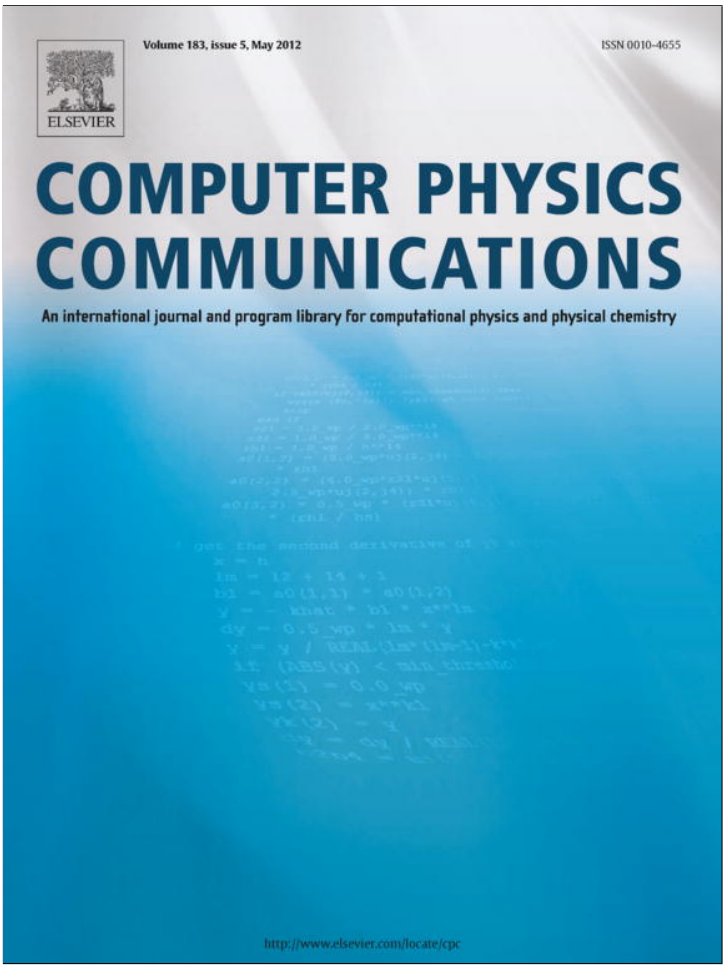
Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.



This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright

Author's personal copy

Computer Physics Communications 183 (2012) 1139–1154

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

MEMPSODE: A global optimization software based on hybridization of population-based algorithms and local searches ☆

C. Voglis [a,*], K.E. Parsopoulos [a], D.G. Papageorgiou [b], I.E. Lagaris [a], M.N. Vrahatis [c]

[a] Department of Computer Science, University of Ioannina, P.O. BOX 1186, GR-45110 Ioannina, Greece
[b] Department of Materials Science and Engineering, University of Ioannina, P.O. BOX 1186, GR-45110 Ioannina, Greece
[c] Department of Mathematics, University of Patras, GR-26110 Patras, Greece

ARTICLE INFO

Article history:
Received 4 April 2011
Received in revised form 5 January 2012
Accepted 6 January 2012
Available online 12 January 2012

Keywords:
Global optimization
Particle swarm optimization
Differential evolution
Memetic algorithms
Local search
Merlin optimization environment

ABSTRACT

We present MEMPSODE, a global optimization software tool that integrates two prominent population-based stochastic algorithms, namely Particle Swarm Optimization and Differential Evolution, with well established efficient local search procedures made available via the Merlin optimization environment. The resulting hybrid algorithms, also referred to as Memetic Algorithms, combine the space exploration advantage of their global part with the efficiency asset of the local search, and as expected they have displayed a highly efficient behavior in solving diverse optimization problems. The proposed software is carefully parametrized so as to offer complete control to fully exploit the algorithmic virtues. It is accompanied by comprehensive examples and a large set of widely used test functions, including tough atomic cluster and protein conformation problems.

Program summary

Program title: MEMPSODE (MEMetic Particle Swarm Optimization and Differential Evolution)
Catalogue identifier: AELM_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AELM_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: Standard CPC license, http://cpc.cs.qub.ac.uk/licence/licence.html
No. of lines in distributed program, including test data, etc.: 14 877
No. of bytes in distributed program, including test data, etc.: 592 244
Distribution format: tar.gz
Programming language: ANSI C, ANSI Fortran-77
Computer: Workstations
Operating system: Developed under the Linux operating system using the GNU compilers v.4.4.3. It has also been tested under Solaris and the Cygwin environment.
RAM: The code uses $O(n \times N)$ internal storage, n being the dimension of the problem and N the maximum population size. The required memory is dynamically allocated.
Word size: 64 bits
Classification: 4.9
Subprograms used:

Cat Id	Title	Reference
AAXW_v4_0	MERLIN-3.1.1	CPC 159 (2004) 70

Nature of problem: Optimization is a valuable mathematical tool for solving a plethora of scientific and engineering problems. Usually, the underlying problems are modeled with objective functions whose minimizers (or maximizers) correspond to the desired solutions of the original problem. In many cases, there is a multitude of such minimizers that correspond to solutions either locally, i.e., in their close neighborhood, or globally, i.e., with respect to the whole search space. There is a significant number of efficient algorithms for addressing optimization problems. One can distinguish two main categories, based on their adequacy in performing better global (exploration) or local (exploitation) search. Standard local optimization algorithms have the ability to rapidly converge towards local minimizers but they are

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: voglis@cs.uoi.gr (C. Voglis).

0010-4655/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2012.01.010

Author's personal copy

also prone to get easily trapped in their vicinity. These algorithms usually exploit local information of the objective function, including first- and second-order derivatives. On the other hand, global optimization algorithms are designed to perform better exploration, although at the cost of questionable convergence properties. Typically, these approaches integrate stochastic operations. The form of the optimization problem at hand plays a crucial role in the selection of the most appropriate algorithm. Objective functions that lack nice mathematical properties (such as differentiability, continuity etc.) may raise applicability issues for algorithms that require derivatives. On the other hand, applications that require high accuracy may be laborious for stochastic algorithms. The existence of a multitude of local and/or global minimizers can render these problems even harder for any single optimization algorithm.

Solution method: Evolutionary Algorithms and Swarm Intelligence approaches have been established as effective global optimization algorithms that make minor assumptions about the objective function. Particle Swarm Optimization (PSO) and Differential Evolution (DE) possess a salient position among the most successful algorithms of these categories. Numerous studies indicate that their performance can be radically improved when combined with efficient local optimization schemes. The resulting hybrid algorithms offer more balanced search intensification/diversification than the original ones, thereby increasing both their efficiency and effectiveness. Such hybrid schemes are called Memetic Algorithms, and they have gained a rapidly growing interest over the past few years.

We present MEMPSODE (MEMetic, PSO and DE), a global optimization software that implements memetic PSO and DE within a unified framework. The software utilizes local search procedures from the established Merlin optimization environment. The performance of the implemented approaches is illustrated on several examples, including hard optimization tasks such as atomic cluster and protein conformation problems.

Restrictions: The current version of the software uses double precision arithmetic. However, it can be easily adapted by the user to handle integer or mixed-integer problems.

Unusual features: The software takes into account only bound constraints. General constraints may be tackled by user-defined penalty or barrier functions that can be easily incorporated in the source code of the objective function.

Additional comments: The use of the Merlin Optimization Environment 3.1.1 (see subprograms above) is optional.

A comprehensive user manual is provided that covers in detail the installation procedure and provides detailed examples of operation.

Running time: The running time depends solely on the complexity of the objective function (and its derivatives, if used) as well as on the available computational budget (number of function evaluations). The test run provided (Rastrigin function $n = 10$), requires 2.5×10^6 function evaluations (2.8 seconds on an i7-920 CPU).

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

Many scientific and engineering problems may be reformulated as optimization problems. This is possible by developing appropriate objective functions that accurately model the problem at hand. Usually, the global minimizers of the objective functions correspond to optimal solutions of the original problem. There are numerous applications that require the solution of global optimization problems as intermediate tasks. Among others, we mention applications related to signal processing, telecommunications, finance and operations research, networks and transportation, engineering design and control, molecular biology, hardware and software design, as well as biomedical engineering [1–5].

Global Optimization (GO) problems that involve objective functions with strong mathematical properties (such as differentiability, Lipschitz continuity etc.) and adequate simplicity (e.g., functions with a unique minimizer) can be addressed efficiently by classical deterministic optimization algorithms. Usually, these approaches use first- and second-order derivative information of the objective function and they are characterized by high rates of convergence [6,7].

However, in most realistic cases the relevant objective functions do not share these smoothness properties. Indeed, the nature of the physical problem itself can pose modeling limitations that prohibit the development of mathematically well-behaved objective functions with desirable characteristics. Multitude of local minimizers, noisy or black-box objective functions, inexact measurements due to equipment failure, are just a few of the issues that must be addressed in modeling real-world problems. In such situations, more tolerant algorithms than the traditional gradient-based approaches should be considered.

Over the past 20 years, Evolutionary Algorithms (EAs) and Swarm Intelligence (SI) approaches have been established as powerful optimization tools for solving optimization problems [8–15]. These algorithms are based on models that draw their inspiration from physical systems. Based on natural selection, DNA structures exhibit a remarkable capability of evolving, producing populations with fitter individuals. Animal herds and fish schools exhibit striking self-organization and collective behavior, based solely on primitive interactions among them. The inherent optimization capabilities of such systems can be modeled using stochastic analysis and discrete dynamical system models, producing new population-based algorithms with significant tolerance in uncertain environments and minimal demands regarding the objective function properties, which in turn creates obstacles to the mathematical analysis of their convergence properties and proper parameter settings. Nevertheless, despite the occasional criticism, EAs and SI approaches have been used in a vast number of applications, offering solutions in problems where other methods usually fail [9,15].

Genetic Algorithms (GAs) [16] where the first EAs used in engineering applications. The binary population representation that was used in the early versions, rendered them attractive for engineering design problems. Later, Evolution Strategies [17] gained popularity due to their ability to solve continuous optimization problems and adapt their parameters during the optimization procedure. The development of Particle Swarm Optimization (PSO) [18] and Differential Evolution (DE) [19] started in the mid nineties. During the first years, the

Author's personal copy

new approaches were shadowed by the established EAs. However, their popularity has been gradually increased due to implementation simplicity that made them accessible to researchers in diverse fields, as well as due to the increased efficiency of recent variants. Today, PSO and DE have been distinguished as two of the most promising population-based approaches.

The rise of EAs and SI algorithms has sparked the development of a closely related category, namely the *Memetic Algorithms* (MAs). MAs constitute a class of hybrid meta-heuristics that combine population-based optimization algorithms with local search procedures [20–22]. The rationale behind their development was the necessity for powerful algorithms where the global exploration capability of EAs and SI approaches would be complemented with the efficiency and accuracy of classical local optimization techniques.

The first implementations of MAs were hybrid algorithms that employed GAs as their global search module and a local search (LS) procedure applied on some individuals at each iteration [23–26]. The resulted GA-LS algorithms have evolved mainly in two classes depending on the employed LS type. The first class hybridizes the GA with local optimization methods such as stochastic hill-climbing, the nonlinear simplex method, conjugate gradient and quasi-Newton methods. In the other class, LS is performed by using specialized crossover operators for local refinement. The first promising results were obtained from the approximate solution of NP-hard optimization problems, where the cooperation of the global and local search modules was observed to be advantageous. [14,27–29].

The successful application of MAs in the discrete domain was followed by a rapid development of continuous optimization variants. Very effective MAs that exhibit outstanding performance both on standard continuous global optimization benchmarks as well as on real-world problems have been developed [30–34]. Recently, PSO and DE have been also used as the global search components of efficient MA schemes [12,34–37].

Since MAs contain a part based upon heuristics, their performance depends on configuration parameters that are determined experimentally. The most important and still open questions, known as the *fundamental memetic questions* [15], are:

(a) *Where* should the LS procedures be applied. The user should pick the individuals that will constitute initial points for the LS.
(b) *When* should the LS procedures be applied. The application frequency of LS shall be determined in order to attain a wise exploitation of the available computational budget.
(c) *How much of the budget* should the LS procedures spend. The user must specify the fraction of the available computational budget that will be devoted to LS.

All these issues were studied in [36], resulting in a very efficient PSO-based memetic strategy.

In the paper at hand, we propose a software that follows closely the PSO-based memetic approaches reported in [36] and extends them also to the DE framework. More specifically, the Unified PSO (UPSO) approach [38,39], which harnesses the strengths of standard local and global PSO variants is implemented. The standard PSO is derived directly as a special case of UPSO. The five fundamental operators of the DE algorithm are implemented as well. In both cases, direct calls to LS procedures are facilitated via the established *Merlin* optimization environment [40,41], providing the ability to develop a variety of MAs. Of course, the user may simply run the UPSO and DE code without employing LS, if not desirable. It must be stressed that no single algorithm or combination of algorithms can address efficiently all possible optimization problems. This is not only intuitively expected but there is significant theoretical evidence suggesting that there is no panacea in optimization [42,43]. This justifies the inclusion of different algorithms in our software.

MEMPSODE is a contemporary software package with state-of-the-art algorithmic components that can serve as an effective general purpose continuous GO tool. MEMPSODE is fully parametrized, enabling thus the user to easily configure the algorithms, taking in account the decisions made as far as the fundamental memetic questions are considered. It may be used either as a standalone executable or as a user-callable routine, thus enabling integration with other software packages. In addition, the source code is designed to easily accept additions of external source code without requiring specialized expertise in programming. Alternatively, the user can simply run the software using the proposed default parameters without any further intervention. The proposed software treats both smooth and non-smooth objectives functions. It includes an arsenal of gradient-based and gradient-free methods provided by Merlin environment and an extension to user provided local search algorithms via the Merlin plug-in mechanism. It also provides a unique interface to the Tinker software [44] for molecular mechanics calculations, thus enabling its application on numerous force fields and molecules.

The rest of the paper is organized as follows: the employed UPSO and DE algorithms are sketched in Section 2, along with a brief description of the Merlin environment and its supported LS procedures. Section 3 is devoted to the description of the most important algorithmic issues related to the proposed software, while Section 4 offers a detailed documentation of the software, including installation, execution and application examples. Finally, we present concluding remarks on this work in Section 6.

2. Background information

In the following sections, we present in brief the three major algorithmic components of the proposed software, namely UPSO, DE and the LS optimization artillery provided by the Merlin optimization environment. Without loss of generality, we will assume that all the considered optimization problems refer to the minimization case. Maximization can be equivalently treated as minimization by flipping the sign of the objective function.

2.1. Unified particle swarm optimization

PSO was introduced by Eberhart and Kennedy [18,45]. The main concept of the method includes a population, also called *swarm*, of search points, also called *particles*, searching for optimal solutions within the search space, simultaneously. The particles move in the search space by assuming an adaptable position shift, called *velocity*, at each iteration.

Moreover, each particle retains in a memory the best position it has ever visited, i.e., the position with the lowest function value. This information can be regarded as the particle's experience and it is communicated to other particles. For this purpose, each particle is assigned a *neighborhood*, which determines the indices of its mates that will share its experience. Intuitively, a neighborhood can be depicted as a graph of nodes that represent the particles and edges that represent their communication links. This is often called the *neighborhood's topology*. Fig. 1 illustrates two such neighborhood topologies, namely the *ring* (left) and the *star* (right).

Author's personal copy

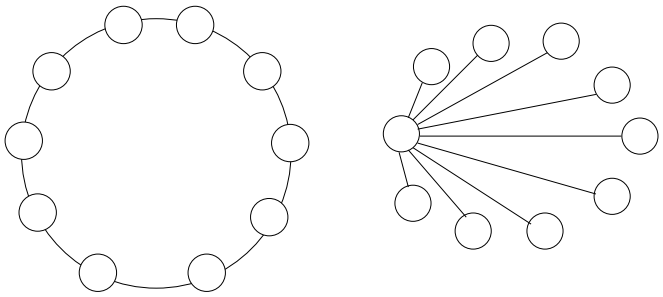


Fig. 1. The *ring* (left) and the *star* (right) neighborhood topology of PSO.

Obviously, the neighborhood scheme affects the flow of information among the particles. For example, in the popular ring topology each particle is assumed to communicate only with its mates with adjacent indices. Such schemes correspond the local PSO variant, usually called *lbest* PSO. Here, locality refers only to the information exchange scheme and should not be confused with local search. It is easily inferred that the communication restrictions of lbest PSO, result in slower flow of information among the particles.

If the whole swarm constitutes a single neighborhood, any new information (best position) is immediately advertised to every single particle in each iteration. This is also called the global (or *gbest*) PSO model. Taking into consideration that the shared experience of a particle is one of the two crucial factors that govern its motion in the search space, we understand that the neighborhood topology can have a significant impact on PSO's convergence speed. In practice, it has been noticed that lbest PSO has slower convergence, but it retains a thorough exploration capability. On the other hand, the gbest PSO model exhibits faster convergence however at the cost of possible entrapment in the vicinity of local minimizers.

Putting our description in a mathematical framework, let us assume the n-dimensional continuous optimization problem:

$$\min_{x\in X\subset\mathbb{R}^n} f(x), \tag{1}$$

where the search space X is an orthogonal hyperbox in $\mathbb{R}^n$:

$$X \equiv [l_1, r_1] \times [l_2, r_2] \times \cdots \times [l_n, r_n].$$

A swarm of N particles is a set of search points:

$$S = \{x_1, x_2, \ldots, x_N\},$$

where the i-th particle is defined as:

$$x_i = (x_{i1}, x_{i2}, \ldots, x_{in})^\top \in X, \quad i = 1, 2, \ldots, N.$$

The velocity (position shift) of x_i is denoted as:

$$v_i = (v_{i1}, v_{i2}, \ldots, v_{in})^\top, \quad i = 1, 2, \ldots, N,$$

and its best position as:

$$p_i = (p_{i1}, p_{i2}, \ldots, p_{in})^\top \in X, \quad i = 1, 2, \ldots, N.$$

In the proposed software we implemented the popular ring neighborhood topology. If m denotes the neighborhood's *radius*, then the neighborhood of x_i consists of the indices of those particles that share information with x_i and it is denoted as:

$$\mathcal{N}_i = \{i - m, \ldots, i - 1, i, i + 1, \ldots, i + m\}.$$

Obviously, in the ring topology the two ends of the ring coincide, i.e., the indices recycle at the end.

Let g_i denote the best particle in $\mathcal{N}_i$, i.e.:

$$g_i = \arg\min_{j\in\mathcal{N}_i} f(p_j),$$

and t denote the algorithm's iteration counter. Then, the particle positions and velocities are updated at each iteration according to the equations [46]:

$$v_{ij}^{(t+1)} = \chi\left[v_{ij}^{(t)} + c_1 r_1\left(p_{ij}^{(t)} - x_{ij}^{(t)}\right) + c_2 r_2\left(p_{g_i j}^{(t)} - x_{ij}^{(t)}\right)\right], \tag{2}$$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + v_{ij}^{(t+1)}, \quad i = 1, 2, \ldots, N, \ j = 1, 2, \ldots, n, \tag{3}$$

where χ is the *constriction coefficient*; c_1 and c_2 are positive constants called *cognitive* and *social* parameter, respectively; and r_1, r_2, are random numbers drawn from a uniform distribution in the range $[0, 1]$. The PSO model defined in Eqs. (2) and (3) was proposed by Clerc and Kennedy in [46] and it is considered as one of the most promising state-of-the-art PSO variants and therefore we have adopted this model in our implementation.

The best position of each particle is updated at each iteration as follows:

Author's personal copy

$$p_{ij}^{(t+1)} = \begin{cases} x_{ij}^{(t+1)}, & \text{if } f(x_{ij}^{(t+1)}) < f(p_{ij}^{(t)}), \\ p_{ij}^{(t)}, & \text{otherwise.} \end{cases} \tag{4}$$

Clerc and Kennedy [46] provided also a stability analysis of the PSO model described above. According to it, the parameters shall be set in proper values such that:

$$\chi = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|}, \tag{5}$$

for $\varphi = c_1 + c_2 > 4$. Thus, the values:

$$\chi = 0.729, \qquad c_1 = c_2 = 2.05,$$

were proposed as the default parameter set. Alternative setups were proposed by Trelea in [47]. The swarm and velocities are usually initialized randomly and uniformly within the search space.

The described PSO model was generalized in the UPSO scheme by combining the lbest and gbest velocity updates. The motivation behind it, stemmed from the speculation that harnessing search directions with different exploration/exploitation properties could lead to more efficient schemes. Thus, if we assume that $G_i^{(t+1)}$ and $L_i^{(t+1)}$ denote the velocity update of x_i in the gbest and lbest PSO model, respectively [38,39]:

$$G_{ij}^{(t+1)} = \chi\left[v_{ij}^{(t)} + c_1 r_1\left(p_{ij}^{(t)} - x_{ij}^{(t)}\right) + c_2 r_2\left(p_{gj}^{(t)} - x_{ij}^{(t)}\right)\right], \tag{6}$$

$$L_{ij}^{(t+1)} = \chi\left[v_{ij}^{(t)} + c_1 r_1\left(p_{ij}^{(t)} - x_{ij}^{(t)}\right) + c_2 r_2\left(p_{g_i j}^{(t)} - x_{ij}^{(t)}\right)\right], \tag{7}$$

where g is the index of the overall best particle, i.e.:

$$g = \arg \min_{j=1,\ldots,N} f(p_j),$$

then the particle is updated as follows [38,39]:

$$U_{ij}^{(t+1)} = uG_{ij}^{(t+1)} + (1-u)L_{ij}^{(t+1)}, \tag{8}$$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + U_{ij}^{(t+1)}, \quad i = 1, 2, \ldots, N,\ j = 1, 2, \ldots, n. \tag{9}$$

The parameter $u \in [0, 1]$ is called the *unification factor* and it balances the influence (trade-off) of the global and local velocity update. Obviously, the lbest PSO model is retrieved for $u = 0$, while for $u = 1$ the gbest PSO model is obtained. All intermediate values produce combinations with diverse convergence properties.

In addition to the standard UPSO model, a mutated one was also proposed [38,39]. According to it, a stochastic parameter is incorporated in UPSO's velocity update, imitating the mutation operation in EAs. Thus, Eq. (8) can be redefined as:

$$U_{ij}^{(t+1)} = ur_3 G_{ij}^{(t+1)} + (1-u)L_{ij}^{(t+1)}, \tag{10}$$

which is mostly based on the lbest term, or as:

$$U_{ij}^{(t+1)} = uG_{ij}^{(t+1)} + (1-u)r_3 L_{ij}^{(t+1)}, \tag{11}$$

which favors the gbest term. The parameter $r_3 \sim N(\mu, \sigma^2 I)$, is a normally distributed random variable and I stands for the identity matrix. Convergence in probability for these variants was studied in [38,39]. Experiments on a set of diverse problems, suggested that UPSO can provide remarkably better results than the standard lbest or gbest PSO models alone [38,39,48,49].

2.2. Memetic particle swarm optimization

As already mentioned, MAs combine EAs and LS procedures producing hybrid schemes that exploit the advantages of both approaches. In [36] a Memetic PSO (MPSO) was introduced and theoretically analyzed. The proposed MPSO was a combination of the standard PSO with the Random Walk with Direction Exploitation [50] algorithm. The reported results, including unconstrained, constrained, minimax and integer programming problems, suggested that MPSO can be considerably more efficient than the standard PSO. Later, this was also confirmed in fixed points and machine learning problems [51,52].

The design of MPSO in [36] was based on three fundamental schemes, henceforth called the *memetic strategies*:

Scheme 1: LS is applied only on the overall best position, p_g, of the swarm.
Scheme 2: LS is applied on each locally best position, p_i, $i = 1, 2, \ldots, N$, with a prescribed fixed probability, $\rho \in (0, 1]$.
Scheme 3: LS is applied both on the best position, p_g, as well as on some randomly selected locally best positions, p_i, $i \in \{1, 2, \ldots, N\}$.

These schemes can be applied either at each iteration or whenever a specific number of consecutive iterations has been completed.

Of course, many other memetic strategies can be considered. For instance, a simple one would be the application of LS on every particle. However, such an approach would be costly in terms of function evaluations. In practice, only a small number of particles are considered as start points for LS, as pointed out in [25]. The memetic strategies proposed in [36] were also adopted in MEMPSODE for both PSO- and DE-based MAs. Further technical details are reported in Section 3.

Author's personal copy

2.3. Differential evolution

The Differential Evolution (DE) algorithm was introduced by Storn and Price [19,53] as a population-based stochastic optimization algorithm for numerical optimization problems. DE is formulated similarly to PSO. A population:

$$P = \{x_1, x_2, \ldots, x_N\},$$

of N individuals is utilized to probe the search space, $X \subset \mathbb{R}^n$. The population is randomly initialized, usually following a uniform distribution within the search space.

Each individual is an n-dimensional vector:

$$x_i = (x_{i1}, x_{i2}, \ldots, x_{in})^\top \in X, \quad i = 1, 2, \ldots, N,$$

serving as a candidate solution of the problem at hand. The population is iteratively evolved by applying two operators, *mutation* and *recombination*, on each individual to produce new candidate solutions. Then, the new and the old individuals are merged and selection takes place to construct the new population consisting of the N best individuals. The procedure continues in the same manner until a termination criterion is fulfilled.

2.3.1. Mutation

The mutation operator produces a new vector, v_i, for each individual, x_i, $i = 1, 2, \ldots N$, by combining some of the rest individuals of the population. There are five basic operators proposed to accomplish this task:

$$\text{OP1:} \quad v_i^{(t+1)} = x_g^{(t)} + F\left(x_{r_1}^{(t)} - x_{r_2}^{(t)}\right), \tag{12}$$

$$\text{OP2:} \quad v_i^{(t+1)} = x_{r_1}^{(t)} + F\left(x_{r_2}^{(t)} - x_{r_3}^{(t)}\right), \tag{13}$$

$$\text{OP3:} \quad v_i^{(t+1)} = x_i^{(t)} + F\left(x_g^{(t)} - x_i^{(t)} + x_{r_1}^{(t)} - x_{r_2}^{(t)}\right), \tag{14}$$

$$\text{OP4:} \quad v_i^{(t+1)} = x_g^{(t)} + F\left(x_{r_1}^{(t)} - x_{r_2}^{(t)} + x_{r_3}^{(t)} - x_{r_4}^{(t)}\right), \tag{15}$$

$$\text{OP5:} \quad v_i^{(t+1)} = x_{r_1}^{(t)} + F\left(x_{r_2}^{(t)} - x_{r_3}^{(t)} + x_{r_4}^{(t)} - x_{r_5}^{(t)}\right), \tag{16}$$

where t denotes the iteration counter; $F \in (0, 1]$ is a fixed user-defined parameter; g denotes the index of the best individual in the population, i.e., the one with the lowest function value; and $r_j \in \{1, 2, \ldots, N\}$, $j = 1, 2, \ldots, 5$, are mutually different randomly selected indices that differ also from the index i. Thus, in order to be able to apply all mutation operators, it must hold that $N > 5$. All vector operations in Eqs. (12)–(16) are performed componentwise. The operators that use the best individual in the population can be considered as operators with higher exploitation property while the others can be considered as operators with higher exploration property.

2.3.2. Recombination

After the mutation, a recombination operator is applied producing a trial vector:

$$u_i = (u_{i1}, u_{i2}, \ldots, u_{in}), \quad i = 1, 2, \ldots, N,$$

for each individual. This vector is defined as follows:

$$u_{ij}^{(t+1)} = \begin{cases} v_{ij}^{(t+1)}, & \text{if } R_j \leqslant CR \text{ or } j = \mathrm{RI}(i), \\ x_{ij}^{(t)}, & \text{if } R_j > CR \text{ and } j \neq \mathrm{RI}(i), \end{cases} \tag{17}$$

where $j = 1, 2, \ldots, n$; R_j is a random variable uniformly distributed in the range $[0, 1]$; $CR \in [0, 1]$ is a user-defined crossover constant; and $\mathrm{RI}(i) \in \{1, 2, \ldots, n\}$, is a randomly selected index.

Finally, each trial vector is compared against the corresponding individual and the best between them comprise the new individual in the next generation, i.e.:

$$x_i^{(t+1)} = \begin{cases} u_i^{(t+1)}, & \text{if } f(u_i^{(t+1)}) < f(x_i^{(t)}), \\ x_i^{(t)}, & \text{otherwise.} \end{cases} \tag{18}$$

Compared to PSO, DE appears to be a greedier algorithm since always the population consists of the best discovered individuals and search directions are always produced through their combinations. Also, special care shall be taken for the determination of the parameters F and CR. Experimental results suggest a significant sensitivity of DE on their values.

2.4. Bound constrained nonlinear local optimization

A local solution to the problem in Eq. (1) is obtained by applying bound (or box) constrained nonlinear local optimization methods. The hybrid schemes implemented in MEMPSODE have a need for deterministic local search procedures that require a starting point, $x_0 \in X$, and generate a sequence of points, $\{x_k\}_{k=0}^{\infty}$ in X, in order to determine a minimizer within a prescribed accuracy.

The generation of a new point, x_{k+1}, in the sequence is based on information collected for the current iterate, x_k. Typically, this information includes the function value at x_k, as well as the first- and probably second-order derivatives of $f(x)$ at x_k. In all cases, the aim is to find a new iterate with lower function value than the current one.

There is a large number of LS algorithms proposed in the literature. The description of these methods is beyond the scope of the current paper. However, there is a plethora of comprehensive sources where the interested reader can find further information [6,7,54].

In the proposed software, we employed only unconstrained LS algorithms. The constrained case can be tackled as unconstrained by using proper penalty functions. However, the development of specific constraint handling techniques is planned for a future version.

Author's personal copy

Algorithm 1: UPSO and DE common operation framework.

```
   // Initialization:
1  Randomly initialize swarm.
2  Evaluate swarm.
   // Main iteration loop:
3  for i = 1, 2, ... do
4  |  Update swarm (Eqs. (8)–(9) for PSO, Eqs. (12)–(17) for DE).
5  |  Evaluate swarm.
6  |  Update best positions (Eq. (4) for PSO, Eq. (18) for DE).
7  |  Check stopping criterion.
8  end
```

2.5. The Merlin optimization environment

The *Merlin* optimization environment [41] is an efficient and robust general purpose optimization package. It is designed to solve multi-dimensional optimization problems in the form of Eq. (1). Merlin offers a variety of well established gradient-based and gradient-free optimization algorithms.

Gradient-based algorithms include three methods from the conjugate gradient family [55–57], the method of Levenberg–Marquardt [58–60], the DFP [61] and several variations of the BFGS algorithm [62–64]. The gradient-free algorithms include a pattern search [65] and the nonlinear simplex method [66,67].

Merlin can be used interactively as a standalone program, admitting instructions from an available repertoire. In addition it can be called as a subprogram from another code that requires local optimization. An important feature of Merlin is its ability to control an optimization process through the *Merlin Control Language* (MCL) [40]. MCL is a high-level programming language with a rich syntax including, among others, variables, loops, condition checks, file handling, I/O procedures and subprograms. Through the use of MCL, one can implement complex and effective optimization strategies.

The usage of a versatile environment, such as Merlin, for local optimization is crucial to our implementation. It provides the capability to deploy different algorithms at the local search step of an MA, suited to the specific form of the objective function (e.g., noisy, nonlinear sum-of-squares, discontinuous derivatives etc.) Assigning the local optimization task to Merlin, one has the immediate benefit that the customization (i.e., selection of the most proper method) requires no intervention in the original source code but only a proper entry into an input file.

3. Algorithmic description

In this section we present some algorithmic details of the MEMPSODE software. To avoid repetitions, we will assume that all presented concepts referring to the swarm, best positions and particles in PSO, hold also for the population and individuals in DE, respectively.

The fundamental goals of the MEMPSODE's architecture were *modularity* and *reusability*. At the beginning, the apparent structural similarities between UPSO and DE were identified and the corresponding procedures were implemented as autonomous modules. A thorough examination of both algorithms exposes the following common features:

(1) They both operate on a set of potential solutions.
(2) They share the same operation framework, as reported in Algorithm 1.
(3) The LS schemes can be uniformly incorporated within the algorithms' general framework.

The memetic strategies implemented in MEMPSODE were adopted from [36] and described in Section 2.2. According to them, a set of initial points (best positions) is probabilistically selected and an LS procedure is applied on each one. If a strictly descending LS algorithm is used, then each LS application is likely to result in a minimizer of the objective function. This minimizer will substitute the original initial point. However, the best positions in PSO do not necessarily change at each iteration. Thus, the probabilistic selection of initial points among them fosters the danger of applying LS on the same points repetitively, gratuitously increasing the algorithm's computational cost.

This problem can be alleviated by exploiting the Merlin's capability of computing the gradient and its norm at a given point. This is done either automatically by using suitable finite difference methods or by using user-provided derivatives of the objective function. Thus, the memetic strategies with strictly descending deterministic LS are subject to a slight modification such that LS is started only from probabilistically selected initial points that are not previously detected minimizers. The pseudocode of these strategies is given in Algorithm 2.

The gain from this implementation detail is twofold. On the one hand, we avoid the futile waste of function evaluations and, on the other hand, we can detect a situation were all best positions are previously detected local minima. In the later case, the user may face a situation where velocities become very small and best positions remain unchanged for a number of iterations, resulting in search stagnation.

Provided that the termination criteria are not satisfied, this situation can be tackled by applying a different strategy to boost the algorithm's exploration capability. In MEMPSODE, we address this deficiency by retaining the particle with the global best position and reinitializing the positions and/or velocities of every other particle. This is experimentally shown to be adequate to enhance the exploration capabilities of the algorithm, as it will be revealed in our example experiments in following sections.

The complete pseudocode of the MAs implemented in MEMPSODE is given in Algorithm 3. Lines 1–7 describe the initialization step, followed by the main iteration loop from line 10 to the end. The UPSO update steps are described in lines 11–26, while DE is described in lines 27–38. The memetic strategies of Algorithm 2 are evoked in line 50, and the loop is completed with the restarting procedure.

Author's personal copy

Algorithm 2: Pseudocode of the memetic strategies implemented in MEMPSODE.

Input: Swarm size, N; best positions, $p_1, p_2, \ldots, p_N$; global best index, g; memetic scheme, `scheme`; probability of local search, ρ; particle local minimum indicator, local; tolerance, $\varepsilon > 0$; objective function, f.

```
// Apply local search on the overall best position
if ((scheme = 1) OR (scheme = 3)) then
    if (local_g = 0) then
        p_g ← LocalSearch(p_g)
        if (‖∇f(p_g)‖ ⩽ ε) then
            local_g ← 1
        end
    end
end
// Apply local search on other best positions
if ((scheme = 2) OR (scheme = 3)) then
    for i = 1, 2, ... N do
        if ((rand() < ρ) AND (local_i = 0)) then
            p_i ← LocalSearch(p_i)
            if (‖∇f(p_i)‖ ⩽ ε) then
                local_i ← 1
            end
        end
    end
end
```

4. Software description

MEMPSODE may be built either as a standalone executable, or as a library providing a user callable interface. In both cases all algorithmic parameters are controlled by a rich variety of options. It's main components are written in ANSI C and are linked against the Merlin optimization environment in case the user wishes to apply local optimization procedures. The software contains a large set of sample objective functions including an interface to the Tinker [44] package for molecular mechanics calculations. During optimization MEMPSODE prints informative messages on the screen and upon termination appropriate output files are created. For installation instructions and detailed examples we refer the interested reader to the user's manual contained in the software distribution.

4.1. User-written subprograms

The user must provide the following subprograms:

(1) `void Objective_F (double x[], int n, double *f)`
Returns the value of the objective function evaluated at x.

`x`	(input)	Array containing the point at which the calculation is desired.
`n`	(input)	The dimensionality of the function.
`f`	(output)	Objective function value.

(2) `void Bounds_F(double l[], double r[], int n)`
Returns the double-precision array `l` with the lower bounds and the double-precision array `r` with the upper bounds of the variables.

`n`	(input)	The dimensionality of the function.
`l`	(output)	Array containing the lower bounds.
`r`	(output)	Array containing the upper bounds.

If the first-order derivatives are analytically known, they can be provided by the following optional subprogram:

(3) `void Objective_G (double x[], int n, double g[])`
Returns the gradient vector evaluated at x.

`x`	(input)	Array containing the point at which the calculation is desired.
`n`	(input)	The dimensionality of the function.
`g`	(output)	Gradient vector.

4.2. Installation and execution of the standalone version

Assuming that the above user-written subprograms are included in file `fun.c`, the standalone executable can be built using the `make` utility:

```
make OBJECTIVE=fun
```

Subsequently, execution is initiated from the command line:

Author's personal copy

Algorithm 3: Pseudocode of the implemented memetic algorithm.

Input: Objective function, $f : X \subset \mathbb{R}^n \to \mathbb{R}$; algorithm PSO/DE: *algo*; swarm size: N; memetic strategy: `memetic`, maximum function evaluations: `maxfev`; unification factor: `UF`; use mutation: `mut`; probability for local search: ρ
Output: Best detected solution: x^*, $f(x^*)$.

```
// Initialization
for i = 1, 2, ..., N do
    Initialize position x_i and velocity u_i
    Set p_i ← x_i // Initialize best position
    f_i ← f(x_i(0)) // Evaluate particle
    f_i^p ← f_i // Best position
    local_i ← 0 // Best position is minimum is set to false
end
// Update Best Indices
Calculate global best index g_1 and local best index array g_2
// Main Iteration Loop
Set t ← 0
while termination criterion do
    // Update Swarm
    if algo = 'pso' then
        for i = 1, 2, ..., N do
            Calculate local best velocity update u_i^l using g_1
            Calculate global best velocity update u_i^g using g_2(i)
            if mut = 1 then
                // Unified PSO with mutation
                R ← N(μ, σ)
                if rand() ⩽ 0.5 then
                    u_i ← R UF u_i^l + (1 − UF) u_i^g // Unified PSO + Mutate local term
                else
                    u_i ← UF u_i^l + R(1 − UF) u_i^g // Unified PSO + Mutate global term
                end
            else
                u_i ← UF u_i^l + (1 − UF) u_i^g // Unified PSO
            end
            x_i = x_i + u_i // Update particle's position
        end
    else if algo = 'de' then
        for i = 1, 2, ..., N do
            x_i ← p_i // Replicate best positions p to swarm array x
        end
        for i = 1, 2, ..., N do
            Calculate u_i using a strategy from Eqs. (12)–(16)
            j_rand ← ⌊n · rand()⌋
            for j = 1, 2, ..., n do
                if rand() ⩽ CR or j = j_rand then
                    x_ij ← u_ij
                else
                    x_ij ← p_ij
                end
            end
        end
    end
    // Evaluate Swarm
    for i = 1, 2, ..., N do
        f_i ← f(x_i) // Evaluate particle
    end
    // Update Best Positions
    for i = 1, 2, ..., N do
        if f_i < f(p_i) then
            p_i ← x_i
            f_i^p ← f_i
        end
    end
    Calculate global best index g_1 and local best index array g_2
    Apply one of the schemes of Algorithm 2 using ρ
    Calculate global best index g_1 and local best index array g_2
    // If all best positions are local minima, restart
    if Σ_{i=1}^{N} local_i = N then
        Keep global best particle and reinitialize the swarm
    end
end
```

Author's personal copy

`mempsode`	`-d 20` problem's dimension	`-a pso` choice between UPSO and DE	`-l 2` memetic scheme	`-s 100` swarm size	`-f 10000` maximum function evaluations

The above command executes MEMPSODE using UPSO (option `-a pso`) on the 20-dimensional instance of the provided objective function (option `-d 20`), using swarm size 100 (option `-s 100`), maximum number of function evaluations 10 000 (option `-f 10000`) and the second memetic scheme (option `-l 2`). An exhaustive list of all available command line options is provided in the user's manual.

During the optimization procedure, the software provides informative printout information (iterations, function evaluations, minimum objective function value) and upon termination a detailed message containing the minimum, the total number of function evaluations, the number of local searches etc. The user's manual contains a detailed description of several output files produced during execution.

4.3. Installation and use of the library version

MEMPSODE may be built as a library by issuing:

```
make lib
```

This will create the file `libmempsode.a` which contains a set of interface routines. In order to use them, the user must provide appropriate call statements as illustrated below:

```
#include "mempsode.h"
...
init_mempsode();
set_mempsode_iparam("dimension", 20);
set_mempsode_cparam("algorithm", "pso");
set_mempsode_iparam("memetic", 2);
set_mempsode_iparam("swarm-size", 100);
set_mempsode_iparam("max-fun-evals", 10000);
mempsode();
get_mempsode_min("minval", &val);
...
```

The above code sets various MEMPSODE parameters, calls the main optimization routine and finally retrieves the best function value found. Assuming that file `main.c` contains the above code fragment and file `fun.c` contains the objective function, they may be linked against the MEMPSODE library using:

```
gcc main.c fun.c -L/path/to/mempsode -I/path/to/mempsode -lmempsode
```

The complete list of interface routines as well as various installation options are described in the user manual.

4.4. Ready to use functions

A large set of objective functions established in the literature for the assessment of global optimization algorithms, are provided in the software distribution (file `SOURCE_problems.c`). Among others the well known Rastrigin [68], Ackley [69], Griewank [70], Levy [2], Bohachevsky [71], and Guilin Hills [72] functions are included. The Lennard-Jones interaction potential for modeling atomic clusters is provided in file `lj.c`. In addition, an interface to the Tinker molecular modeling package is provided (file `tinker.c`) that evaluates the energy of biomolecular systems using a selection of different force fields.

4.5. Stopping criteria

Execution of the software terminates when any one of the following criteria is met:

(a) The number of function evaluations performed so far (calls to the user supplied routine `Objective_F`) exceeds a user-defined upper bound which is specified using the `-f` command line option.
(b) The number of gradient evaluations performed so far (calls to the user supplied routine `Objective_G`) exceeds a user-defined upper bound which is specified using the `-g` command line option.
(c) The number of UPSO or DE cycles exceeds a user-defined upper bound, which is specified using the `-i` command line option.
(d) The value of the objective function reaches or falls below a user-defined target value, which is specified using the `-t` command line option.

If any one of the `-g`, `-i` or `-t` options is not supplied then the corresponding termination criterion is not used. In case the `-f` option is not specified then a upper bound of $100\,000 \times n$ function evaluations is assumed.

5. Sample applications

We provide three sample applications to illustrate the use of MEMPSODE and demonstrate its applicability in various types of problems.

5.1. The Rastrigin objective function

The Rastrigin function is well established in the assessment of global optimization methods. It is given by:

Author's personal copy

$$f(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right), \tag{19}$$

where $x_i \in [-5.12, 5.12]$, $i = 1, 2, \ldots, n$. This function has a global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with value $f(x^*) = 0$. In the software distribution the Rastrigin function is coded in `rastrigin.c`. The user compiles MEMPSODE using the command:

```
make OBJECTIVE=rastrigin
```

We focus on the 30-dimensional instance of this function, using UPSO with unification factor 1.0 (standard gbest PSO), swarm size 20, memetic Scheme 2 with local search probability 0.1, and a maximum of 5×10^5 function evaluations. In addition intermediate results are displayed every 500 iterations and the initial velocity scaling factor is 0.01. Since local search is used, the user must also define the LS method and its parameters. In this example we use the BFGS method with a maximum of 1000 function evaluations per LS. The corresponding file is defined with the `-y` command line option and contains the following line:

```
bfgs noc 1000
```

The program is executed through the command:

```
mempsode -d 30 -a pso -l 2 -s 20 -f 500000 -c 0.01 -u 1.0 -n 1 -D 500 -y in.dat
```

and produces the following screen output:

```
---------------------------------
Experiment 0
---------------------------------
Iter:  500, FunEvals: 63121,  Val: 3.283363E+01, Std: 4.250217, Vel: 0.102400
Iter: 1000, FunEvals: 106916, Val: 1.492438E+01, Std: 3.148346, Vel: 0.102400
Iter: 1500, FunEvals: 127425, Val: 1.492438E+01, Std: 3.146285, Vel: 0.102400
Iter: 2000, FunEvals: 166089, Val: 1.094454E+01, Std: 3.068551, Vel: 0.102400
Iter: 2500, FunEvals: 196388, Val: 8.954626E+00, Std: 3.407066, Vel: 0.102400
Iter: 3000, FunEvals: 229887, Val: 5.969754E+00, Std: 3.105790, Vel: 0.102400
Iter: 3500, FunEvals: 266735, Val: 4.974795E+00, Std: 2.830046, Vel: 0.102400
Iter: 4000, FunEvals: 312303, Val: 4.974795E+00, Std: 3.100652, Vel: 0.102400
Iter: 4500, FunEvals: 356134, Val: 2.984877E+00, Std: 3.757278, Vel: 0.102400
Iter: 5000, FunEvals: 395173, Val: 1.989918E+00, Std: 2.084046, Vel: 0.102400
Iter: 5500, FunEvals: 426884, Val: 9.949591E-01, Std: 2.120236, Vel: 0.102400
Iter: 6000, FunEvals: 464246, Val: 9.949591E-01, Std: 3.980862, Vel: 0.102400
---------------------------------
PROBLEM PARAMETERS
Problem : 0
Dimension : 30
NumOfExp : 1
Seed : 1
MaxIter : inf
MaxFev : 500000
MaxGev : inf
Target : -inf
Xmin : [ -5.120 -5.120 -5.120 -5.120 -5.120 -5.120 -5.120 -5.120 ...
Xmax : [ 5.120 5.120 5.120 5.120 5.120 5.120 5.120 5.120 5.120 ...

UPSO PARAMETERS
Memetic : 2
UF : 1.000
Vscale : 0.010
Prob : 0.010
x, c1, c2 : 0.729 2.050 2.050
Nradius : 1
R3use : 0
R3mean : 0.00
R3std : 1.00
SS : 20
=============================================================================================================
EXP - S - F(SOL) - ITER - FEVALS - LOCAL - GEVALS -        LAST HIT             - BPUPD - CPU TIME
                                                    ITER , FEVALS, GEVALS, LOCAL

 1 - 0 - 0.000000E+00   - 6388   - 500667 -  502 - 0 - [ 6053, 473070,  0,  470] - 7602  -  4.38
=============================================================================================================
Execution Complete
```

Author's personal copy

Table 1
Results for the Rastrigin test problem.

Prob. dim.	DE-Scheme 2			UPSO-Scheme 2		
	Succ.	LOCAL	FEVALS	Succ.	LOCAL	FEVALS
$n = 10$	100%	215	114571	100%	397	231690
$n = 30$	100%	2039	2020261	100%	3241	3389970
$n = 50$	100%	4836	6879354	98%	6140	9257422
Prob. dim.	DE-Scheme 3			UPSO-Scheme 3		
	Succ.	LOCAL	FEVALS	Succ.	LOCAL	FEVALS
$n = 10$	100%	222	118884	100%	428	254345
$n = 30$	100%	2024	2016684	100%	3106	3249649
$n = 50$	100%	4972	7012893	98%	6001	9063438

In the upper part of the output, MEMPSODE prints intermediate results every 500 iterations (`Iter`). Note that an iteration of the algorithm corresponds to a complete cycle where all particles are updated. The output includes the number of function evaluations (`FunEvals`) performed up to the specific iteration (including those of the local search) and the best approximation to the global minimum found so far (`Val`). In addition the output contains the standard deviation of the average swarm position, defined as:

$$\texttt{Std} = \sum_{i=1}^{N} (x_i - \tilde{x})^{\top} (x_i - \tilde{x}),$$

where $\tilde{x} = \dfrac{1}{N} \sum_{i=1}^{N} x_i$. In the case of PSO the output also contains the maximum velocity component defined as:

$$\texttt{Vel} = \max_{\substack{i=1,\ldots,N \\ j=1,\ldots,n}} |v_{ij}|. \tag{20}$$

When execution completes, MEMPSODE prints a complete list of the parameters used in the experiment(s) as well as the final output in a single line per experiment. `EXP` is the serial number of the experiment and `S` is a flag that is equal to 1 if the algorithm successfully reached the target value (as specified by the `-t` command line option). Two sets of counters are reported: `ITER`, `FEVALS`, `LOCAL` and `GEVALS` are the total number of iterations, function evaluations, local searches and gradient evaluations respectively, that were performed by the algorithm. The value of these counters at the time the lowest function value was discovered is displayed under the heading `LAST HIT`. `BPUPD` is the total number of best position updates and `CPU TIME` is the execution time for the specific experiment in seconds. MEMPSODE writes the global minimizer in an output file that follows a specific naming convention and ends in `_sol` (details are given in the user's manual).

Using this test function we performed a series of runs for ($n = 10, 30, 50$) using both UPSO and DE with the default operator (OP1) on a swarm of 50 particles. In each case two different memetic schemes (Schemes 2 and 3) were employed. Each run was repeated 50 times and was terminated when the target value 0.0 or a maximum number of 10^7 function evaluations was reached. For the local search we applied the BFGS method with an upper bound of 5000 function evaluations. For the Rastrigin instance with $n = 10$, the following command lines were used:

```
mempsode -d 10 -a pso -l 2 -s 50 -f 10000000 -t 0.0 -e 50 -y in.dat
mempsode -d 10 -a pso -l 3 -s 50 -f 10000000 -t 0.0 -e 50 -y in.dat
mempsode -d 10 -a de  -l 2 -s 50 -f 10000000 -t 0.0 -e 50 -y in.dat
mempsode -d 10 -a de  -l 3 -s 50 -f 10000000 -t 0.0 -e 50 -y in.dat
```

The percentage of successful runs (those that approximate the global minimum value within 10^{-6}, the average number of local searches conducted and the average number of function evaluations are presented in Table 1. The global minimum was located with 100% probability with the exception of UPSO in the case $n = 50$. However when we increased the maximum number of function evaluations to 2×10^7 the success rate of this case also reached 100%. The experiments indicate that for this test, hybrid DE with Schemes 2 and 3, performs better than the corresponding hybrid UPSO. Schemes 2 and 3 appear to have almost the same performance.

5.2. Lennard-Jones clusters

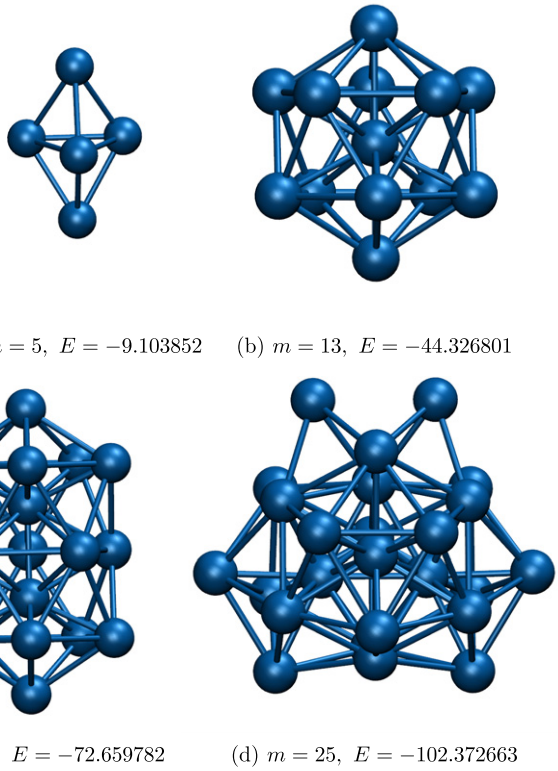
The study of atomic clusters has attracted much interest during the past decades due to their unique properties and their potential applications. Finding the global minimum conformation for a cluster of atoms interacting under two-body central forces, belongs to the class of NP-hard problems [73,74]. In this example application, we find the global minimum of atomic clusters with $m = 5, 13, 19$ and 25 atoms interacting through the Lennard-Jones potential. The total potential energy of the cluster is given:

$$E = \sum_{i<j}^{m} 4\varepsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right], \tag{21}$$

where r_{ij} is the distance between atoms i and j.

The above function, using reduced units ($\sigma = \varepsilon = 1$), is provided in file `lj.c`. The cluster size was set to $5, 13, 19$ and 25 atoms resulting in $15, 39, 57$ and 75 optimization variables respectively. The same parameters as in the previous example were used, with the exception of the target value (`-t`). In the case where $m = 5$ the following command lines were executed:

Author's personal copy



(a) $m = 5,\ E = -9.103852$ (b) $m = 13,\ E = -44.326801$

(c) $m = 19,\ E = -72.659782$ (d) $m = 25,\ E = -102.372663$

Fig. 2. Lennard-Jones clusters.

Table 2
Results for the Lennard-Jones cluster problem.

Cluster size	DE-Scheme 2			UPSO-Scheme 2		
	Succ.	LOCAL	FEVALS	Succ.	LOCAL	FEVALS
5	100%	1	3016	100%	1	2628
13	100%	67	318368	100%	66	313052
19	100%	334	1641552	100%	313	1525869
25	94%	644	3179776	92%	561	2772891
Cluster size	DE-Scheme 3			UPSO-Scheme 3		
	Succ.	LOCAL	FEVALS	Succ.	LOCAL	FEVALS
5	100%	1	185	100%	1	415
13	100%	45	216352	100%	62	287993
19	100%	246	1212390	100%	311	1510478
25	98%	621	2918633	98%	589	2955767

```
mempsode -d 15 -a pso -l 2 -s 50 -f 10000000 -e 50 -y in.dat
mempsode -d 15 -a pso -l 3 -s 50 -f 10000000 -e 50 -y in.dat
mempsode -d 15 -a de  -l 2 -s 50 -f 10000000 -e 50 -y in.dat
mempsode -d 15 -a de  -l 3 -s 50 -f 10000000 -e 50 -y in.dat
```

The global minimum for the four different clusters is displayed in Fig. 2 and it is in accordance with previously reported results [75]. In addition, in Table 2 we list the percentage of successful runs, the average number of local searches and the average number of function evaluations. Since we do not provide a target value, the numbers in Table 2 were taken from the reported `LAST HIT` counters. The cluster with $m = 25$ atoms appears to be a difficult case since all tested methods exhibit a small failure percentage. However when the maximum number of function evaluations was raised to 2×10^8 the success rate reached 100%. Hybrid UPSO performs better when combined with Scheme 2, while hybrid DE shows increased performance in combination with Scheme 3.

5.3. Protein conformation

In order to understand the diverse functionality of proteins at the molecular level, it is necessary to determine their three-dimensional structure. This task is often formulated as a global optimization problem of a suitable chosen potential energy function. In this example we find the minimum energy conformation of a gas phase Alanine octamer. For the potential energy we employ the Amber [76] force field which has the form:

Author's personal copy

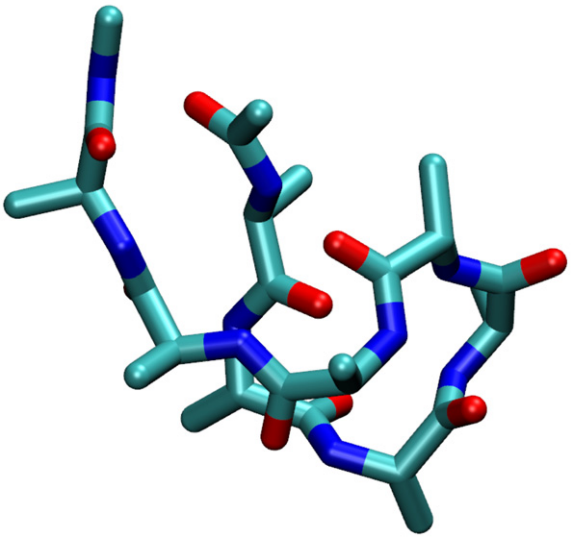


Fig. 3. Global minimum conformation of Alaline octamer ($E = -44.45$ kcal/mol).

Table 3
Dihedral angles (degrees) for the global minimum conformation.

Residue	ϕ	ψ	ω	χ
1	−153.3	143.2	−179.1	60.3
2	−50.2	120.0	−173.3	62.7
3	61.2	−65.6	−167.9	66.5
4	−46.0	−41.1	−174.5	−58.0
5	−136.4	62.3	164.8	−62.6
6	60.4	−76.9	−176.3	65.3
7	−142.3	−82.4	−164.6	59.2
8	−153.4	145.0	−175.8	−59.9

$$E = \sum_{\text{bonds}} k_r(r - r_0)^2 + \sum_{\text{angles}} k_\theta(\theta - \theta_0)^2 + \sum_{\text{dihedrals}} \frac{V_n}{2}\left[1 + \cos(n\phi - \gamma)\right] + \sum_{i<j}\left[\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}}\right] + \sum_{i<j}\left[\frac{q_i q_j}{\epsilon r_{ij}}\right].$$

The parameters are taken from the 1996 parametrization [77]. In order to implement the above energy function we have used the Tinker, molecular modeling package [44]. In the software distribution we provide the file `tinker.c` which offers an interface to the Tinker energy function and its gradient. In order to enable Tinker support, one must supply the full path of the Tinker installation. Consequently MEMPSODE is built by using:

```
make OBJECTIVE=tinker TINKER=/path/to/tinker/source
```

Before running the executable the user has to prepare a Tinker input file containing the description of the protein in internal coordinates. The dimensionality of the problem is determined by the number of free dihedral angles in the input. In this example we use an Alanine octamer blocked by the Acetyl and N-Methyl groups. Each Alanine residue has four dihedral angles (ϕ, ψ, ω, χ) resulting in 35 optimization variables. A corresponding input file (`vp8.int`) is provided in the distribution, while further details on running the software with Tinker support, is provided in the user manual.

We performed 20 experiments using UPSO on a 100 particle swarm, with unification factor 1.0 memetic strategy 2 and probability 0.1 for the local search. Since Tinker provides analytic gradients, they were utilized by the local search procedure. A maximum of 5×10^7 function evaluations was imposed.

All experiments succeeded in locating the global minimum conformation which is shown in Fig. 3, while the corresponding dihedral angles are listed in Table 3. Our result is in agreement with the one reported in [78] using the same force field.

6. Conclusions

Particle Swarm and Differential Evolution are effective and robust population based algorithms. Combined with local search procedures the effectiveness of these methods is enhanced. In this work we presented an implementation of a hybrid algorithm that combines the above population based methods with local searches. The new software is easy to install and use and offers complete control via a plethora of command line options. One of the assets of the software is the use of the Merlin optimization environment that has proved to be a highly effective and versatile package for local optimization. We present three applications of variable complexity and many local minima. For molecular modeling problems we provide an interface to the Tinker package that facilitates the use of a variety of force fields. Our computational experiments confirm the applicability of the hybrid algorithm in diverse optimization problems.

Author's personal copy

References

[1] R. Horst, P.M. Pardalos, Handbook of Global Optimization, Kluwer Academic Publishers, London, 1995.
[2] A. Levy, A. Montalvo, S. Gomez, A. Galderon, Topics in Global Optimization, Springer-Verlag, New York, 1981.
[3] J.D. Pintér, Global Optimization in Action, Academic Publishers, 1996.
[4] S.S. Rao, Engineering Optimization – Theory and Practice, Wiley, 1996.
[5] A. Törn, A. Žilinskas, Global of Optimization, Springer-Verlag, Berlin, 1989.
[6] R. Fletcher, Practical Methods of Optimization, Wiley, New York, 1987.
[7] J. Nocedal, S.J. Wright (Eds.), Numerical Optimization, Springer, 2006.
[8] T. Bäck, Evolutionary Algorithms in Theory and Practice, Oxford University Press, New York, 1996.
[9] H.-G. Beyer, Evolutionary algorithms in noisy environments: Theoretical issues and guidelines for practice, Comput. Methods Appl. Mech. Engrg. 186 (2000) 239–269.
[10] E. Bonabeau, M. Dorigo, G. Théraulaz, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, New York, 1999.
[11] C.A. Coello Coello, D.A. Van Veldhuizen, G.B. Lamont, Evolutionary Algorithms for Solving Multi-Objective Problems, Kluwer, New York, 2002.
[12] A.P. Engelbrecht, Fundamentals of Computational Swarm Intelligence, Wiley, 2006.
[13] J. Kennedy, R.C. Eberhart, Swarm Intelligence, Morgan Kaufmann Publishers, 2001.
[14] M.W.S. Land, Evolutionary algorithms with local search for combinatorical optimization, PhD thesis, University of California, San Diego, USA, 1998.
[15] K.E. Parsopoulos, M.N. Vrahatis, Particle Swarm Optimization and Intelligence: Advances and Applications, Information Science Reference (IGI Global), Hershey, PA, USA, 2010.
[16] D. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, Reading, MA, 1989.
[17] H.-P. Schwefel, Evolution and Optimum Seeking, Wiley, New York, 1995.
[18] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proc. IEEE Int. Conf. Neural Networks, vol. IV, IEEE Service Center, Piscataway, NJ, 1995, pp. 1942–1948.
[19] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optimization 11 (1997) 341–359.
[20] R. Dawkins, The Selfish Gene, Oxford University Press, New York, 1976.
[21] P. Moscato, On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, Technical Report C3P Report 826, Caltech Concurrent Computation Program, California, USA, 1989.
[22] P. Moscato, Memetic algorithms: A short introduction, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, London, 1999, pp. 219–235.
[23] R.K. Belew, J. McInerny, N.N. Schraudolph, Evolving networks: Using the genetic algorithm with connectionist learning, in: C.G. Langton, C. Taylor, J.D. Farmer, S. Rasmussen (Eds.), Proceedings of the Second Conference in Artificial Life, Addison-Wesley, 1991, pp. 511–548.
[24] R. Geesing, D.G. Stork, Evolution and learning in neural networks: the number and distribution of learning trials affect the rate of evolution, in: R.P. Lippmann, J.E. Moody, D.S. Touretzky (Eds.), NIPS 3, Morgan Kaufmann, 1991, pp. 804–810.
[25] W.E. Hart, Adaptive global optimization with local search, PhD thesis, University of California, San Diego, USA, 1994.
[26] G.E. Hinton, S.J. Nowlan, How learning can guide evolution, Complex Systems 1 (1987) 495–502.
[27] N. Krasnogor, Studies on the theory and design space of memetic algorithms, PhD thesis, University of the West of England, Bristol, United Kingdom, 2002.
[28] B. Liu, L. Wang, Y.-H. Jin, An effective pso-based memetic algorithm for flow shop scheduling, IEEE Transactions on Systems, Man and Cybernetics, Part B 37 (2007) 18–27.
[29] P. Merz, Memetic algorithms for combinatorial optimization: fitness landscapes and effective search strategies, PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany, 1998.
[30] B. Liu, L. Wang, Y.-H. Jin, D.-X. Huang, Designing neural networks using PSO-based memetic algorithm, in: D.-R. Liu, S.-M. Fei, Z.-G. Hou, H.-G. Zhang, C.-Y. Sun (Eds.), Advances in Neural Networks ISNN 2007, in: Lecture Notes in Computer Science, vol. 4493, Springer, Berlin/Heidelberg, 2007, pp. 219–224.
[31] M. Lozano, F. Herrera, N. Krasnogor, D. Molina, Real-coded memetic algorithms with crossover hill-climbing, Evolutionary Computation 12 (3) (2004) 273–302.
[32] D. Molina, M. Lozano, C. García-Martínez, F. Herrera, Memetic algorithms for continuous optimisation based on local search chains, Evolutionary Computation 18 (1) (2010) 27–63.
[33] Q.H. Nguyen, Y.S. Ong, M.H. Lim, A probabilistic memetic framework, IEEE Trans. Evol. Comput. 13 (3) (2009) 604–623.
[34] N. Noman, H. Iba, Accelerating differential evolution using an adaptive local search, IEEE Trans. Evol. Comput. 12 (1) (2008) 107–125.
[35] S. Muelas, A. La Torre, J. Pea, A memetic differential evolution algorithm for continuous optimization, in: Ninth International Conference on Intelligent Systems Design and Applications, ISDA'09, IEEE, 2009, pp. 1080–1084.
[36] Y.G. Petalas, K.E. Parsopoulos, M.N. Vrahatis, Memetic particle swarm optimization, Annals of Operations Research 156 (1) (2007) 99–127.
[37] O. Schutze, E. Talbi, C.C. Coello, L.V. Santana-Quintero, G.T. Pulido, A memetic PSO algorithm for scalar optimization problems, in: Swarm Intelligence Symposium, SIS 2007, IEEE, 2007, pp. 128–134.
[38] K.E. Parsopoulos, M.N. Vrahatis, UPSO: A unified particle swarm optimization scheme, in: Proceedings of the International Conference of Computational Methods in Sciences and Engineering (ICCMSE 2004), in: Lecture Series on Computer and Computational Sciences, vol. 1, VSP International Science Publishers, Zeist, The Netherlands, 2004, pp. 868–873.
[39] K.E. Parsopoulos, M.N. Vrahatis, Parameter selection and adaptation in unified particle swarm optimization, Mathematical and Computer Modelling 46 (1–2) (2007) 198–213.
[40] D.G. Papageorgiou, I.N. Demetropoulos, I.E. Lagaris, Merlin control language for strategic optimization, Comp. Phys. Commun. 109 (1) (1998) 250–275.
[41] D.G. Papageorgiou, I.N. Demetropoulos, I.E. Lagaris, MERLIN-3.1. 1. A new version of the Merlin optimization environment, Comp. Phys. Commun. 159 (1) (2004) 70–71.
[42] D.H. Wolpert, W.G. Macready, No free lunch theorem for optimization, IEEE Trans. Evol. Comput. 1 (1997) 67–82.
[43] S. Droste, T. Jansen, I. Wegener, Optimization with randomized search heuristics – the (A) NFL theorem, realistic scenarios, and difficult functions* 1, Theoretical Computer Science 287 (1) (2002) 131–144.
[44] J.W. Ponder, et al., TINKER: Software Tools for Molecular Design, Department of Biochemistry and Molecular Biophysics, Washington University School of Medicine, St. Louis, MO, 1998.
[45] R.C. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings Sixth Symposium on Micro Machine and Human Science, IEEE Service Center, Piscataway, NJ, 1995, pp. 39–43.
[46] M. Clerc, J. Kennedy, The particle swarm–explosion, stability, and convergence in a multidimensional complex space, IEEE Trans. Evol. Comput. 6 (1) (2002) 58–73.
[47] I.C. Trelea, The particle swarm optimization algorithm: Convergence analysis and parameter selection, Information Processing Letters 85 (2003) 317–325.
[48] K.E. Parsopoulos, M.N. Vrahatis, Unified particle swarm optimization for solving constrained engineering optimization problems, in: L. Wang, K. Chen, Y.S. Ong (Eds.), Lecture Notes in Computer Science (LNCS), vol. 3612, Springer-Verlag, 2005, pp. 582–591.
[49] K.E. Parsopoulos, M.N. Vrahatis, Studying the performance of unified particle swarm optimization on the single machine total weighted tardiness problem, in: A. Sattar, B.H. Kang (Eds.), Lecture Notes in Artificial Intelligence (LNAI), vol. 4304, Springer, 2006, pp. 760–769.
[50] S.S. Rao, Optimization: Theory and Applications, Wiley Eastern, 1992.
[51] Y.G. Petalas, K.E. Parsopoulos, M.N. Vrahatis, Stochastic optimization for detecting periodic orbits of nonlinear mappings, Nonlinear Phenomena in Complex Systems 11 (2) (2008) 285–291.
[52] Y.G. Petalas, K.E. Parsopoulos, M.N. Vrahatis, Improving fuzzy cognitive maps learning through memetic particle swarm optimization, Soft Computing 13 (1) (2009) 77–94.
[53] K. Price, Differential evolution: A fast and simple numerical optimizer, in: Proceedings NAFIPS'96, 1996, pp. 524–525.
[54] P.E. Gill, W. Murray, M.H. Wright, Practical Optimization, Academic Press, London, 1981.
[55] R. Fletcher, C.M. Reeves, Function minimization by conjugate gradients, The Computer Journal 7 (2) (1964) 149.
[56] E. Polak, G. Ribiere, Note sur la convergence de méthodes de directions conjuguées, Revue Française d'Informatique et de Recherche Opérationnelle 16 (1969) 35–43.
[57] K.M. Khoda, Y. Liu, C. Storey, Generalized Polak–Ribiere algorithm, Journal of Optimization Theory and Applications 75 (2) (1992) 345–354.
[58] K. Levenberg, A method for the solution of certain nonlinear problems in least squares, Quart. Appl. Math. 2 (2) (1944) 164–168.

Author's personal copy

[59] D.W. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, Journal of the Society for Industrial and Applied Mathematics 11 (2) (1963) 431–441.
[60] J. More, The Levenberg–Marquardt algorithm: Implementation and theory, Numerical Analysis (1978) 105–116.
[61] W.C. Davidon, Variable metric method for minimization, Argonne Natl. Labs., ANL-5990 Rev., 1959.
[62] R. Fletcher, A new approach to variable metric algorithms, The Computer Journal 13 (3) (1970) 317.
[63] M.J.D. Powell, A tolerant algorithm for linearly constrained optimization calculations, Mathematical Programming 45 (1) (1989) 547–566.
[64] M.J.D. Powell, Rank one methods for unconstrained optimization, Technical report, Atomic Energy Research Establishment, Harwell (England), 1969.
[65] G. Evangelakis, Merlin – a portable system for multidimensional minimization, Comp. Phys. Commun. 46 (3) (1987) 401–415.
[66] J.A. Nelder, R. Mead, A simplex method for function minimization, The Computer Journal 7 (4) (1965) 308.
[67] J.A. Nelder, R. Mead, A simplex method for function minimization, The Computer Journal 8 (1) (1965) 27 (errata).
[68] A. Törn, A. Zilinskas, Global Optimization, Lecture Notes in Computer Science, vol. 350, 1989.
[69] D.H. Ackley, A Connectionist Machine for Genetic Hillclimbing, Kluwer Academic Publishers, Boston, 1987.
[70] A.O. Griewank, Generalized descent for global optimization, Journal of Optimization Theory and Applications 34 (1) (1981) 11–39.
[71] I.O. Bohachevsky, M.E. Johnson, M.L. Stein, Generalized simulated annealing for function optimization, Technometrics 28 (3) (1986) 209–217.
[72] F.J. Hickernell, Y.A.A. Yuan, A simple multistart algorithm for global optimization, OR Transactions 1 (2) (1997) 1–11.
[73] L.T. Wille, J. Vennik, Computational complexity of the ground-state determination of atomic clusters, Journal of Physics A: Mathematical and General 18 (1985) L419.
[74] G.W. Greenwood, Revisiting the complexity of finding globally minimum energy configurations in atomic clusters, Zeitschrift für Physikalische Chemie 211 (1999) 105–114.
[75] D.J. Wales, J.P.K. Doye, Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms, J. Phys. Chem. A 101 (28) (1997) 5111–5116.
[76] W.D. Cornell, P. Cieplak, C.I. Bayly, I.R. Gould, K.M. Merz, D.M. Ferguson, D.C. Spellmeyer, T. Fox, J.W. Caldwell, P.A. Kollman, A second generation force field for the simulation of proteins, nucleic acids, and organic molecules, Journal of the American Chemical Society 117 (19) (1995) 5179–5197.
[77] P. Kollman, R. Dixon, W. Cornell, T. Fox, C. Chipot, A. Pohorille, The development/application of a “minimalist” organic/biochemical molecular mechanic force field using a combination of ab initio calculations and experimental data, Computer Simulations of Biomolecular Systems: Theoretical and Experimental Applications (1997) 83–96.
[78] P.N. Mortenson, D.J. Wales, Energy landscapes, global optimization and dynamics of the polyalanine Ac (ala) NHMe, The Journal of Chemical Physics 114 (2001) 6443.