env = simulateChanges(env)$; endif
11     $fitness = evaluateFitness(pop)$;
12     if (change occured in current generation $t$)
13         $acc = computePredictionAccuracy(env, predictedEnv)$;
14         if ($acc < par.accuracyThreshold$)
15             $memoryPop = getIndividualsFromMemory(env)$;
16             $pop = updatePopulation(memoryPop)$;
17         endif
18         $saveChangesInPredictionKnowledge(env)$;
19         $predictedGen = predictGenerationOfNextChange()$;
20         $predictedEnv = predictNewEnvironment()$;
21         $predictionPop = getIndividualsFromMemory(predictedEnv)$;
22     endif
23     $parents = selectForReproduction(pop, fitness, par.numberOfParents)$;
24     $children = recombineParents(parents)$;
25     $children = mutateIndividuals(children)$;
26     $pop = updatePopulation(pop, children)$;
27     if ($(t \bmod par.extractionPeriod) == 0$) $extraction(pop, fitness)$; endif
28     $t = t + 1$;
29 endwhile
```

Listing 1.1. Pseudocode for IMDEA

4.5 Associative Memory

An *associative* memory has the advantage that after a change the EA can reuse individuals that had been successful in a similar environment before. It needs to address four main issues [4]: (1) how to organize the memory, (2) when to *extract* which individuals from the EA to the memory (3) how to *update* the memory and (4) when to *inject* individuals from the memory to the EA (cf. Fig. 1).



WEIGHT			
ENV-ID	WEIGHT001	…	WEIGHT500
Integer	Double	…	Double
0	533,000	…	852,000
1	599,625	…	852,000
…	…	…	…

VALUE			
ENV-ID	VALUE001	…	VALUE500
Integer	Double	…	Double
0	276,000	…	103,000
1	299,544	…	103,000
…	…	…	…

CAPACITY	
ENV-ID	CAPACITY
Integer	Double
0	231001,0000
1	255267,3097
…	…

GENOTYPES				
ENV-ID	INDIVIDUAL	GENE001	…	GENE500
Integer	Integer	Integer		Integer
0	8	1	…	0
0	3	1	…	1
…	…	…	…	…

Fig. 3. Database schema of associative memory for knapsack instance with 500 items

The associative memory of IMDEA is organized in column tables on the IMDB. Figure 3 illustrates database schema of the memory for a knapsack instance with $j = 500$ items. The tables are organized according to the problem definition. Table *GENOTYPES* stores good individuals of earlier generations. Each column *GENE-j* ($j = \{001..500\}$) stores one gene x_j, which is 1 if item j is chosen, and 0 otherwise (cf. Eq. 1). The other three tables contain environmental information. Every environment is referenced by an *ENV-ID* in the first column of all tables. Column *WEIGHT-j/VALUE-j* in table *WEIGHT/VALUE* stores weight w_j, or value v_j of item j, respectively. When a change is detected, the IMDEA compares the current environment to the environmental information from the memory. If a suitable environment is found in the memory, good solutions from table *GENOTYPES* are injected into the IMDEA. These solutions had previously been successful in a similar environment. The *similarity* of two environments $e_1 = e(t_1)$ and $e_2 = e(t_2)$ at times t_1 and t_2 is computed as a weighted sum as follows:

$$sim(e_1, e_2) = \eta_c \cdot sim_C(e_1, e_2) + \eta_w \cdot sim_W(e_1, e_2) + \eta_v \cdot sim_V(e_1, e_2) \tag{4}$$

with $\eta_c, \eta_w, \eta_v \in [0, 1]$ and $\eta_c + \eta_w + \eta_v = 1$. Similarities sim_C, sim_W and sim_V signify the proportion of capacities ($C(t_1)$ and $C(t_2)$), weights ($w(t_1)$ and $w(t_2)$), and values ($v(t_1)$ and $v(t_2)$), respectively:

$$sim_C(e_1, e_2) = \min\left\{\frac{C(t_1)}{C(t_2)}, \frac{C(t_2)}{C(t_1)}\right\}, \tag{5}$$

$$sim_W(e_1, e_2) = \frac{1}{n} \cdot \sum_{j=1}^{n} \min\left\{\frac{w_j(t_1)}{w_j(t_2)}, \frac{w_j(t_2)}{w_j(t_1)}\right\}, \tag{6}$$

$$sim_V(e_1, e_2) = \frac{1}{n} \cdot \sum_{j=1}^{n} \min\left\{\frac{v_j(t_1)}{v_j(t_2)}, \frac{v_j(t_2)}{v_j(t_1)}\right\}. \tag{7}$$

The min-function normalizes the outcome between 0 and 1. An alternative way to calculate sim_W and sim_V is

$$sim_W(e_1, e_2) = \frac{1}{n} \cdot \sum_{j=1}^{n} st_W(e_1, e_2, j) \text{ with} \tag{8}$$

$$st_W(e_1, e_2, j) = \begin{cases} 1, & \text{if } \min\left\{\frac{w_j(t_1)}{w_j(t_2)}, \frac{w_j(t_2)}{w_j(t_1)}\right\} \geq \tau_W \\ 0 & \text{otherwise} \end{cases}. \tag{9}$$

The first calculation (6, 7) is more flexible because it does not declare items as dissimilar based on a threshold. The second calculation (8, 9) on the other hand allows for a strict control of the similarity threshold τ_W if required and prevents the commingling of similarities of different items. This paper uses the second calculation because it focuses on the performance of the IMDEA for environments that reappear in exactly the same way.

The *extraction* of good individuals from the population is performed at equally spaced intervals. Copies of the individuals are stored in the IMDB. Yang [20] uses dynamic time patterns for extraction to reduce the risk that extraction and change coincide. As this paper combines an associative memory with predictive analytics, the prediction on change periods can be used to adapt the extraction period accordingly. As recommended by Grefenstette and Ramsey [7] we extract 50% of the population after a change. To decide which individuals to extract the IMDEA calculates an *importance value* based on [5] for each individual i as

$$imp(i) = \gamma_f \cdot imp_{fit}(i) + \gamma_d \cdot imp_{div}(i) + \gamma_a \cdot imp_{age}(i) \tag{10}$$

with $\gamma \in [0, 1]$ and $\gamma_f + \gamma_d + \gamma_a = 1$. The terms $imp_{fit}(i)$, $imp_{div}(i)$ and $imp_{age}(i)$ express the relative importance of individual i with respect to the fitness, diversity and age of the population. At generation t these importance terms are computed as follows:

$$imp_{fit}(i) = \frac{fitness(i)}{\sum_{k=1}^{p} fitness(k)}, \tag{11}$$

$$imp_{div}(i) = \frac{\sum_{k=1}^{p} d(i, k)}{\sum_{h=1}^{p} \sum_{k=1}^{p} d(h, k)}, \tag{12}$$

$$imp_{age}(i) = \frac{age(i, t)}{\sum_{k=1}^{p} age(k, t)}, \tag{13}$$

where $d(h, k)$ is the Hamming distance between individuals h and k. The age of an individual i in generation t is $age(i, t) = 0$ if the individual was created in generation t, and $age(i, t-1) + 1$ otherwise. The population is sorted descending by importance and the 50% with the highest importance value are extracted.

If the current environment does not exist in the memory so far, the new environmental information is stored in the in-memory database and the extracted individuals are *inserted* to the memory. The IMDEA checks whether the current environment already exists in the memory by applying Eq. 4. If a similar environment exists in the memory, the extracted individuals will be used to *update* table $GENOTYPES$ (see Fig. 3). In this case the extracted individuals and the memory individuals are merged and Eq. 10 is used to decide which individuals become the new memory individuals.

When the associative memory is called by the IMDEA (i.e. due to a new prediction output or during *injection* after an unpredicted change, cf. Sect. 4.4), the environmental tables in the memory are searched for an environment similar to the new environment based on Eq. 4. If the search is successful, the stored individuals from the memory will replace similar individuals in the population of the EA. Otherwise only immigrants will be generated to increase diversity.

4.6 Change Prediction

The associative memory component interacts with the predictive analytics component. Prediction is triggerd after each change and aims to anticipate the generation and nature of the next change. Therefor *prediction knowledge* on previous changes is stored in the in-memory database (cf. Fig. 1). This paper uses the *Predictive Analysis Library (PAL)* [28]. HANA is organized in two layers [26]. Applications, like the implemented EA, run as part of the *control flow logic*. Interaction with the data is controlled by the *calculation logic*. PAL is part of the calculation logic and is thus very suitable for the IMDEA because the predictive algorithms run close to the data they analyze. Furthermore PAL is well compatible with the HANA IMDB. Its functionality is based on stored procedures. The input data has to be stored in database tables and must be organized in the specific way required by the respective procedure.

PAL comprises functions for statistics, time series analysis, regression, clustering, classification and preprocessing of data [28]. Simões et al. [9] showed the effectiveness of regression for prediction. Therefore this paper uses polynomial regression to predict upcoming changes. Other methods like *forecast smoothing* or *neural networks* are potential candidates for predictive analytics as well but polynomial regression has a slight advantage regarding computation time. Based on the previous changes that are stored in the *prediction knowledge*, the IMDEA calls the stored procedure from PAL for polynomial regression to calculate

- the anticipated generation of the next change,
- which of the parameters C, w_j and v_j are going to change and
- how they will change (cf. Listing 1.1).

PAL stores its output in dedicated database tables from where the results are selected. Based on the output of the predictive analytics component the expected environment can be simulated and the IMDEA searches the associative memory for individuals, which had been successful in an environment similar to the simulated one. If such individuals are found, they remain in a temporary buffer

PREDICT-NEXT-GENERATION-INPUT		
ID	Y	X1
Integer	Integer	Integer
...	...	...
4	28000	4
5	35000	5
6	42000	6

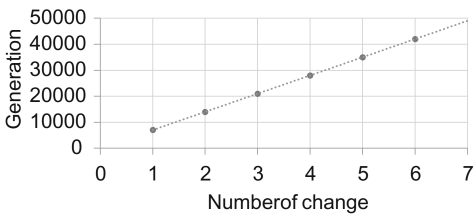


Fig. 4. Prediction of the generation of the next change using regression

in order to be injected right before the anticipated change occurs. If no suitable individuals are found in the memory, immigrants will be inserted to the population when the next change occurs to increase diversity.

Figure 4 illustrates an example for the calculation of the generation of the next change in a scenario where change occurs every 7000 generations ($T = 7000$). The input table for the stored procedure is organized in the way required by PAL [28, p. 273]. The dotted line in the diagram illustrates the corresponding regression function, which is used to calculate the generation of the 7^{th} change. The next step of the prediction is to analyze which of the parameters C, w_j and v_j are going to change. Therefor information on how often the parameters changed before is used: the more often they changed, the higher the likelihood that they are going to change next time. Afterwards their new numerical value is anticipated with the same PAL procedure as described above.

In order to measure the efficiency of the implemented predictive analysis, every time a change occurs the *prediction accuracy* is calculated as $acc = 1 - err$. The prediction error err is One if the actual change occurs too early or if no prediction was made at all. Otherwise err is calculated as the arithmetic mean of the relative error of each parameter.

5 Evaluation Results

Extensive tests were conducted to evaluate the performance the associative memory and the predictive component of IMDEA. This paper uses a knapsack instance with 500 items [29], a constant population size p of 40 individuals, *mating restriction* [19] and *Fitness Sharing* to maintain diversity with $\alpha = 1$ and $\sigma_s = 257.6$ based on [16] and a quadratic penalty in the fitness function ($\lambda = 0.5$, Eq. 2).

Based on *Design of Experiment (DoE)* principles a 2^3 full factorial design was prepared to evaluate the effect of the three factors *associative memory*, *polynomial regression prediction* and *diversity maintenance*. In the diagrams each factor combination is coded with three letters, the first one indicating whether predictive analysis was used (P) or not (O), the second one indicating the same for diversity maintenance (D or O) and the third one standing for associative memory (M or O). For each factor combination ten tests were run. Each test ran for eight hours simulating dynamic changes with $T = 7000$ and $L = 32$ (cf. Sect. 4.3, Fig. 2) resulting in more than 1000000 generations, 150 changes and five repetitions of each environment. The main performance measures for evaluation are *decrease* and *recovery* of the fitness after a change, mean *best fitness* per factor, *computation time* and *prediction accuracy*.

Figure 5 compares the best fitness per generation. The best fitness declines over the generations because the diagram shows a stage of decreasing capacity. Figure 6 shows the arithmetic mean of the percentaged decrease of the best fitness per factor combination averaged over all changes as well as the arithmetic mean of the best fitness over all generations (cf. [4], $\overline{F}_{BOG}$).

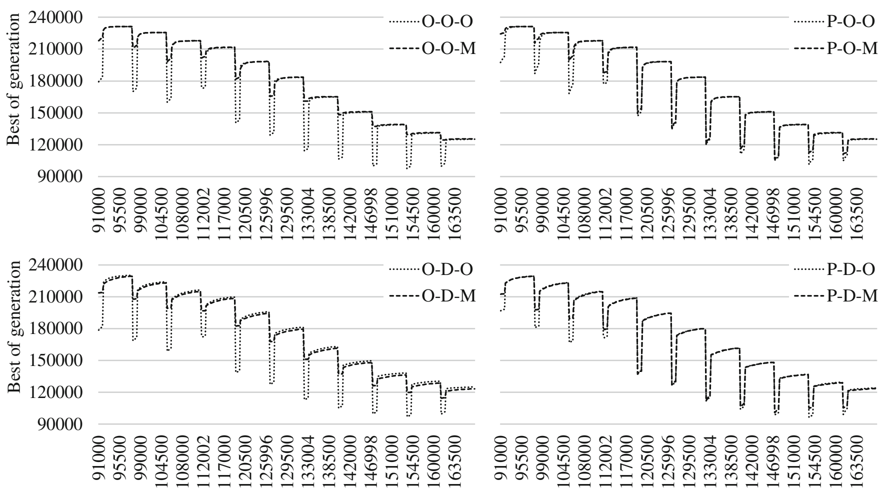


Fig. 5. Excerpt of the best-of-generation

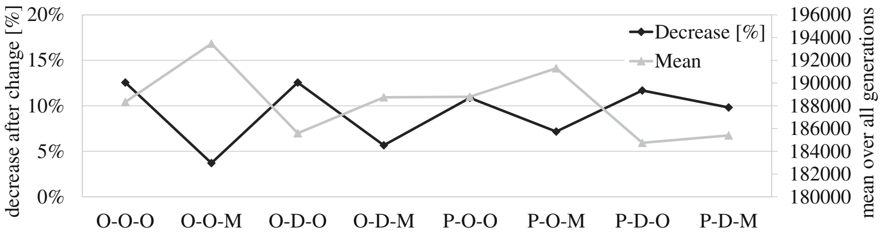


Fig. 6. Mean best fitness and percentaged decrease of best fitness

Both diagrams clearly show, that using the IMDB as associative memory for the IMDEA has a positive effect on the fitness for recurring environments. In the scenario with no dynamic adaptation (O-O-O) the average decrease of the best fitness after a change is 12.55%. However when using an IMDB memory (O-O-M) the decrease drops to only 3.70%. That is merely about a quarter of the decrease of the non-adapted EA. Correspondingly the implemented memory in average also realizes the highest outcome for the best fitness.

On the other hand, Figs. 5 and 6 show that combining the associative memory with *Fitness Sharing*, mating restriction and polynomial regression prediction does not further improve the algorithm but diminishes it. Using only diversity maintenance (O-D-O) or prediction (P-O-O) results in an improvement compared to the non-adapted EA, too (O-O-O). If prediction or diversity maintenance is used and the memory is added (O-D-M, P-O-M, P-D-M), the results will be better than without memory (O-D-O, P-O-O, P-D-O). But no factor combination leads to better results than memory alone.

The idea behind diversity maintenance is that a diverse population can adapt to changes more easily than a converged population. Yet for IMDEA it diminishes the algorithm. A logical explanation is the difficulty of finding an appropriate value for σ_s, as pointed out by Sareni et al. [17]. The test results suggest that *Fitness Sharing* is not always the best solution for diversity maintenance.

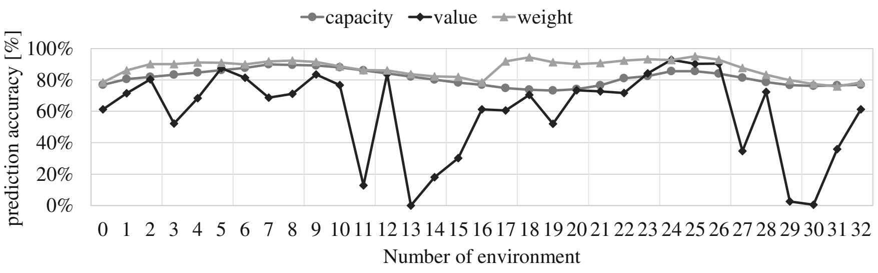


Fig. 7. Prediction accuracy per environment

The reason for the negative effect of the predictive component is not the prediction itself but the hardness of predictability. Figure 7 illustrates the prediction accuracy (cf. Sect. 4.6) for each environment. The simulator was programmed to simulate changes that are hard to predict on purpose, because easy changes such as linear increase are no challenge for predictive analysis. Therefore the simulator includes a sine function (capacity, cf. Fig. 2) which is relatively easy to predict and small jumps (value, cf. Fig. 2) based on [27] which are hard to predict. Figure 7 shows that polynomial regression can estimate a sine function with an accuracy of about 82%. However the jumps are so hard to predict, that the polynomial regression varies significantly from the actual jumps. As a result there are some cases where the memory is queried for individuals from the wrong environment which are then inserted to the EA before the next change. Due to these individuals originating from the wrong environment the prediction weakens the performance of the memory.

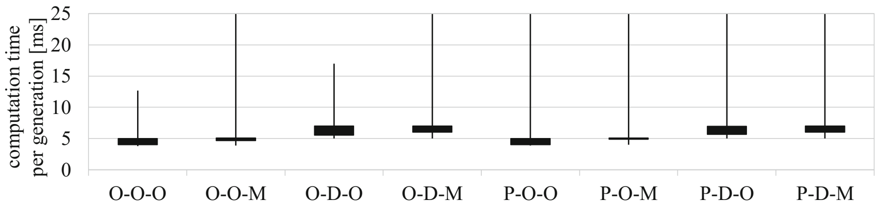


Fig. 8. Computation time per generation

Communication between IMDEA and the IMDB needs to be taken into account when evaluating its performance. IMDEA is implemented as an application within the IMDB thus reducing communication overhead to a minimum. It uses PAL for predictive analysis to ensure that the prediction is carried out as close as possible to the stored data. Figure 8 illustrates the computation time in milliseconds per generation for each factor combination as box plot diagram. The ordinate is limited to 25 $\frac{\text{ms}}{\text{g}}$. Diversity maintenance raises the average computation time from 5 $\frac{\text{ms}}{\text{g}}$ to 7 $\frac{\text{ms}}{\text{g}}$ because extra time to compute the shared fitness is required. Prediction and memory do not influence the average computation time but the maximum values every T generations: During each prediction cycle more than 30 s of computational cost are lost in the IMDB. This is due to the used library and can not be improved by IMDEA. *Extraction* and *injection* require 500 ms to read from the IMDB and to write into it.

Similar to results obtained by Yang [22] and Branke [5] these tests conducted with IMDEA prove the effectiveness of a memory, but in contrast to previous work IMDEA introduces the benefit of storing large amounts of data in an IMDB. IMDEA accomplishes a high prediction accuracy for linear changes but for noisy environments it is not able to outperform existing approaches like Simões [9].

After determining that memory alone has the best effect of the performance of the IMDEA, further test were conducted to evaluate interdependences between changes and the extraction periodicity. The results show, that there is no correlation between the extraction periodicity and the severity of jumps. However, there is a strong interdependence between the extraction periodicity and the frequency of change. If changes occur often, extraction will have to take place often as well. For a low frequency of change a less frequent extraction strategy is beneficial. This strong interdependence leads to the conclusion that in dynamic environments where the frequency of change itself is fluctuating, the EA performs best if the memory component constantly and automatically adapts its extraction points according to the frequency of change.

6 Conclusion

This paper reported on an implemented approach to dynamic evolutionary optimization exploring the possibilities of integrating in-memory computing into evolutionary algorithms. Empirical studies suggest that an in-memory database (e.g. SAP HANA) can enable the optimizer to learn from the decisions of the past and to make better informed decisions in the forthcoming iterations of the optimization algorithm. This positive effect of associative memory seems becomes particularly apparent in recurring environments. In such cases, the contribution of associative memory is strong. Using an IMDB allows storing and maintaining huge amounts of data on previously visited solutions. By implementing the optimizer as an application within the IMDB the time required for communication between the database and the optimizer is reduced to 500 ms. The test results also indicate, that there is a strong interdependence between the frequency of change and the extraction strategy, meaning that the interval for extraction needs to adapt to the frequency of change in order to ensure maximum efficiency of the associative memory.

Additionally, the results demonstrate that some changes (i.e. jumps) are hard to predict. To improve the accuracy of predictive analysis of stored knowledge, further evaluation of suitable analysis methods such as different regression models, forecast smoothing or neural networks may be explored. The optimizer could include a learning component to automatically improve the selection of the best prediction method. Alternate approaches of implementing the predictive component – besides PAL – should be tested to reduce the computation time of the prediction cycle. Future work targets the associative memory by testing alternate database schemes addressing typical access patterns. It is also envisaged to apply IMDEA to other usage scenarios including, e.g., constrained-based product configuration systems [30], a form of the SAT problem.

Acknowledgments. The work for this paper was generously supported by the HPI Future SOC Lab in the scope of the project "Big Data in Bio-inspired Optimization".

References

1. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading (1989)
2. Plattner, H.: A Course in in-Memory Data Management: The Inner Mechanics of in-Memory Databases. Springer, Heidelberg (2014)
3. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Heidelberg (2004)
4. Nguyen, T.T., Yang, S., Branke, J.: Evolutionary dynamic optimization: a survey of the state of the art. Swarm Evol. Comput. **6**, 1–24 (2012)
5. Branke, J.: Memory enhanced evolutionary algorithms for changing optimization problems. In: Congress on Evolutionary Computation, CEC 1999, pp. 1875–1882 (1999)
6. Yang, S.: Explicit memory schemes for evolutionary algorithms in dynamic environments. In: Yang, S., Ong, Y.S., Jin, Y. (eds.) Evolutionary Computation in Dynamic and Uncertain Environments. SCI, vol. 51, pp. 3–28. Springer, Berlin London (2007)
7. Grefenstette, J.J., Ramsey, C.L.: Case-based initialization of genetic algorithms. In: Proceedings of the 5th ICGA, pp. 84–91 (1993)
8. Rossi, C., Abderrahim, M., Díaz, J.C.: Tracking moving optima using Kalman-based predictions. Evol. Comput. **16**, 1–30 (2008)

9. Simões, A., Costa, E.: Prediction in evolutionary algorithms for dynamic environments. Soft Comput. **18**, 1471–1497 (2014)
10. Fogel, L., Owens, A., Walsh, M.: Artificial Intelligence Through Simulated Evolution. Wiley, New York (1966)
11. Cruz, C., Gonzalez, J.R., Pelta, D.A.: Optimization in dynamic environments: a survey on problems, methods and measures. Soft Comput. **15**, 1427–1448 (2011)
12. Hatzakis, I., Wallace, D.: Dynamic multi-objective optimization with evolutionary algorithms. In: Cattolico, M. (ed.) Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO 2006, pp. 1201–1208. ACM, New York (2006)
13. Simões, A., Costa, E.: Variable-size memory evolutionary algorithm to deal with dynamic environments. In: Giacobini, M. (ed.) EvoWorkshops 2007. LNCS, vol. 4448, pp. 617–626. Springer, Heidelberg (2007). doi:10.1007/978-3-540-71805-5_68
14. Simões, A., Costa, E.: Evolutionary algorithms for dynamic environments: prediction using linear regression and Markov chains. In: Rudolph, G., Jansen, T., Beume, N., Lucas, S., Poloni, C. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 306–315. Springer, Heidelberg (2008). doi:10.1007/978-3-540-87700-4_31
15. Simões, A., Costa, E.: Prediction in evolutionary algorithms for dynamic environments using Markov chains and nonlinear regression. In: Rothlauf, F. (ed.) Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO 2009, pp. 883–890. ACM, New York (2009)
16. Deb, K., Goldberg, D.E.: An investigation of niche and species formation in genetic function optimization. In: Proceedings of the 3rd ICGA, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc., pp. 42–50 (1989)
17. Sareni, B., Krähenbühl, L.: Fitness sharing and niching methods revisited. IEEE Trans. Evol. Comput. **2**, 97–106 (1998)
18. Mahfoud, S.W.: Niching methods for genetic algorithms. Ph.D. thesis, University of Illinois UMI Order No. GAX95-43663 (1995)
19. Ishibuchi, H., Shibata, Y.: Mating scheme for controlling the diversity-convergence balance for multiobjective optimization. In: Deb, K. (ed.) GECCO 2004. LNCS, vol. 3102, pp. 1259–1271. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24854-5_121
20. Yang, S.: Memory-based immigrants for genetic algorithms in dynamic environments. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, GECCO 2005, pp. 1115–1122. ACM, New York (2005)
21. Yang, S.: Associative memory scheme for genetic algorithms in dynamic environments. In: Rothlauf, F., Branke, J., Cagnoni, S., Costa, E., Cotta, C., Drechsler, R., Lutton, E., Machado, P., Moore, J.H., Romero, J., Smith, G.D., Squillero, G., Takagi, H. (eds.) EvoWorkshops 2006. LNCS, vol. 3907, pp. 788–799. Springer, Heidelberg (2006). doi:10.1007/11732242_76
22. Yang, S., Yao, X.: Population-based incremental learning with associative memory for dynamic environments. Evol. Comput. **12**, 542–561 (2008)
23. Plattner, H.: A common database approach for OLTP and OLAP using an in-memory column database. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, pp. 1–2. ACM, New York (2009)
24. Silvia, P., Frye, R., Berg, B.: SAP HANA - An Introduction. Rheinwerk Verlag, Birmingham (2016)
25. Plattner, H., Leukert, B.: The In-Memory Revolution: How SAP HANA Enables Business of the Future. Springer, Heidelberg (2015)
26. SAP: SAP HANA XS JavaScript Reference: SAP HANA Platform SPS 12, Document Version: 1.0, 11 May 2016

27. Li, C., Yang, S.: A generalized approach to construct benchmark problems for dynamic optimization. In: Li, X., Kirley, M., Zhang, M., Green, D., Ciesielski, V., Abbass, H., Michalewicz, Z., Hendtlass, T., Deb, K., Tan, K.C., Branke, J., Shi, Y. (eds.) SEAL 2008. LNCS, vol. 5361, pp. 391–400. Springer, Heidelberg (2008). doi:10.1007/978-3-540-89694-4_40
28. SAP: SAP HANA Predictive Analysis Library (PAL): SAP HANA Platform SPS 11, Document Version: 1.0, 25 November 2015
29. Beasley, J.E.: mknapcb3 (2004). http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/mknapcb3.txt. Accessed 25 August 2016
30. Klein, M., Greiner, U., Genßler, T., Kuhn, J., Born, M.: Enabling interoperability in the area of multi-brand vehicle configuration. In: Gonçalves, R.J. (ed.) Enterprise Interoperability II, pp. 759–770. Springer, London (2007)