



Optimization of analytic density functionals by parallel genetic algorithm

Matthew A. Thompson ^{*,1}, Brett I. Dunlap

Code 6189, Theoretical Chemistry Section, Naval Research Laboratory, Washington, DC 20375, United States

ARTICLE INFO

Article history:

Received 15 July 2008

In final form 20 August 2008

Available online 23 August 2008

ABSTRACT

We report the construction and implementation of a parallel genetic algorithm (PGA) for use in optimization of analytic density-functional theory. Our new algorithm demonstrates comparable performance to our previous simplex optimizer when benchmarked against the extended G2 set, and the considerable scatter between different optimizations shows that robust methods are required. This new PGA will allow us to further expand our parameter space while efficiently utilizing cluster and supercomputing resources as the problems grow larger.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

There are two main categories of electronic structure calculations performed today. The first is Gaussian-based *ab initio* quantum chemistry building on Hartree–Fock (HF) theory [1] and further, more refined HF-based theories. The energy and other properties calculated via these theories have the advantage of analytical calculation of properties, allowing calculation to arbitrary precision. Moreover, HF and associated methods have the advantage that they can be systematically improved toward exact results. Their disadvantage is that the calculations are quite expensive, scaling as the fourth-power of the number of atomic orbitals, N^4 , at best.

The second category of electronic structure calculation is that based on density-functional methods: Kohn–Sham (KS) density-functional theory (DFT) [2–4]. A major advantage of DFT is that the calculations scale as N^3 , so that larger systems are possible in comparison to HF-based theories. In DFT, the Kohn–Sham energy can be partitioned as so

$$E^{\text{KS}}[\rho] = T_s + E_{\text{nuc}} + J_{ee} + E_{\text{xc}}[\rho] \quad (1)$$

where T_s is the so-called ‘noninteracting’ kinetic energy, E_{nuc} is the electron–nuclei interaction energy, J_{ee} is the Coulomb interaction, and E_{xc} is the exchange–correlation (XC) energy. This XC energy encapsulates both the difference of the exact kinetic energy from the noninteracting kinetic energy and the difference between the exact electron–electron interaction and the sum of the Coulomb interaction, the exchange interaction and electron correlation interaction. This also accounts for a spurious self-interaction term that is corrected in HF theory, but not in KS theory. If this XC energy functional were known exactly, the Kohn–Sham energy would be the exact energy of the system; unfortunately, the XC functional is

not known exactly. Much of current DFT research is dedicated to better and more complex approximations to this XC functional with a ‘Jacob’s ladder’ of techniques [5], such as the generalized-gradient approximation (GGA), *meta*-GGA, and hyper-GGA.

While the first three terms of the Kohn–Sham energy can be calculated analytically, to date modern XC functionals and functional derivatives must be calculated on a numerical grid, even if an analytically fitted density is used to generate the XC energy and its functional derivatives on that grid of points [6]. Numerical grids, of course, do not have the advantage of analytical computation in that precision is no longer arbitrary, but dependent on the complexity of the grid. Moreover, energies calculated on a numerical grid are dependent on the orientation of the molecule to the grid, obviously an unphysical property. Plus, while grid refinements are always possible, the refining comes at the cost of computational speed.

Recently, however, we have formulated a completely variational and analytical solution for DFT which has shown promising results [7]. In this method, one of the few XC functionals that can be expressed and calculated analytically, Slater’s X_α exchange functional, is used

$$E_{\text{xc}}[\rho] = E_x[\rho] = -\alpha \cdot \frac{9}{4} \left(\frac{3}{4\pi} \right)^{1/3} \langle \rho^{4/3} \rangle, \quad (2)$$

where α is set on a per-element basis. In recent work, Zope and Dunlap calculated a set of element-specific α values, herein denoted by $\{\alpha\}$, by optimizing against the mean absolute error in the atomization energy [8] for the 148 molecules in the extended G2 set [9–11] using a simplex algorithm. This optimization allowed the analytic DFT (ADFT) using this simple functional to perform on par with PBE-GGA, a functional based on the GGA, whose evaluation requires computation of both orbital values and gradients on a numeric grid. Analytic DFT has also shown good performance in calculation of dipole moments [12] as well, and the technique has been used for 1-D, 2-D, and 3-D periodic systems in solid state physics [13]. Plus, being both density-functional based and analytic, the

* Corresponding author. Fax: +1 202 767 1716.

E-mail address: matthew.thompson.ctr@nrl.navy.mil (M.A. Thompson).

¹ Supported by an NRC/NRL Postdoctoral Research Associateship.

method scales better and is faster than comparable HF methods and so can be applied to larger molecules for the same cost. Moreover, by its analytic nature, it is generally faster than grid-based DFT methods as well, although recent developments in using analytically fitted densities for the three-center integrals [14] and exchange-correlation potential [6] have led to significant speedups for grid-based methods. In fact, with ADFT, it is now possible to optimize the geometry of a 2160-atom fullerene using an all-electron triple- ζ polarization basis [15] (c. 39000 basis functions).

Thus, ADFT holds great promise as an efficient method of calculating molecular properties, but, an extension beyond the local density approximation (LDA) would require optimization of larger numbers of element-specific parameters. Such an increase in optimization space would quickly overwhelm traditional optimizers like least-squares/simplex, requiring the integration of more robust optimizers, specifically a genetic algorithm.

Genetic algorithms (GAs) are optimization algorithms that use the evolutionary principle of survival of the fittest to find optimum solutions to problems. Genetic algorithms were first presented by Holland [16] as an abstraction of biological evolution. The genetic algorithm he presented is a method of generating a new population of ‘chromosomes’ or ‘individuals’ from old ones via ‘crossover’ (mating), ‘mutation’, and ‘inversion’. This algorithm was a major innovation in the field of evolutionary computation that began with Rechenberg [17] in the 1960s and 70s. Genetic algorithms are considered robust in that they work in a myriad of domains that frustrate more traditional search methods based on gradients. They are commonly used to solve problems that scale exponentially with the size of the system (so-called NP-hard problems). Finding a minimum of a function with a large amount of noise can frustrate gradient-based solvers. Gradient solvers tend to get caught in local minima, which GAs can avoid; they thus have a much better chance to find the global minimum [18].

2. Methods

A genetic algorithm was constructed using version 2.5.1 of the open-source Python programming language [19]. Within this GA, the individuals were a set of α values for use in our ADFT code, one for each of the elements under investigation. The initial population of 32 individuals (i.e., sets of α values) was constructed by selecting for each element, a random α uniformly selected from between 0.5 and 0.9. This range was chosen primarily from experience [8] that lower and higher α tend to lead to unphysical results and testing showed this to be true in our case.

The GA then proceeds as usual. First, the fitness, f_j , for a particular set of α values, $\{\alpha\}_j$, was determined as in the previous work [8] by calculating the mean absolute error (MAE) of the atomization energy versus the benchmark value over a set of compounds. (For this work, the fitness, f_j , in our algorithm is the *inverse* of the MAE as the selection routine is based on finding the maximum fitness.) In this work, we chose to optimize over the extended G2 set [9–11] of molecules to directly compare to the previous optimization [8]. The best individual, $\{\alpha\}_{\text{best}}$, with the best fitness, f_{best} , is the set of α values leading to the minimal MAE.

Individuals were then selected for reproduction according to the fan selection algorithm [20] which has been shown to perform better than the classic roulette algorithm with convergence behavior. We first calculate the *relative fitness*, f_j

$$f_j = \frac{f_j}{\sum_{i=1}^{\text{popsize}} f_i} \quad (3)$$

as normal with roulette selection. We then modify the relative fitness of the best individual, f_{best} , using the ‘fan parameter’ a as follows:

$$\tilde{f}_{\text{best}} = f_{\text{best}} + (1 - f_{\text{best}}) \cdot a, \quad (4)$$

while other individuals’ relative fitnesses, f_j , are modified via

$$\tilde{f}_j = (1 - \tilde{f}_{\text{best}}) \cdot \left(f_j + \frac{f_{\text{best}}}{\text{popsize} - 1} \right), \quad (5)$$

where the ‘tilded’ fitness, \tilde{f} , refers to the newly modified fitnesses. In our case, we selected a ‘fan parameter’ $a = 0.3$; we note, however, that from limited testing, any parameter greater than 0.0 (i.e., any deviation from standard roulette selection) led to faster convergence behavior for the GA with no effect on the final result of the optimization.

The selected individuals were then mated using a crossover operator which, in our case, was an arithmetic average of the α of the parents, element by element. A mutation operator, replacement of a random α in a child by a random value in the range cited above, was then applied according to the selected mutation rate. For the purposes of the results presented in this Letter, the GA used a crossover rate of 1.0 and a mutation rate of 0.05. We note that an elitism operator, wherein the best of a previous generation is automatically included in the next, was programmed into our GA; however, our limited testing suggests that elitism slowed convergence for our system.

As stated above, our problem is inherently embarrassingly parallel in that all of the expensive calculations, i.e., the many DFT fitness calculations involving 148 geometry optimizations, have no dependency between the tasks and can easily be distributed. Thus, a parallelized version of our GA is most important. While it is true that the ADFT code used in this study has been ported to use the MPI library for parallelization, the small size of the molecules in the extended G2 set along with the inherent speed of the ADFT method means that even single-processor calculations run rapidly enough that MPI is not useful in this study. Instead, we focus on parallelizing the Python portion of our code, specifically the evaluation of the fitness of a set of α values.

However, this is a challenge with Python. The Python interpreter is thread unsafe and by design includes a global interpreter lock (GIL) which prevents multi-*threaded* programming. That said, there are numerous modules that have been written for Python that allow for multi-*processor* programming. For our study we chose a recently developed package, Parallel Python [21], which provides a simple application programming interface (API) allowing for rapid construction and deployment of a parallelized Python program. It is simple to implement, works on varying architectures (as long as the ADFT code, which is written in Fortran 90 and uses MPI and Scalapack calls, can work on it), and works well with cluster computers. As currently written, our parallel genetic algorithm (PGA) is fully capable of utilizing multiple cores on multiple nodes of a cluster with very little overhead due to communication. Moreover, Parallel Python allows for autodiscovery of additional workers. Thus, if more computational power is freed up in a cluster, we have the option of starting more workers, speeding up the evaluation.

There are many ways to parallelize the GA code. Parallel Python works using a server-worker model, where there is a server process and many worker processes that do the actual calculation. One possible way to parallelize our problem, then, would be to submit a job for a set of α to each worker process to do one calculation for one molecule. For various developmental reasons, however, we chose a more coarse-grained algorithm sending to each worker a set of α values and have that worker then do all the atomic and molecular calculations for each set (totaling 163 calculations per set of α values) and report back the final optimized energies for each molecule. The GA then proceeds to calculate the errors against a database of experimental zero-point-exclusive atomization energies, and use this value as fitness to construct the next

generation as stated above. These steps occur only on the master node, but are so cheap computationally compared to the ADFT calculation that no attempt was made to parallelize them.

3. Results

The results of our optimization using our PGA are presented in Table 1. For the sets of α values due to our group, α_{EA} , α_{OPT} , and α_{PGA} , all calculations used what is referred to as the DGDZVP2/A2 basis set. The ADFT method requires four Gaussian basis sets: one for the orbital expansion and others to fit three powers of the electron density. For the orbital expansion we chose the D_{GAUSS} [24] valence double- ζ basis set (DGDZVP2) [25]. The s -type fitting bases were obtained by scaling and uncontracting the s part of the orbital basis as explained in previous work [8], and the pd -type fitting was from the A2 charge-density-fitting basis [25]. All other ADFT details for our current calculation are as detailed in previous work [8]. In addition to the GA parameters specified above, our code was run for 50 generations. Also, by the random nature of GA selection of initial values for the α , it is probable that α values will be selected that will not lead to convergent ADFT calculations for certain molecules. In these cases, our GA sets the energy of that molecule to 50 hartrees. This has the effect of weighting such α so that they are much less likely to participate in future generations.

As we see from Table 1, the resulting parameters from our PGA, α_{PGA} , are able to better the performance of that of previous work, α_{OPT} . The simplex algorithm values, α_{OPT} , led to a mean absolute error (MAE) of 14.5 kcal/mol and a mean signed error (MSE) of -4.5 kcal/mol. Our current PGA code resulted in a MAE of 13.8 kcal/mol and a MSE of 1.3 kcal/mol. Both optimized sets of α values are superior to the ‘non-optimized’ sets, α_{EA} , based upon reproducing the exact atomic energy [23], and α_{HF} , the historical choice of α giving HF atomic energies [22].

We note that the GA’s MSE value nearer zero is due to a more equal mix of both negative and positive signed errors for the molecules in the extended G2 set and is not a measure of fitness. Indeed, when we look at the largest errors of members of the extended G2 set for each algorithm, we find a similar spread. For the simplex algorithm, α_{OPT} , the largest signed errors were -70.4 kcal/mol for ClNO and -58.3 kcal/mol for ClF₃; for the GA, the largest signed errors were -59.8 kcal/mol for SiF₄ and 72.0 kcal/mol for benzene. In each case, the largest errors for each method are of similar size.

Table 1
The set of α values and MAE and MSE in atomization energies (in kcal/mol) from previous and present work

Element	α_{HF}	α_{EA}	α_{OPT}	α_{PGA}
H	0.77627	0.781240	0.753703	0.760819
Li	0.77157	0.792110	0.839295	0.570837
Be	0.76823	0.796140	0.557058	0.877363
B	0.76206	0.786770	0.674430	0.616307
C	0.75331	0.776650	0.675039	0.669355
N	0.74522	0.767260	0.645482	0.660067
O	0.74188	0.764480	0.657869	0.625517
F	0.73587	0.760010	0.575118	0.594423
Na	0.73115	0.752870	0.823779	0.539225
Al	0.72853	0.748690	0.743690	0.746373
Si	0.72751	0.746020	0.767716	0.537605
P	0.72620	0.743970	0.860982	0.813704
S	0.72475	0.743500	0.743745	0.714879
Cl	0.72325	0.742720	0.662447	0.671239
MAE	25.4	33.7	14.5	13.8
MSE	15.4	25.6	-4.8	1.3

The α_{HF} values are due to Schwarz (Ref. [22]), the α_{EA} values are from Ref. [23], the α_{OPT} values are from Ref. [8], and the α_{PGA} are from the current work. All calculations use the DGDZVP2/A2 basis set combination referred to in the text.

Although not used in the fitness evaluations for our GA, we have also calculated bond lengths of selected molecules in the extended G2 set using α_{PGA} . This selection consists of all molecules in the extended G2 set that are uni- or di-elemental with a single characteristic bondlength (i.e., one characteristic bond length in a Z-matrix). This is a superset of the bond lengths examined in our previous work [8] where we have used the same experimental values found therein with the additional values obtained from the experimental data section of the CCCBDB [26]. With this superset of bond lengths, α_{PGA} had a MAE of 0.05 Å and a MSE of 0.03 Å which compares favorably with an MAE of 0.04 Å and an MSE of -0.01 Å when α_{OPT} is used against the superset.

Finally, since our method is a *parallelized* GA, we also report some preliminary timings for our code. The code was run on a cluster consisting of nodes with dual-core Intel Xeon CPUs running at 3.20 GHz with 4 GB of RAM and a SATA/150 scratch disk per node. In all cases, both cores in each node were used. For the results presented in Table 1, eight nodes (=sixteen cores) were used taking 67.5 h to complete 50 generations. To test the scalability, calculations were also run using four and sixteen nodes leading to calculation times of 120.7 h and 47.9 h. These calculations were identical in setup, consisting of 1600 calculations of 148 molecules in total. We note that the random nature of the initiation and execution of a GA means the runs cannot be identical beyond the initial settings. Nevertheless, each gave consistent MAE results, 13.9 ± 0.2 kcal/mol, demonstrating the robustness of the method.

4. Discussion

The parallel genetic algorithm (PGA) provides several benefits over the simplex optimizer, the most important of which is scalability. Our current set of elements contains hydrogen and the thirteen first- and second-row elements of the periodic table, save the inert gases and magnesium. As the number of elements we look at increases, the size and complexity of the search ‘hypersurface’ to be optimized will need to work on increases dramatically. This also increases the chance that a simplex optimizer will get trapped in less-optimal minima rather than finding better, more global minima. Optimizers like GAs are specifically designed to work with more intractable search spaces such as this. As our PGA starts with a random selection of α , the GA is able to explore a greater range of the space. And, by judicious use of the mutation operator, our GA is less likely to get trapped in local, less-optimal minima. This property is amply demonstrated in Table 1 where we see that some of the α_{PGA} values are different from the α_{OPT} values by 50% or more, a trend seen in all PGA optimizations run. Obviously, a robust optimizer is needed to explore these areas which would be out of reach for a simplex optimizer.

There are, of course, cons to the use of a GA compared to an intelligently-devised simplex algorithm. The simplex algorithm devised by Zope and Dunlap [8] optimizes over one element at a time and works toward a minimum self-consistently until convergence. The advantage of such a method is that if one is currently optimizing only the α for boron, one need only to calculate properties of boron-containing molecules in the test set. In datasets such as the extended G2 set, which are obviously weighted toward important organic molecules, this can save calculation time. Likewise, intelligent selection of a starting set of α values can also speed up convergence.

In contrast, during the early generations of a GA, there will be many sets of α values that are patently bad due to the random starting set. If one of these sets of α values was, say, expressly wrong for carbon, then all carbon-containing compounds in a test set will be calculated even though the problematic set of α values will have little chance of being selected for mating. However, a

strength of GAs is that there is still a chance for inclusion in mating for these bad results leading to the more diverse nature of the optimizing. And, we note that use of increased computational power together with our parallel genetic algorithm, can alleviate some burden brought on by less useful sets of α values.

Despite the success of our PGA in finding sets of α values that are (slightly) better than our previous efforts, there are still improvements that can be made to our code. The most obvious is changing the parallelization of our PGA to improve the timing results presented above. We can better load balance our worker processes by instead parallelizing over the molecular calculations themselves rather than the sets of α values. That is, develop a fine-grained algorithm wherein we loop not over the population itself, but within the population. This way we can build a queue of jobs that Parallel Python can distribute as previous DFT calculations conclude. While there is still a possibility of many of the worker processes having to do calculations with bad α , we would remove the problem of having idle workers waiting for work to do. This could also have the effect of improving the scaling of our PGA as the number of processes increases over that seen in our limited tests reported above.

Beyond these algorithmic details, we also need to more fully explore the effects of altering the various settings of the GA on the convergence performance. For example, we currently use low mutation rates, no elitism, and fan selection in our code. But, it is possible that due to the large search space of our problem, that using high mutation along with some elitism (i.e., carry the best one or two individuals unchanged into the next generation) could provide an alternative—but just as effective—performance. Likewise, the use of a hybrid approach using a local minimizer upon each GA solution could also boost performance.

The ultimate goal of this research is the development of an optimizer with the ability to extend the use and power of ADFT. With our PGA, we are now in a position to incorporate more databases and properties, more elements, and more parameters into creating a more effective ADFT XC potential.

The most obvious next step is the inclusion of third-row and heavier elements into our calculations. By adding these new elements we will gain access to even larger benchmark databases such as Curtis' G3/05 [27] and the Truhlar group's transition metals databases [28–30]. The simplest and first integration will be the additional third-row atomization energies provided in these databases.

However, analysis of our current results shows the need to expand even our current first- and second-row test databases. In four complete PGA runs, elements that are better represented in the test set—C, N, H, O—tend to have more consistent α values over the different runs as shown in Fig. 1. Elements that are less well-represented in the extended G2 set—Be, P, Si, Na—have considerably more deviation in their values of α_{PGA} . Also, we note that while Al, B, and Li seem to have consistent α values, we have no confidence that this is not just an artifact of the limited set of runs completed. In fact, one of the second-row atoms for which we currently have basis sets constructed for use in our ADFT code, magnesium, is not represented at all in the extended G2 set. Additional compounds could be easily added following established procedures [31,32] used to develop current databases.

A final step to take is to use the current code to test and find α which are appropriate for a given property. For example, the Truhlar group has developed many databases [29,33–41] that are currently accessible with the elements optimized in this code. Integration of these databases and their concomitant properties (e.g., atomic energies, ionization potentials, proton affinities, etc.) will be one of our next steps in the research. Moreover, ongoing improvements in the ADFT code could open up even more of the Truhlar group's databases [28,42–50]. Likewise, we could also

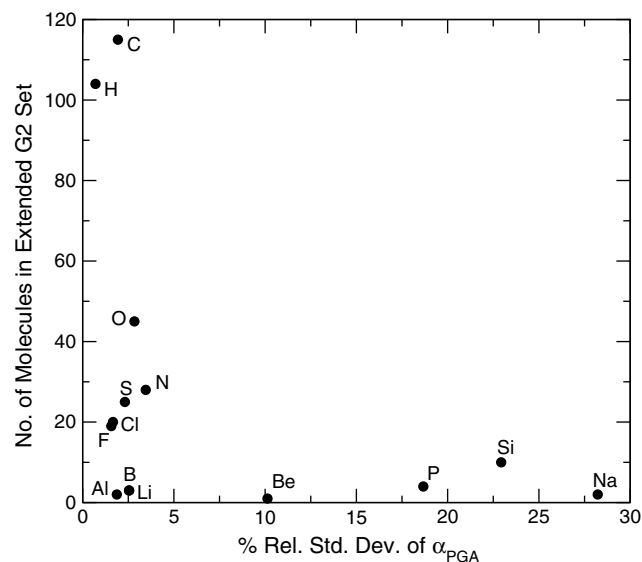


Fig. 1. The percent relative standard deviation in elemental α_{PGA} values over four runs compared to the number of molecules containing that molecule in the extended G2 set. The labels identify the specific elemental α_{PGA} %RSD.

use our algorithm to focus on development of sets of α values specific to certain types of molecules, e.g. aromatics, peptides, etc.

In any case, whether we expand the size of the test set involved, increase the scope of the elements we include, or consider other properties, it is crucial that a robust, expandable algorithm such as our parallel genetic algorithm is used as the problem becomes intractable for the usual methods. We feel this is the only way that a problem that demonstrably *must* get larger can be tackled in a timely, efficient manner.

Acknowledgements

This research was sponsored by the Office of Naval Research, directly and through the Naval Research Laboratory. M.A.T. gratefully acknowledges an NRC/NRL Postdoctoral Research Associateship. We thank Dr. Rajendra Zope for initial scripts, databases, and ongoing discussions, Vitalii Vanovschi for his Parallel Python software and his discussions and prompt fixes for issues encountered with integrating PP with a PBS cluster, and Dr. Zhao Yan for his discussions on the Minnesota Thermochemistry and Thermochemical Kinetic Databases.

References

- [1] J.A. Pople, Rev. Mod. Phys. 71 (1999) 1267.
- [2] P. Hohenberg, W. Kohn, Phys. Rev. 136 (1964) B864.
- [3] W. Kohn, L.J. Sham, Phys. Rev. 140 (1965) A1133.
- [4] W. Kohn, Rev. Mod. Phys. 71 (1999) 1253.
- [5] J.P. Perdew, K. Schmidt, in: V. Van Doren, C. Van Alsenoy, P. Geerlings (Eds.), Density Functional Theory and its Applications to Materials, American Institute of Physics, Melville, New York, 2001, p. 1.
- [6] A.M. Köster, J.U. Reveles, J.M. del Campo, J. Chem. Phys. 121 (2004) 3417.
- [7] R.R. Zope, B.I. Dunlap, Phys. Rev. B 71 (2005) 193104.
- [8] R.R. Zope, B.I. Dunlap, J. Chem. Phys. 124 (2006) 044107.
- [9] L.A. Curtiss, K. Raghavachari, G.W. Trucks, J.A. Pople, J. Chem. Phys. 94 (1991) 7221.
- [10] L.A. Curtiss, K. Raghavachari, P.C. Redfern, J.A. Pople, J. Chem. Phys. 106 (1997) 1063.
- [11] L.A. Curtiss, P.C. Redfern, K. Raghavachari, J.A. Pople, J. Chem. Phys. 109 (1998) 42.
- [12] B.I. Dunlap, S.P. Karna, R.R. Zope, J. Chem. Phys. 125 (2006) 214104.
- [13] S.B. Trickey, J.A. Alford, J.C. Boettger, in: J. Leszczynski (Ed.), Computational Materials Science of Theoretical and Computational Chemistry, vol. 15, Elsevier, 2004, p. 171.
- [14] A.M. Köster, J. Chem. Phys. 118 (2003) 9943.
- [15] B.I. Dunlap, R.R. Zope, Chem. Phys. Lett. 422 (2006) 451.

- [16] J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.
- [17] I. Rechenberg, *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Frommann-Holzboog, Stuttgart, 1973.
- [18] D.E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
- [19] G. van Rossum, *Python programming language* (1990).
- [20] A. Slowik, M. Białko, in: L. Rutkowski, J. Siekmann, R. Tadeusiewicz, L.A. Zaden (Eds.), *Artificial Intelligence and Soft Computing – ICAISC 2004*, vol. 3070/2004 of *Lecture Notes in Computer Science*, Springer Berlin, Heidelberg, 2004, p. 474.
- [21] V. Vanovschi, *Parallel Python software* (2008).
- [22] K. Schwarz, *Phys. Rev. B* 5 (1972) 2466.
- [23] R.R. Zope, B.I. Dunlap, *J. Chem. Theory Comput.* 1 (2005) 1193.
- [24] J. Andzelm, E. Wimmer, *Physica B* 172 (1991) 307.
- [25] N. Godbout, D.R. Salahub, J. Andzelm, E. Wimmer, *Can. J. Chem.* 70 (1992) 560.
- [26] NIST Computational Chemistry Comparison and Benchmark Database, September 2006.
- [27] L.A. Curtiss, P.C. Redfern, K. Raghavachari, *J. Chem. Phys.* 123 (2005) 124107.
- [28] N. Schultz, Y. Zhao, D. Truhlar, *J. Phys. Chem. A* 109 (2005) 4388.
- [29] Y. Zhao, D.G. Truhlar, *J. Chem. Phys.* 125 (2006) 194101.
- [30] J.M.L. Martin, *J. Chem. Phys.* 97 (1992) 5012.
- [31] P. Fast, N. Schultz, D. Truhlar, *J. Phys. Chem. A* 105 (2001) 4143.
- [32] J.M.W. Chase, *NIST-JANAF Thermochemical Tables*, fourth edn., Monograph No. 9, AIP, 1998.
- [33] B. Lynch, Y. Zhao, D. Truhlar, *J. Phys. Chem. A* 107 (2003) 1384.
- [34] Y. Zhao, N.E. Schultz, D.G. Truhlar, *J. Chem. Phys.* 123 (2005) 161103.
- [35] Y. Zhao, N. Schultz, D. Truhlar, *J. Chem. Theory Comput.* 2 (2006) 364.
- [36] Y. Zhao, D. Truhlar, *J. Phys. Chem. A* 109 (2005) 5656.
- [37] Y. Zhao, D. Truhlar, *J. Chem. Theory Comput.* 1 (2005) 415.
- [38] Y. Zhao, D. Truhlar, *J. Phys. Chem. A* 110 (2006) 10478.
- [39] Y. Zhao, D. Truhlar, *J. Chem. Theory Comput.* 3 (2007) 289.
- [40] Y. Zhao, D.G. Truhlar, *Theor. Chim. Acta* 120 (2008) 215.
- [41] P. Jurecka, J. Sponer, J. Cerny, P. Hobza, *Phys. Chem. Chem. Phys.* 8 (2006) 1985.
- [42] Y. Zhao, D. Truhlar, *Org. Lett.* 8 (2006) 5753.
- [43] E. Izgorodina, M. Coote, L. Radom, *J. Phys. Chem. A* 109 (2005) 7558.
- [44] H. Woodcock, H. Schaefer, P. Schreiner, *J. Phys. Chem. A* 106 (2002) 11923.
- [45] Y. Zhao, N. Gonzalez-Garcia, D. Truhlar, *J. Phys. Chem. A* 109 (2005) 2012.
- [46] Y. Zhao, B.J. Lynch, D.G. Truhlar, *Phys. Chem. Chem. Phys.* 7 (2005) 43.
- [47] Y. Zhao, D. Truhlar, *J. Phys. Chem. A* 110 (2006) 13126.
- [48] N. Schultz, Y. Zhao, D. Truhlar, *J. Phys. Chem. A* 109 (2005) 11127.
- [49] Y. Zhao, D.G. Truhlar, *J. Chem. Phys.* 124 (2006) 224105.
- [50] F. Furche, J.P. Perdew, *J. Chem. Phys.* 124 (2006) 044103.