Real-Parameter Optimization by Iterative Prototype Optimization with Evolved Improvement Steps

Jiří Kubalík

***Abstract*— Evolutionary algorithms are typically used to evolve a population of complete candidate solutions to a given problem. Recently, a novel framework called *Iterative Prototype Optimization with Evolved Improvement Steps* has been proposed. This is a general optimization framework, where a possible improvement of a prototype solution is being evolved by the evolutionary algorithm. The framework has already been used to solve binary string optimization problems and the combinatorial optimization problem. In this paper we use this optimization framework to solve real-parameter optimization problems. The algorithm was tested on problems collected for the Special Session on Real-Parameter Optimization of the IEEE Congress on Evolutionary Computation 2005. The achieved results show a potential of the presented optimization framework for solving hard real-parameter optimization problems.**

I. Introduction

Evolutionary algorithms (EAs) have already been more or less successfully applied to a wide range of optimization problems. They are typically used to evolve a population of candidate solutions to a given problem. Each of the candidate solutions encodes a complete solution - i.e. a complete set of the problem control parameters, a complete schedule in case of scheduling problems, a complete tour for the traveling salesman problem, etc. This implies, that especially for large instances of the solved problem the EA operates on an enormous huge space of potential solutions, so it might easily fail to find the optimal (or at least good enough) solution.

Recently, a new approach called Iterative **P**rototype **O**ptimisation with **E**volved I**M**provement **S**teps (POEMS) [1] has been proposed, where the EA does not handle the complete candidate solution to the problem at hand. Instead, the EA is employed within the iterative optimisation framework to evolve the best modification of the current solution prototype in each iteration. Thus, the total load of searching for the best complete solution is cut into pieces, each of them representing a process of seeking for the best transformation of the current solution prototype to the new possibly better one. The POEMS algorithm has been tested on problems from two different optimisation problem domains - binary string optimisation and the traveling salesman problem. The results revealed interesting aspects of using the algorithm for the two classes of optimisation problems. The aim of this paper is to show that the algorithm can be used for solving real-parameter optimisation problems as well.

The paper is structured as follows. In section 2, the general outline of the algorithm of POEMS is described. In section 3, the implementation of the algorithm for the real-parameter optimisation problems is described. Sections 4 and 5 describe the problems and the experimental set-up used for the proof-of-concept validation of POEMS. In section 6, the results achieved with POEMS are presented and compared with other evolutionary approaches. The paper ends with conclusions on effectiveness of POEMS, and its possible further extensions.

Jiří Kubalík is with the Department of Cybernetics, Czech Technical University in Prague, Technická 2, 166 27 Prague, Czech Republic (phone: 420-224-357-391; fax: 420-224-357-227; email: kubalik@labe.felk.cvut.cz).

II. POEMS

The main idea behind POEMS (see Figure 1) is that some initial prototype[1] solution is further improved in an iterative process, where the most suitable modification of the current prototype is sought for using an evolutionary algorithm (EA) in each iteration. The modifications are represented as a sequence of primitive actions/operations, defined specifically for the problem at hand. The evaluation of action sequences is based on how good/bad they modify the current prototype, which is an input parameter of the EA. Sequences that do not change the prototype at all are penalized in order to eliminate generating trivial solutions. After the EA finishes, it is checked whether the best evolved sequence improves the current prototype or not. If an improvement is found, then the sequence is applied to the current prototype and the resulting solution becomes the new prototype. Otherwise the current prototype remains unchanged for the next iteration.

Representation. The EA evolves linear chromosomes of length *MaxGenes*, where each gene represents an instance of certain action chosen from the set of elementary actions defined for the given problem. Each action is represented by a record, with an attribute *action_type* followed by parameters of the action. Besides actions that truly modify the prototype there is also a special type of action called *nop* (no operation). Actions with $action\_type$=nop are interpreted as void actions with no effect on the prototype, regardless of the values of their parameters. A chromosome can contain one or more instances of the *nop* operation. In this way the variable effective length of chromosomes is implemented. An important aspect of this implementation is that any temporarily inactivated action can be activated again later on (with its formerly evolved parameters) just by switching its $action\_type$ on.

Operators. The representation allows to use a variety of possible recombination and mutation operators such as standard 1-point, 2-point or uniform crossover and a simple gene-modifying mutation. In [1] a generalized uniform crossover

[1]The initial prototype can be generated either by random or using some problem specific heuristic.

```
1  generate(Prototype)
2  repeat
3      BestSequence ← run_EA(Prototype)
4      Candidate ← apply_to(BestSequence, Prototype)
5      if(Candidate is_better_than Prototype)
6          Prototype ← Candidate
7  until(POEMS termination condition)
8  return Prototype
```

Fig. 1. An outline of POEMS algorithm

was used, that forms a valid offspring as an arbitrary combination of parental genes. Both parents have the same probability of contributing its genes to the generated child, and each gene can be used only once. Mutation operator changes either the $action\_type$ (activates or inactivates the action) or the parameters of the action.

Evolutionary model. The design and configuration of the EA can differ for each particular optimisation problem. Figure 2 shows a simple steady-state evolutionary algorithm that iteratively modifies a population of individuals. In each iteration a couple of individuals is selected that undergoes the crossover operation with the specified crossover probability. The offspring is then modified by the mutation operation and assigned the fitness. Then one of the worst individuals in the current population is chosen as the replacement. Finally, the offspring is placed on the position of the replacement in the population if it is better than the replacement.

In general, the EA is expected to be executed many times during the whole run of the POEMS. Thus, it must be configured to converge fast in order to get the optimised action sequence in short time. As the EA is evolving sequences of actions to improve the solution prototype, not the complete solution, the maximal length of chromosomes $MaxGenes$ can be short compared with the size of the problem. For example, $MaxGenes = 10$ that is much smaller than the problem size of the binary string optimisation problems (200 bits) and much smaller than the number of cities (100-2000 cities) in case of the TSP problem was used in [1]. The relaxed requirement on the expected EA output and the small size of evolved chromosomes enables to setup the EA so that it converges within a few generations.

```
1   initialize(Population)
2   repeat
3       Parents ← select(Population)
4       if(rand() < Pcross)
5           Offspring ← cross_over(Parents)
6       else
7           Offspring ← Parents
8       mutate(Offspring)
9       evaluate(Offspring)
10      Replacement = find_loser(Population)
11      Population[Replacement] ← Offspring
12  until(EA termination condition)
13  BestSequence ← best_of(Population)
14  return BestSequence
```

Fig. 2. An outline of a simple steady-state EA

It is important to note, that POEMS does not perform prototype optimisation via improvement steps that are purely local with respect to the current prototype. In fact, long phenotypical as well as genotypical distances between the prototype and its modification can be observed if the system possesses a sufficient explorative ability, see [1]. The space of possible modifications of the current prototype is determined by the set of elementary actions and the maximum allowed length of evolved action sequences $MaxGenes$. The less explorative actions are and the shorter sequences are allowed the more the system searches in a prototype neighborhood only and the more it is prone to get stuck in a local optimum early, and vice versa.

III. Implementation Issues

POEMS is a general optimisation framework that can be used more or less effectively for any optimisation problem. The problem specific part is the implementation of the engaged EA. One has to design the representation of the evolved action sequences, genetic operators operating on the sequences, and the evolutionary model. Perhaps, the most important is a proper choice of the set of action types. They should be chosen so that the space of possible candidate action sequences is rich enough to ensure a sufficient exploration capabilities of the whole system.

Representation. In [1], the action sequences for the binary string optimisation problems were composed of just one type of action called $invert(gene)$. The action simply inverts specified $gene$ within the prototype. Thus, the EA was searching for the best prototype modification among the joint-mutations of up to $MaxGenes$=10 genes in each POEMS iteration. For the TSP problem a direct path representation of the tour was used. The prototype tour was modified by action sequences composed of actions of the following types

- $move(city_1, city_2)$ moves $city_1$ right after $city_2$ in the tour,
- $invert(city_1, city_2)$ inverts a subtour between $city_1$ and $city_2$,
- $swap(city_1, city_2)$ swaps $city_1$ and $city_2$.

Again, the EA search space contained a lot of possible action sequences of wide range of the impact they have on the prototype.

In this work, besides the nop action we have used effective actions of the following types

- $add(parameter)$ adds the value of $parameter$ to the respective prototype variable x_i. When initializing or mutating this action, the value of $parameter$ is chosen randomly from the interval $0.1 * (max_i - min_i)$, where max_i and min_i is the upper bound of variable x_i and the lower bound of variable x_i, respectively.
- $sub(parameter)$ subtracts the value of $parameter$ from the respective prototype variable x_i.

- $sample\_right(parameter)$ picks a new value of the respective prototype variable x_i from the interval (x_i, max_i) according to the formula

$$(max_i - x_i) \cdot \frac{e^{4 \cdot parameter} - 1}{e^4 - 1},$$

  This action moves the respective variable x_i towards its upper bound, preferring the smaller changes to the larger ones. Note, the effect of this action is quite similar to the sub action with the difference that it samples the interval (x_i, max_i) non-linearly with respect to the action parameter.
- $sample\_left(parameter)$ picks a new value of the respective variable x_i from the interval (min_i, x_i). The sampling strategy being the same as in case of $sample\_right$ action.

The chromosome (i.e. the candidate action sequence) is represented as an ordered list of D actions (active or inactive), each of them operating on the corresponding prototype variable x_i, where D is the dimension of the problem at hand. At least one of the actions in each sequence must be active with non-zero argument in order to eliminate trivial solutions (i.e. action sequences that do not modify the prototype at all).

Operators. The crossover operator used in this work is a combination of the standard uniform and arithmetical crossover operators. Given the pair of parental chromosomes $P1$ and $P2$, the offspring chromosome inherits $i-th$ action either from the first or the second parent. If the $i-th$ actions are of the same type in both parents then the value of the offspring action $parameter$ is picked randomly from the interval $(P1_{xi}, P2_{xi})$, otherwise the offspring action inherits the $parameter$ value from the same parent as the $action\_type$.

The mutation operator modifies either the $action\_type$ or the $parameter$. If the $action\_type$ is to be changed the new $action\_type$ out of the four effective plus the nop is chosen with a uniform probability. If the action parameter changes, then the new parameter value is pick from the interval $(0.9 * parameter, 1.1 * parameter)$ making sure that the new value lies within the interval specified for the given action, see above.

Evolutionary model. The steady-state evolutionary model shown in Figure 2 was implemented in this work. The action sequences in the initial population were generated with half-and-half active and nop actions. Moreover, the initial population of the EA in the second and later iterations were not generating completely from scratch. A portion of actions (specified by the parameter P_{reuse}) from the final population of the previous EA were reused instead. The motivation for that is (i) to keep the search direction that proved beneficial in the past and (ii) to reuse the unexploited yet possibly useful material (inactive actions or actions of other well-fit sequences besides the best one) evolved in the last iteration.

A tournament selection was used for selecting the parents. The inverse tournament strategy (i.e. the worst individual out of N competing ones) was used to find the replacements.

IV. Test Problems

The algorithm is tested on the 10D and 30D reference optimisation functions collected for the Special Session on Real-Parameter Optimization of the IEEE Congress on Evolutionary Computation 2005. There are 25 functions in the suit that are divided into the four groups – unimodal functions, multimodal basic functions, multimodal expanded functions, and multimodal hybrid composition functions. A detailed description of the functions is given in [2].

Due to the space limitations just one representative function of each group was chosen for our experiments:

- Shifted Sphere Function (Problem 1 in [2]). Properties: unimodal, shifted, separable, scalable.
- Shifted Rastrigins Function (Problem 9 in [2]). Properties: multi-modal, shifted, separable, scalable, local optimas number is huge.
- Shifted Expanded Griewanks plus Rosenbrocks Function (Problem 13 in [2]). Properties: multi-modal, shifted, non-separable, scalable.
- Non-Continuous Rotated Hybrid Composition Function (Problem 23 in [2]). Properties: multi-modal, non-separable, scalable, a huge number of local optima, different functions properties are mixed together, non-continuous, global optimum is on the bound.

The set of four test problems is sufficient for our purposes since the experiments are considered just a proof-of-concept evaluation of usability of the POEMS framework for the real-parameter optimisation.

V. Experimental Setup

For each problem, 25 runs have been executed. The error value $(f(x) - f(x^*))$, where x^* is the optimal parameter vector and x is a solution obtained by the algorithm, was recorded at termination for each run. Out of the 25 error values for each experiment, the $best$ (smallest), $median$, and the $worst$ (largest) values are presented together with $Mean$ and $StDev$ values.

The results achieved with the POEMS algorithm are compared to the other algorithms presented at the Special Session on Real-Parameter Optimization of the IEEE Congress on Evolutionary Computation 2005. The algorithms denoted as A1-A9 map to the reffered papers as follows – A1 [3], A2 [4], A3 [5], A4 [6], A5 [7], A6 [8], A7 [9], A8 [10], A9 [11].

The following configuration of the EA engaged in POEMS was used in the experiments.

- Population size: 200 (for both 10D and 30D),
- Number of fitness function evaluations: 1500 (for 10D), 2000 (for 30D),
- *MaxGenes*: 10 (for 10D), 30 (for 30D),
- $P_{crossover} = 0.95$, $P_{mutation} = 0.1$, $P_{reuse} = 0.5$.

The total number of fitness function evaluations of the POEMS was set to 100000 (for 10D) and 300000 (for 30D), respectively.

TABLE I

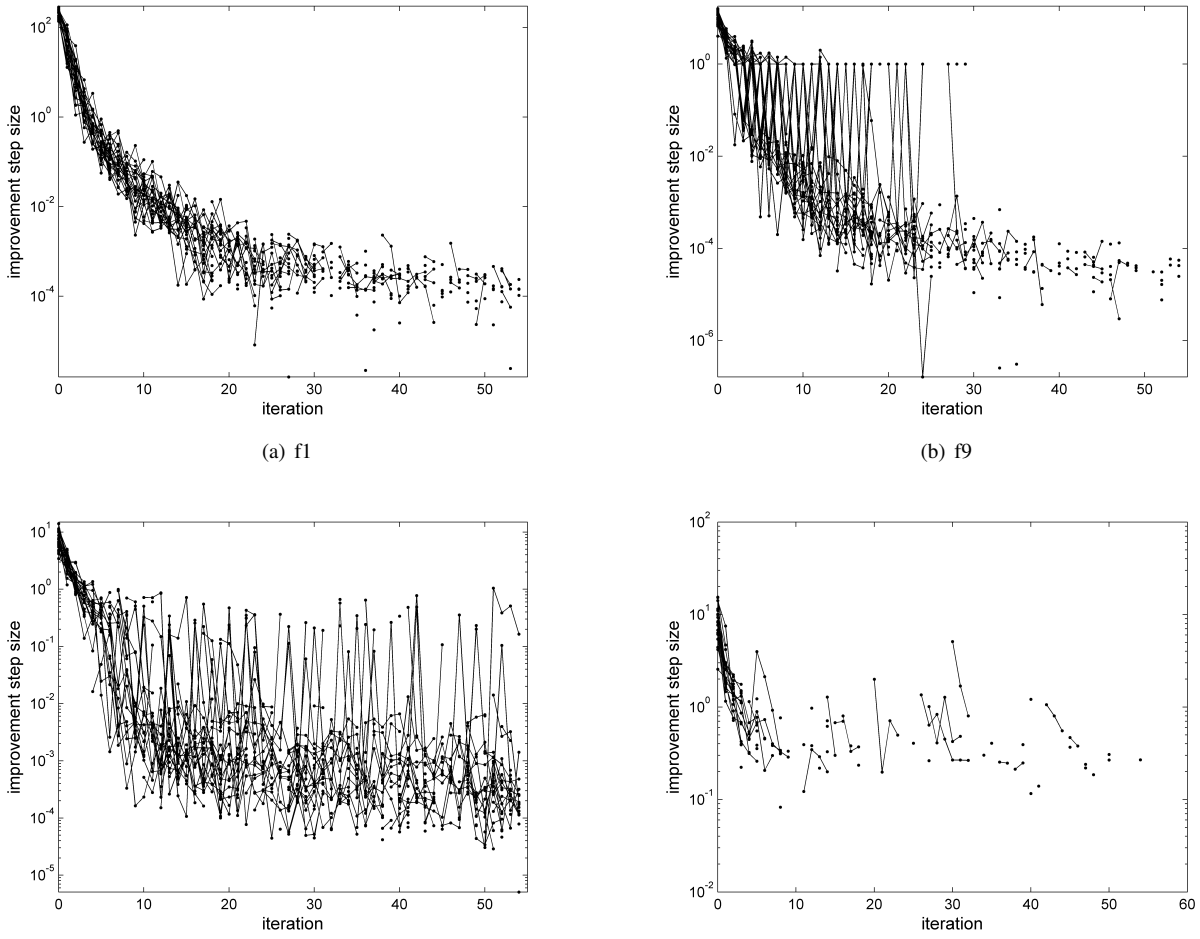
ERROR VALUES FOR 10D PROBLEMS

	Algorithm	POEMS	A1	A2	A3	A4	A5	A6	A7	A8	A9
Problem 1	Best	7.02e-9T	4.82e-9T	6.49e-9T	0.0e+0	3.76e-9T	0.0e+0	0.0e+0	1.84e-9	0.0e+0	4.60e-9T
	Median	6.60e-8	8.49e-9T	8.98e-9T	0.0e+0	8.31e-9T	0.0e+0	0.0e+0	5.65e-9	0.0e+0	9.28e-9T
	Worst	4.62e-7	9.96e-9T	9.99e-9T	0.0e+0	9.89e-9T	0.0e+0	0.0e+0	9.34e-9	0.0e+0	9.93e-9T
	Mean	1.09e-7	8.34e-9	8.90e-9	0.0e+0	8.71e-9	0.0e+0	0.0e+0	5.20e-9	0.0e+0	8.83e-9
	StDev	1.11e-7	1.41e-9	9.39e-10	0.0e+0	1.22e-9	0.0e+0	0.0e+0	1.94e-9	0.0e+0	1.33e-9
Problem 9	Best	2.08e-7	6.75e-9T	9.95e-1	0.0e+0	4.76e-9T	2.28e+0	0.0e+0	1.52e-10	0.0e+0	8.98e+0
	Median	5.57e-7	9.95e-1	3.98e+0	9.95e-1	8.82e-9T	5.83e+0	0.0e+0	6.14e-10	0.0e+0	1.93e+1
	Worst	1.56e-6	3.98e+0	1.19e+1	2.98e+0	2.98e+0	1.21e+1	0.0e+0	9.95e-1	0.0e+0	3.09e+1
	Mean	6.32e-7	1.15e+0	4.02e+0	9.55e-1	1.19e-1	5.42e+0	0.0e+0	2.39e-1	0.0e+0	1.92e+1
	StDev	3.26e-7	7.96e-1	2.27e+0	9.73e-1	5.97e-1	1.91e+0	0.0e+0	4.34e-1	0.0e+0	6.62e+0
Problem 13	Best	9.95e-3	3.67e-1	3.49e-1	1.39e-1	3.28e-1	8.73e-1	2.54e-1	4.07e-1	1.20e-1	4.69e-1
	Median	2.72e-1	7.91e-1	8.18e-1	9.63e-1	7.15e-1	1.89e+0	3.61e-1	6.82e-1	2.17e-1	1.03e+0
	Worst	6.34e-1	1.12e+0	1.32e+0	2.44e+0	1.07e+0	2.47e+0	4.73e-1	1.05e+0	3.12e-1	2.14e+0
	Mean	2.81e-1	7.50e-1	8.38e-1	9.77e-1	6.53e-1	1.84e+0	3.69e-1	6.96e-1	2.20e-1	1.14e+0
	StDev	1.31e-1	2.10e-1	2.69e-1	4.67e-1	2.06e-1	3.40e-1	5.64e-2	1.50e-1	4.11e-2	4.33e-1
Problem 23	Best	5.59e+2	5.59e+2	5.59e+2	5.59e+2	5.59e+2	5.59e+2	5.59e+2	5.59e+2	N/A	4.25e+2
	Median	9.71e+2	5.59e+2	5.59e+2	5.59e+2	1.10e+3	5.59e+2	7.21e+2	5.59e+2	N/A	8.23e+2
	Worst	1.29e+3	1.27e+3	9.71e+2	7.21e+2	1.11e+3	9.71e+2	9.71e+2	5.59e+2	N/A	1.09e+3
	Mean	9.69e+2	6.39e+2	5.76e+2	5.72e+2	1.06e+3	6.41e+2	7.30e+2	5.59e+2	N/A	8.35e+2
	StDev	2.70e+2	2.06e+2	8.22e+1	4.48e+1	1.50e+2	1.39e+2	1.66e+2	3.24e-11	N/A	1.64e+2

TABLE II

ERROR VALUES FOR 30D PROBLEMS

	Algorithm	POEMS	A1	A2	A3	A4	A5	A6	A7	A8	A9
Problem 1	Best	1.00e-2	6.29e-9T	8.58e-9T	0.0e+0	5.11e-9T	0.0e+0	0.0e+0	3.98e-9	0.0e+0	8.29e-9T
	Median	2.65e-2	9.07e-9T	9.62e-9T	0.0e+0	9.18e-9T	0.0e+0	0.0e+0	5.20e-9	0.0e+0	1.23e-2
	Worst	5.28e-2	9.99e-9T	9.93e-9T	0.0e+0	9.96e-9T	5.68e-14	0.0e+0	7.51e-9	0.0e+0	1.18e+1
	Mean	2.65e-2	8.88e-9	9.35e-9	0.0e+0	8.95e-9	9.09e-15	0.0e+0	5.42e-9	0.0e+0	7.97e-1
	StDev	9.10e-3	9.85e-10	4.63e-10	0.0e+0	9.90e-10	2.13e-14	0.0e+0	9.80e-10	0.0e+0	2.49e+0
Problem 9	Best	9.86e-2	6.96e+0	1.09e+1	9.95e+0	8.27e-9T	1.57e+2	1.29e+1	4.35e-6	0.0e+0	9.16e+1
	Median	1.71e-1	1.52e+1	2.39e+1	1.81e+1	1.16e-8	1.80e+2	1.79e+1	9.95e-1	0.0e+0	1.34e+2
	Worst	3.88e-1	2.68e+1	3.68e+1	3.28e+1	9.95e-1	2.03e+2	2.49e+1	4.97e+0	5.68e-14	1.83e+2
	Mean	1.82e-1	1.51e+1	2.39e+1	1.85e+1	2.79e-1	1.79e+2	1.76e+1	9.38e-1	2.27e-15	1.31e+2
	StDev	5.66e-2	5.04e+0	6.25e+0	5.20e+0	4.56e-1	1.02e+1	3.02e+0	1.18e+0	1.14e-14	2.39e+1
Problem 13	Best	1.99e+0	1.87e+0	1.88e+0	1.83e+0	2.27e+0	1.32e+1	1.38e+0	1.10e+0	9.54e-1	5.20e+0
	Median	3.02e+0	3.18e+0	3.40e+0	3.18e+0	1.33e+1	1.55e+1	2.36e+0	2.61e+0	1.18e+0	8.89e+0
	Worst	3.66e+0	1.39e+1	6.42e+0	4.97e+0	1.49e+1	1.67e+1	3.33e+0	3.20e+0	1.46e+0	1.68e+1
	Mean	2.91e+0	5.15e+0	3.59e+0	3.23e+0	1.19e+1	1.53e+1	2.36e+0	2.49e+0	1.21e+0	9.02e+0
	StDev	0.43e+0	4.02e+0	1.09e+0	8.23e-1	3.80e+0	9.52e-1	5.28e-1	5.13e-1	1.34e-1	2.28e+0
Problem 23	Best	5.34e+2	5.34e+2	5.34e+2	5.34e+2	8.65e+2	5.34e+2	N/A	5.34e+2	N/A	6.07e+2
	Median	5.34e+2	5.70e+2	5.34e+2	5.34e+2	8.66e+2	5.34e+2	N/A	5.34e+2	N/A	8.58e+2
	Worst	1.16e+3	9.17e+2	5.34e+2	5.34e+2	8.68e+2	5.34e+2	N/A	5.34e+2	N/A	1.22e+3
	Mean	6.63e+2	5.87e+2	5.34e+2	5.34e+2	8.66e+2	5.34e+2	N/A	5.34e+2	N/A	9.22e+2
	StDev	2.19e+2	7.70e+1	3.49e-4	4.26e-4	8.07e-1	2.71e-4	N/A	2.22e-4	N/A	1.81e+2



(a) f1

(b) f9

(c) f13

(d) f23

Fig. 3. Evolution of the improvement step size

VI. Main Results

Tables I and II compare the results achieved with POEMS to the other algorithms (T in some numbers indicates the optimum was found before reaching the maximum number of evaluations). POEMS exhibits about-average performance among the compared algorithms on both the 10D and 30D problems. The only extremes can be observed for 10D and 30D versions of problem 1. It shows that POEMS is not able to converge quickly to the optimum of an easy unimodal problem. On the other hand, it is competitive to most of the algorithms on the multimodal problems. This may be attributed to the claimed property of POEMS that it is able to evolve improvement steps that do not restrict to the close neighborhood of the current prototype. In principal, the evolved action sequence may move the current prototype far form its current position, so it can escape from the local extreme.

Tables III and IV show fragments of an execution of POEMS on 10D problem 9 and 10D problem 13. For each iteration the following parameters are presented

- $Prototype$ – the current prototype,
- $action\_type$ – the type of the action x_i,
- $parameter$ – the parameter of the respective action,
- $Prototype'$ – the new prototype obtained by application of the evolved action sequence to the $Prototype$.

In fact, the size of the improvement steps tends to decrease during the course of the run. The reason is that in early iterations the prototype is rather of bad fitness so it is easier to evolve some "innovative" action sequence that dramatically modifies the prototype. On the other hand, in latter stages of the run the prototype is already well-fit, so the EA tends to produce action sequences that do not modify the prototype much. In other words, POEMS starts with a global exploration mode at the beginning of the run and converts to the local refinement mode at the latter stages of the run.

The progressive conversion from the global to the local exploration is illustrated in Figure 3. The plots show the evolution of the *improvement step size* of the 25 runs for each problem. The improvement step size is calculated as the Euclidian distance between the original prototype and the new prototype obtained by application of the evolved action sequence to the original one. Isolated points and

TABLE III

A FRAGMENT OF AN EXECUTION OF POEMS ON 10D PROBLEM 9

		x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
1. iter.	*Prototype* (-82.7584)	-1.0300	-1.0400	-2.8300	4.6100	-1.5400	2.5300	-4.4500	-3.0300	-0.1500	1.8500
	action_type	*nop*	2	1	3	4	4	4	2	1	3
	parameter	-0.1755	0.4269	-0.0745	0.6531	0.0867	0.9847	0.7313	-0.3441	0.2158	0.9403
	Prototype′ (-276.1552)	-1.0300	-1.4669	-2.9045	-4.2116	-1.5787	-2.1667	3.9043	-2.6859	0.0658	-4.2312
2. iter.	*Prototype* (-276.1552)	-1.0300	-1.4669	-2.9045	-4.2116	-1.5787	-2.1667	3.9043	-2.6859	0.0658	-4.2312
	action_type	3	1	2	3	3	4	4	4	*nop*	3
	parameter	0.7711	-0.1154	0.0399	0.6001	0.1902	0.7943	0.7665	0.0279	0.1835	0.6121
	Prototype′ (-304.1126)	0.9152	-1.5822	-2.9444	-3.2760	-1.4723	-4.3107	1.9960	-2.6969	0.0658	-3.2452
3. iter.	*Prototype* (-304.1126)	0.9152	-1.5822	-2.9444	-3.2760	-1.4723	-4.3107	1.9960	-2.6969	0.0658	-3.2452
	action_type	4	3	1	3	3	1	*nop*	1	3	4
	parameter	0.0307	-0.0058	-0.0113	0.0669	0.9409	0.0223	0.7863	0.0249	0.1372	-0.0871
	Prototype′ (-320.1851)	0.9030	-1.5822	-2.9557	-3.2474	2.4555	-4.2884	1.9960	-2.6721	0.1340	-3.2452
4. iter.	*Prototype* (-320.1851)	0.9030	-1.5822	-2.9557	-3.2474	2.4555	-4.2884	1.9960	-2.6721	0.1340	-3.2452
	action_type	*nop*	1	3	*nop*	3	1	*nop*	3	4	1
	parameter	0.0318	0.0189	0.6101	0.0575	0.0897	0.0167	0.8699	0.0222	0.1112	0.0003
	Prototype′ (-323.8139)	0.9030	-1.5633	-1.9784	-3.2474	2.4958	-4.2716	1.9960	-2.6634	0.0818	-3.2449
16. iter.	*Prototype* (-329.0046)	0.9057	-1.5644	-0.9788	-2.2535	2.4990	-3.2853	0.9760	-3.6660	0.1000	-3.2466
	action_type	1	3	3	*nop*	*nop*	*nop*	4	*nop*	4	*nop*
	parameter	0.9954	0.0002	0.0013	-0.0265	0.3869	0.0313	-0.1011	0.0129	0.0004	0.0015
	Prototype **(-329.9995)**	**1.9011**	**-1.5643**	**-0.9783**	**-2.2535**	**2.4990**	**-3.2853**	**0.9760**	**-3.6660**	**0.0999**	**-3.2466**
	Optimum **(-330.0000)**	**1.9005**	**-1.5644**	**-0.9788**	**-2.2536**	**2.4990**	**-3.2853**	**0.9759**	**-3.6661**	**0.0985**	**-3.2465**

TABLE IV

A FRAGMENT OF AN EXECUTION OF POEMS ON 10D PROBLEM 13

		x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
1. iter.	*Prototype* (2.1894e+6)	-1.0300	-1.0400	-2.8300	4.6100	-1.5400	2.5300	-4.4500	-3.0300	-0.1500	1.8500
	action_type	*nop*	2	3	4	*nop*	4	3	3	*nop*	4
	parameter	0.1758	0.1686	0.5133	0.9543	-0.1004	0.9554	0.5607	0.9503	-0.1177	0.7605
	Prototype′ (1169.7497)	-1.0300	-1.2086	-2.1964	0.4602	-1.5400	-1.6375	-3.6646	1.0521	-0.1500	-0.0110
2. iter.	*Prototype* (1169.7497)	-1.0300	-1.2086	-2.1964	0.4602	-1.5400	-1.6375	-3.6646	1.0521	-0.1500	-0.0110
	action_type	1	2	3	1	3	1	3	2	2	4
	parameter	0.4064	0.4044	0.6153	-0.9229	0.4527	0.9340	0.7069	0.6620	0.3862	0.5703
	Prototype′ (-122.7340)	-0.6236	-1.6129	-1.1964	-0.4627	-1.0629	-0.7035	-2.1810	0.3901	-0.5362	-0.8308
3. iter.	*Prototype* (-122.7340)	-0.6236	-1.6129	-1.1964	-0.4627	-1.0629	-0.7035	-2.1810	0.3901	-0.5362	-0.8308
	action_type	2	2	1	1	1	1	3	4	2	2
	parameter	0.0313	0.4599	0.2641	0.1716	0.5156	0.5527	0.6991	0.5373	0.3387	0.3481
	Prototype′ (-128.5585)	-0.6549	-2.0728	-0.9323	-0.2911	-0.5473	-0.1508	-0.7455	-0.3168	-0.8749	-1.1789
4. iter.	*Prototype* (-128.5585)	-0.6549	-2.0728	-0.9323	-0.2911	-0.5473	-0.1508	-0.7455	-0.3168	-0.8749	-1.1789
	action_type	2	1	2	1	1	4	1	4	4	2
	parameter	-0.1350	0.5125	-0.8127	0.1527	0.2341	0.3964	-0.4143	0.5000	0.3634	0.2476
	Prototype′ (-129.1438)	-0.5199	-1.5604	-0.1196	-0.1384	-0.3132	-0.5129	-1.1598	-0.9127	-1.1807	-1.4265
52. iter.	*Prototype* (-129.822420)	-0.51151	-1.5526	-0.1083	-0.3832	-0.4325	-0.6219	-0.9807	-0.9237	-1.1727	-1.6543
	action_type	3	*nop*	*nop*	*nop*	*nop*	*nop*	*nop*	*nop*	*nop*	*nop*
	parameter	0.0001	0.0003	0.0353	0.0002	-0.3478	0.0012	0.3546	0.4182	0.7918	0.0027
	Prototype′ (-129.822421)	-0.51148	-1.5526	-0.1083	-0.3832	-0.4325	-0.6219	-0.9807	-0.9237	-1.1727	-1.6543
	Optimum (-130.0000)	-0.7529	-1.8497	-0.4371	-0.7327	-0.3126	-0.8605	-1.2764	-0.5828	-1.5007	-1.5204

discontinuities are there due to the fact that not in every iteration the action sequence that would improve the current prototype was found by the EA. A common tendency of the improvement step size to decrease (non monotonously) as the time progresses can be observed for all the problems. However, the step size can jump high even in latter stages as can be observed on multimodal problems in Figures 3(b)-3(d). The peaks indicate the iterations, in which the prototype probably moved from the basin of one local extreme to the other one. Very sparse plot in Figure 3(d) reveal the fact the problem 23 is hard for the POEMS since the iterations where any improving action sequence was evolved were rather rare.

The tables show the first four iterations of the runs. In case of the problem 9 the 16th iteration is shown, in which the solution of the required accuracy was found. In case of the problem 13 the 52nd iteration is the last iteration of the run, in which the improving action sequence was found. A common characteristic observed in both run fragments is that the number of active actions in the latter action sequences decreases as the time progresses. Note, that the proportion of active and nop actions in the starting population of each EAs is half and half. Thus, the EA prefers more active actions in early iterations whilst it suppresses the active actions in the latter stages of the run. This is in agreement with what we have observed on the evolution of the improvement step size that the bigger changes of the prototype are frequent in the beginning of the run whilst rather small changes are applied in the final stages of the run.

VII. Conclusions

We have proposed the implementation of the algorithm called *Iterative Prototype Optimisation with Evolved Improvement Steps* (POEMS) for the real-parameter optimisation. POEMS iteratively improves the prototype solution so that in each iteration an evolutionary algorithm is used to search for a sequence of actions, which would improve the current prototype. The POEMS algorithm was tested on problems collected for the Special Session on Real-Parameter Optimization of the IEEE Congress on Evolutionary Computation 2005 and the results were compared to the results of the algorithms presented there. The achieved results show a potential of the presented optimisation framework for solving hard real-parameter optimisation problems.

An interesting aspect of the POEMS approach is that it functions as a global explorer in early stages of the run whilst it converts to fine tuning of the prototype solution later on. This is due to the fact, that as the prototype gets better and better, it becomes very hard for EA to evolve an action sequence that would both improve and considerably change the prototype. Instead, trivial action sequences that do not modify the prototype much are produced. Obviously, the algorithm would perform much better if it was able to keep the exploration capabilities during the whole run.

One of the possible extensions of the algorithm might utilize a backtracking strategy to restart the algorithm from a *new prototype* when it begins to stagnate. An archive of candidate prototypes (i.e. solutions of good fitness that considerably differ from the current prototype) would be maintained along the run. If the algorithm was not able to generate any improving action sequence for the current prototype within a specified number of iterations then the new prototype, most distinct from the current one, would be picked from the archive. Obviously, this could work only if the EA is able to produce a population of high-quality and distinct solutions otherwise the algorithm would restart with a new prototype similar to the current one. In order to support the diversity of the evolved population some kind of niching strategy can be used. Another way to improve the algorithm would be to design the set of actions and the genetic operators that would extend the exploration capabilities of the EA. Investigation of the possibilities to make the POEMS algorithm resistant against stagnation will be the content of the future research.

Acknowledgments. The research has been supported by the research program No. MSM6840770012 "Transdisciplinary Research in the Area of Biomedical Engineering II" of the CTU in Prague, sponsored by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] J. Kubalík and J. Faigl, "Iterative Prototype Optimisation with Evolved Improvement Steps", P. Collet, M. Tomassini, M. Ebner, S. Gustafson and A. Ekart (Eds.), *Proc. Genetic Programming, 9th European Conference, EuroGP 2006*, LNCS 3905, Springer 2006, ISBN 3-540-33143-3, pp. 154–165.

[2] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 Special Session on Real-Parameter Optimization, tech. rep., Nanyang Technological University, Singapore, May 2005. http://www.ntu.edu.sg/home/epnsugan.

[3] C. Garca-Martnez and M. Lozano, "Hybrid Real-Coded Genetic Algorithms with Female and Male Differentiation", *Proc. IEEE Congress on Evolutionary Computation*, Edinburgh, UK, Sept. 2005, volume 2, pp. 896–903.

[4] Pedro J. Ballester, J. Stephenson, J. N. Carter, and K. Gallagher, "Real-Parameter Optimization Performance Study on the CEC-2005 benchmark with SPC-PNX", *Proc. IEEE Congress on Evolutionary Computation*, Edinburgh, UK, Sept. 2005, volume 1, pp. 498–505.

[5] J. Rnkknen, S. Kukkonen, and K. V. Price, "Real-Parameter Optimization with Differential Evolution", *Proc. IEEE Congress on Evolutionary Computation*, Edinburgh, UK, Sept. 2005, volume 1, pp. 506–513.

[6] A. Sinha, S. Tiwari, and K. Deb, "A Population-Based, Steady-State Procedure for Real-Parameter Optimization", *Proc. IEEE Congress on Evolutionary Computation*, Edinburgh, UK, Sept. 2005, volume 1, pp. 514–521.

[7] B. Yuan and M. Gallagher, "Experimental Results for the Special Session on Real-Parameter Optimization at CEC 2005: A Simple, Continuous EDA", *Proc. IEEE Congress on Evolutionary Computation*, Edinburgh, UK, Sept. 2005, volume 2, pp. 1792–1799.

[8] J. J. Liang and P. N. Suganthan, "Dynamic Multi-Swarm Particle Swarm Optimizer with Local Search", *Proc. IEEE Congress on Evolutionary Computation*, Edinburgh, UK, Sept. 2005, volume 1, pp. 522–528.

[9] A. Auger and N. Hansen, "A Restart CMA Evolution Strategy With Increasing Population Size", *Proc. IEEE Congress on Evolutionary Computation*, Edinburgh, UK, Sept. 2005, volume 2, pp. 1769–1776.

[10] A. K. Qin and P. N. Suganthan, "Self-adaptive Differential Evolution Algorithm for Numerical Optimization", *Proc. IEEE Congress on Evolutionary Computation*, Edinburgh, UK, Sept. 2005, volume 2, pp. 1785–1791.

[11] P. Pošík, "Real-Parameter Optimization Using the Mutation Step Co-evolution", *Proc. IEEE Congress on Evolutionary Computation*, Edinburgh, UK, Sept. 2005, volume 1, pp. 872–879.