

DL-FIND

A Geometry Optimiser for Atomistic Simulation Codes

Project Plan and User Manual

Revision: 393
Johannes Kästner,
STFC Daresbury Laboratory,
November 10, 2009

Note: the interface to the calling program is documented in section 3.1 on page 5.

1 General aim and target

DL-FIND is a modern and flexible structure optimiser to be included in electronic structure codes. It provides a stable way to find reaction energy differences as well as activation barriers. This can be performed starting from input structures in the region of the reactant and the product. The whole process should require as little action from the user as possible. The code should parallelise at least the energy and gradient evaluations and possibly expensive calculations within the optimiser. All units write restart information at regular intervals to enable a restart of the optimiser.

This version of DL-FIND incorporates an MPI split-communicator taskfarming parallelisation. Such taskfarming is only employed in the parallel optimisers at present, but could easily be used more widely.

A light-weighted description of some capabilities of DL-FIND can be found in the 2007 issue of *Frontiers*:

http://www.cse.scitech.ac.uk/ccg/j.kaestner/pdf/DLFIND_frontiers.pdf.

1.1 Functionalities

Emphasised: remains to be implemented

1.1.1 Coordinate systems:

- Cartesians (including frozen atoms and components)
- Mass-weighted cartesians (including frozen atoms and components)
- Internals (including all constraints):
 - DLC (delocalised internal coordinates)
 - DLC-TC (total connection)
 - HDLC
 - HDLC-TC

- *Fractional coordinates and unit cell optimisation?*
- *Parallel optimisation in internal coordinates?*

A microiterative optimisation in a QM/MM context should be available as well as a microiterative TS search. Those can eventually be combined.

1.1.2 Combinations of coordinates (images):

- NEB (nudged elastic band)
- *(Growing) String method*
- Dimer method [1–3]
- *Replica path method following [4]*

All of the combinations should work with all versions of coordinate systems.

1.1.3 Optimisers:

- steepest descent
- conjugate gradient
- Damped dynamics
- L-BFGS
- P-RFO Hessian update mechanisms: Powell [5] and Bofill [6]. Hessian either by input or by finite-difference. In the latter case either in cartesians (then the update also in cartesians, and one can output frequencies), or in internals.

A criterion can be specified with absolute eigenvalues of the hessian below that criterion are frozen (considered to be “soft”). At maximum 6 eigenmodes are frozen.

- Random (stochastic) search [7, 8]
- Genetic algorithm [9–11]
- *Reaction path following / IRC ?*

1.1.4 Line search algorithms:

- Simple scaling of the proposed step (covering the maximum step length)
- Trust radius based on energy decrease
- Trust radius based on the projection of the gradient on the step
- *Trust radius based on the overlap of the lowest eigenmode (for P-RFO)*
- *full line search ? which algorithm?*

The design allows new methods to be easily implemented.

1.1.5 A global task manager:

Should define which methods are used depending on the input. In case a method fails, this should be recognised and another method should be tried.

1.1.6 Parallelisation:

The energy evaluations and all relevant parts of the optimiser should be parallelised.

The parallel optimisers (random search and genetic algorithm) can be run in task-farming mode using MPI (message passing interface), where each taskfarm calculates the energy and gradient for a non-overlapping subset of the current population. The load-balancing is static, and the number of taskfarms must be a factor of the total number of processors for a given job. If the number of taskfarms requested is less than the total number of processors, then the single-point energy and gradient calculations for each individual in a taskfarm can also be parallelised. This is handled by the program that provides the energy and gradient routines. Thus, two-way and two modes of one-way parallelisation are possible, in theory.

Wrappers for the required MPI routines are located in the file `dlf_mpi.f90`, and the corresponding “dummy” subroutines are in `dlf_serial.f90`. Which of the corresponding .o object files is linked should depend on the build option chosen. For example, the default build of the standalone DL-FIND with its test driver program gives a serial executable. A `make parallel` command will produce the parallel-enabled standalone executable, `Pfind.x`. See `makefile` and `Makefile.standalone` for details.

Notes:

- If DL-FIND is in charge of disentangling the standard output from all the processors, then the variable `keep_alloutput` in `dlf_global_module.f90` is used. Rather than being an input option, it is hardwired in the code (so that it can be known before any output occurs). For example, `keep_alloutput` is used in `main.f90` (when a standalone program is made for testing) in the interface subroutine `dlf_output` as follows: if true, then for processor n ($n \neq 0$) in the global communicator (`MPI_COMM_WORLD`), the file `output.proc< n >` is opened on unit `stdout`; if false, then for processor n ($n \neq 0$), `/dev/null` is “opened” on unit `stdout`. Output from the rank-zero processor in either case goes to standard out or a named file on unit `stdout`.

If such I/O issues are dealt with by the main, calling program, then simply use the subroutine `dlf_output` to change the DL-FIND defaults (`stdout` and `stderr` stored in `dlf_global_module.f90`) for the required unit numbers, if necessary.

- Integer variables `mpi_rk` and `mpi_ik` are declared in `dlf_mpi_module` and set in subroutine `dlf_mpi_initialize` to the values of the MPI datatypes `MPI_Double_precision` and `MPI_Integer`, respectively. This matches the use of `real(rk)` in DL-FIND’s declarations, where `rk = kind(1.d0)`. However, problems will arise if a compiler flag is used to change the nature of double precision numbers, as the MPI libraries will (probably) have been compiled with a different specification for double precision. Could use `MPI_Sizeof` and `MPI_Type_match_size` instead if this turns out to be a common problem.

- Random numbers are an integral part of the parallel optimisers. Therefore, the seeding of the random-number generator is dealt with in `dlf_parallel_opt.f90`. Current policy is that only the rank-zero processor generates random numbers for the parallel optimisers. Only the routines that manipulate the population need random numbers, so currently only the rank-zero processor does any work in such routines.

1.1.7 Restart mechanism:

The optimiser should be fully restartable. Status: everything is restartable.

1.1.8 Compilation:

The portland compiler version 7.1 and 7.1-1 (and 7.1-2) do not compile the code properly. They do not save the contents of the `hdlc` derived type. This is a compiler bug. The portland compilers 6.1 and 7.0-4 do compile it.

1.2 Time Line

A detailed time line for the DL-FIND project is not envisaged. There are, however, some milestones:

- Working implementation of NEB and the dimer method in GAMESS-UK and ChemShell. To be completed by 31/03/2007. Completed.
- Implementation of an exhaustive test set. To be completed by 30/09/2007.
- Implementation of a global task manager to enable automated optimisation. To be completed by 31/03/2008.

2 Quality Assurance Plan

The software should compile on the following compilers: `g95`, `ifort`, `gpf95`, `gfortran`, as well as `xlF95`. It should pass standard Fortran check programs (*TODO: to be specified*). For the time being, these are to be checked at regular intervals by the author (J. Kästner).

3 Software Design Plan

Language: Fortran 95 + TR 15581 (technical report: meaning, allocatable arrays in derived types are used). Pointers are only to be used where strictly necessary.

3.1 Interface to the Calling Program

DL-FIND is designed as a library to be linked to quantum chemical or MD codes. However, for testing purposes, a driver module (`main.f90`) providing some analytic energy functions is used. DL-FIND is included in GAMESS-UK and ChemShell.

The routine to be called by the main program is `dl_find(nvarin,nvarin2,nspec,master)`, in the file `dl_find.f90`. It only returns after the complete geometry optimisation.

The main program has to provide the following routines, which are called from `dl_find.f90` and `dlf_formstep.f90`:

- `dlf_get_params(...)` Provide input parameters. The argument list is expected to change as DL-FIND is developed. New arguments should only be appended to existing ones which makes it possible to keep interfaces up to date. The arguments are documented in `dlf_global_module.f90`.
- `dlf_get_gradient(nvar,coords,energy,gradient,iimage,status)` Calculate the energy and gradient at the position `coords`.
- `dlf_get_hessian(nvar,coords,hessian,status)` Calculate the Hessian at the position `coords`.
- `dlf_get_multistate_gradients(nvar,coords,energy,gradient,coupling,needcoupling,iimage,status)` Calculate gradients of multiple (currently implemented: 2) electronic states. Required for conical intersection search.
- `dlf_put_coords(nvar,switch,energy,coords,iam)` Feeds a geometry back to the calling program. If `switch` is 1, `coords` contains the actual geometry. If `switch` is 2, `coords` contains the transition mode. The presence of `iam` allows the behaviour on the rank zero processor to be coded differently: for example, turning off the writing of coordinates to a file from all but the rank-zero processor.
- `dlf_error()` Error termination. Return to the calling program. `dlf_error` should not return.
- `dlf_update()` Allows the calling code to update any neighbour list, i.e. allows for discontinuities in the potential energy surface. This routine is called after a reset of the optimisation algorithm.

These may be C routines but have to be Fortran-callable. Examples of interface routines are available in `main.f90`, `dlf.c`, and `dlfind.gamess.m`.

If the taskfarming functionality is required, then the following additions to the main program may also be made:

- call `dlf_mpi_initialize()` to either have DL-FIND set up the global MPI communications, including calling `MPI_Init`, or to get the required parameters for the global communications that have already been set up in the main program. In the former case, this call should be placed before any I/O occurs, as close to the start of the main program as possible, as usual with `MPI_Init`. If the main program sets up the communications, then this call should be made after the rank of each processor and the total number of processors are known (ideally, immediately afterwards). This call is an obligatory addition to the main program.

- call `dlf_mpi_finalize()` to close down MPI before exiting the main program. Not necessary if the main program calls `MPI_Finalize` already; can be added safely after such a call in the main program (but is redundant).
- call `dlf_output(dlf_stdout, dlf_stderr)` passing unit numbers for I/O from the main program to DL-FIND. The call may be omitted if it is not necessary to change the defaults set in `dlf_global_module.f90`, and if the main program deals with the output from different processors.
- provide subroutine `dlf_output(dlf_stdout, dlf_stderr)`, if required, to set the DL-FIND variables `stdout` and `stderr`. If necessary, implement the strategy for dealing with standard output from the different processors here.
- provide subroutine `dlf_put_procinfo(dlf_nprocs, dlf_iam, dlf_global_comm)`. Called from `dlf_mpi_initialize` if `MPI_Init` was called there. Passes the total number of processors, the rank of the current process and a variable set to the handle for the global communicator (`MPI_COMM_WORLD`) to the main program.
- provide subroutine `dlf_get_procinfo(dlf_nprocs, dlf_iam, dlf_global_comm)`. Called from `dlf_mpi_initialize` if `MPI_Init` had already been called by the main program. The total number of processors, the rank of the current process and a variable set to the handle for the global communicator (`MPI_COMM_WORLD`) are passed from the main program to DL-FIND.
- provide subroutine `dlf_put_taskfarm(dlf_ntasks, dlf_nprocs_per_task, dlf_iam_in_task, dlf_mytask, dlf_task_comm, dlf_ax_tasks_comm)`. Called from `dlf_make_taskfarm` if DL-FIND sets up the split communicators. Passes to the main program the number of farms, the number of processors per farm, the rank of the current process in its farm, the rank of the current process's farm, and variables containing the handles for communicators within each farm and for the rank-*n* processor in each farm. The main program indicates to DL-FIND which of the two should set up the split communicators via an argument to the general interface routine `dlf_get_params`. `tdlf_farm = 0` means the main program does the setup; `tdlf_farm ≠ 0` means DL-FIND does.
- provide subroutine `dlf_get_taskfarm(dlf_ntasks, dlf_nprocs_per_task, dlf_iam_in_task, dlf_mytask, dlf_task_comm, dlf_ax_tasks_comm)`. Called from `dlf_make_taskfarm` if the main program sets up the split communicators. The number of farms, the number of processors per farm, the rank of the current process in its farm, the rank of the current process's farm, and variables containing the handles for communicators within each farm and for the rank-*n* processor in each farm, are all passed from the main program to DL-FIND. The main program indicates to DL-FIND which of the two should set up the split communicators via an argument to the general interface routine `dlf_get_params`. `tdlf_farm = 0` means the main program does the setup; `tdlf_farm ≠ 0` means DL-FIND does.
- ensure either `MPI_Abort` or `dlf_mpi_abort()` is called from the `dlf_error()` subroutine.

Recipies of how to include DL-FIND in ChemShell and GAMESS-UK are in the file `building_chemshell_with_dl-find`.

3.2 Program Units and Their Contents:

3.2.1 Modules

The code makes use of four modules that can be used from any subroutine throughout the DL-FIND: `dlf_parameter_module`, which only provides the real kind `rk`. It is located in the file `dlf_stat_module.f90`. `dlf_checkpoint`: variables and subroutines for reading and writing checkpoint files. `dlf_stat`: statistics. May be deleted and replaced in the future. `dlf_global`: contains many global parameters and also arrays (Cartesian coordinates, internal coordinates, the step, ...). All those parameters are part of a variable `glob`, which has a derived type (also defined in the module `dlf_global`). `pi`, `stdout`, and `stderr` are defined there as well.

All other modules must only be used within the file they are defined! If data of these module should be provided to other files (units), `get-` and `set` routines must be used!

3.2.2 Main Units:

- Main Unit (`dl_find.f90`)
- Convergence tester
- Coordinate transform
- Optimisation algorithms (`dlf_formstep.f90` and routines called from there)
- Scalestep: line search and trust radius approaches
- Utility units

3.2.3 Other details:

File units that are used longer than for an immediate write (> 1000): `1001 – 1001+nimage,max 1050` for xyz of NEB (`dlf_neb.f90`)

When arrays are allocated, the variables `glob%storage` and `glob%maxstorage` should be adjusted accordingly to enable control over the memory usage.

3.3 Program Documentation

The documentation is done using the `robodoc` tool. Each (important) subroutine should have an entry marking its connection within the code, the input and output variables of the global module, and its main purpose. An example file is `dlf_dimer.f90`.

Details of the documentation:

Fortran subroutines are to be documented as functions. The header gives the unit and the full subroutine name “NAME” should not be specified “SYNOPSIS” should be the actual code statement beginning the subroutine.

Module header names should be something short and descriptive rather than the actual Fortran name.

4 Code fragments obtained from external sources

Part of `dlf_lbfgs.f90` were obtained from <http://www.ece.northwestern.edu/~nocedal/lbfgs.html>.

The HDLC part has been taken from the HDLC optimiser in ChemShell. These are the files: `dlf_hdlc_constraint.f90` `dlf_hdlc_hdlcplib.f90` `dlf_hdlc_interface.f90` `dlf_hdlc_matrixlib.f90` and `dlf_hdlc_primitive.f90`. Walter Thiel agreed to make them publically available (even under GNU license), as long as he gets DL-FIND for the MNDO code:

[...] I have included the coordinate transformation part of the HDLC optimiser into DL-find. I expect HDLCs to work with the dimer method, and maybe they even work with NEB. I hope there are no objections from you as long as it only goes into ChemShell and gamess. However, I would like to make it available more broadly. It would be interesting to include it into other Daresbury codes (I mainly think of DL_POLY and possibly Crystal – if those people are interested, maybe the solid state people (around Walter Temmerman) are also interested). At a later stage, I think it would also be worth making DL-find with the HDLC coordinate transform available under the GNU license. Alternatively, I would have to make a DL-find version that runs without HDLCs.

What do think about that? I.e. would you give us (DL) permission to distribute part of the HDLCopt code more widely?

Dear Johannes,

all this is fine with me. It is a good idea to make HDLCopt available to a wider audience. In return, I would like to include DL-find in the MNDO program (with no restrictions concerning its distribution). I had planned to add other optimisers to MNDO anyway, and it would obviously be good not to duplicate such work.

Best wishes and Happy New Year,

Walter Thiel

5 Test Plan

The driver program will enable tests of all functionalities with analytic energy function. Should noisy gradients be tested as well, a random part can be added to the gradient. Test cases from analytic 2-dimensional potentials showing minima and transition states (e.g. the Müller–Brown potential [12]) as well as clusters of Lennard–Jones particles will be used. The latter can be extended to systems with very many degrees of freedom.

6 Method documentation

This section documents non-standard implementations in DL-FIND. Published standard methods are not documented here.

6.1 Weights

A list of weights (one for each atom) can be specified in the input array `coords2`. It will be remapped onto weights of each degree of freedom (internal coordinates) to be optimised.

It can be used to restrict the NEB path to a certain set of atoms, or the direction of the dimer.

7 Documentation of the Input Options – User Documentation

System documentation is available in the source code after the subroutine headers. The input options will be explained here (User Documentation).

Print level (Variable `print1`):

0 no printout

2 print something

4 be verbose

6 debug

Type of coordinate system (Variable `icoord`):

“unit place” means `icoord` modulo 10

0–9 The whole system is to be treated as one image

Unit place 0 Cartesians

Unit place 1 HDLC - internals [13]

Unit place 2 HDLC - TC [13]

Unit place 3 DLC - internals [13]

Unit place 4 DLC - TC [13]

Unit place 5 Mass-weighted Cartesians (\sqrt{mx}) – will be deleted: use `glob%massweight` for that now!

1X *Lagrange-Newton conical intersection search, with two extra coordinates corresponding to the gradient difference vector and interstate coupling gradient constraints.*

10X NEB with endpoints free

11X NEB with endpoints moving only perpendicular to their tangent direction

12X NEB with frozen endpoints.

13X NEB with endpoints free. Only initialisation in coordinates X, optimisation in cartesians.

14X NEB with endpoints moving only perpendicular to their tangent direction. Only initialisation in coordinates X, optimisation in cartesians.

15X NEB with frozen endpoints. Only initialisation in coordinates X, optimisation in cartesians.

The NEB version implemented is the “improved-tangent” NEB [14] (also called “upwind scheme”) with a climbing image.

20X Dimer method [1, 2]. Translation and rotation of the dimer are covered by the optimiser specified through `iopt`. Requires two energy evaluation per iteration.

21X Dimer method. Rotation of the dimer is done by a line search within the dimer module, two energy calculations are used per rotation. Requires at least two energy evaluation per iteration.

22X Dimer method. Rotation of the dimer is done by a line search within the dimer module, one energy calculation is done per iteration, the other one is interpolated. Requires at least two energy evaluation per iteration.

In all dimer versions: If a second set of coordinates is provided, it determines the dimer direction, if not, the dimer direction is randomised.

Multistate calculations (Variable `imultistate`):

0 Single state calculation (default)

1 Conical intersection optimisation (penalty function algorithm).

2 Conical intersection optimisation (gradient projection algorithm).

3 *Conical intersection optimisation (Lagrange-Newton algorithm).*

Type of optimisation algorithm (Variable `iopt`):

- 0 Steepest descent
- 1 Conjugate gradient following Polak–Ribière [15] (with automatic restarts based on the criterion by Powell and Beale) that is not coded properly ...
- 2 Conjugate gradient following Polak–Ribière [15] with CG restart every 10 steps (hardcoded at the moment)
- 3 L-BFGS [16, 17]
- 10 P-RFO [18–21] A switching mechanism for the mode to be followed is included, but does not seem to help in any of the cases I tried so far.
- 11 Just calculate the Hessian and to a thermal analysis (harmonic approximation for entropy, ...) – implementation only started.
- 20 Newton–Raphson (mainly implemented to test Hessian-based methods)
- 30 Damped dynamics using the variables `timestep`, `fric0`, `fricfac`, and `fricp`. The frictions are defined that 0 corresponds to free (undamped) dynamics, and 1 corresponds to steepest descent.
- 51 Random (stochastic) search [7, 8], using the variables `po_pop_size`, `po_radius`, `po_contraction`, `po_tolerance_r`, `po_tolerance_g`, `po_distribution`, `po_maxcycle`, `po_scalefac`.
- 52 Genetic algorithm [9–11], using the variables `po_pop_size`, `po_radius`, `po_tolerance_g`, `po_maxcycle`, `po_init_pop_size`, `po_reset`, `po_mutation_rate`, `po_death_rate`, `po_nsave`

Type of line search or trust radius (Variable `iline`):

- 0 simple scaling of the proposed step, taking `maxstep` into account
- 1 Trust radius based on energy as acceptance criterion (recommended for L-BFGS optimisation)
- 2 Trust radius based on gradient as acceptance criterion (recommended for CG optimisation)
- 3 Hard-core line search. Does not work at the moment...

Type of initial Hessian (Variable `inithessian`):

- 0 Calculate externally using `dlf_get_hessian`. Defaults to two point finite difference if an external Hessian is unavailable.
- 1 Build by one point finite difference of the gradient
- 2 Build by two point finite difference of the gradient
- 3 Build a diagonal Hessian with a single one point finite difference
- 4 Set the Hessian to be an identity matrix

Hessian update mechanism (Variable update):

- 0** No update. Always recalculate the Hessian
- 1** Powell update [5]
- 2** Bofill update [6]
- 3** BFGS update

Fragment and frozen atom information (Variable spec): The array `spec` has an entry for each atom. Meaning:

- >0** Fragment (residue) number the atom belongs to. Active in the optimisation.
- 0** Active, and treated in Cartesian coordinates, even if other atoms in the system are covered by (H)DLCs.
- 1** Frozen
- 2** x-component frozen (Cartesians only)
- 3** y-component frozen (Cartesians only)
- 4** z-component frozen (Cartesians only)
- 23** x and y-components frozen (Cartesians only)
- 24** x and z-components frozen (Cartesians only)
- 34** y and z-components frozen (Cartesians only)

Atoms with `spec < -1` will be completely frozen if used in HDLCs. One may use Cartesian constraints there to specify frozen components. Atoms with `spec > 0` will be free in Cartesian coordinates.

There are numerous other paramers that can be set via the routine `dlf_get_params`. These are at the moment explained in the global module, `dlf_global_module.f90`.

7.0.1 Restarting

`dump` after how many energy and gradient evaluations is a restart file (checkpoint file) to be written? Default: 0 (never).

`restart` 0: new run (default), 1: read all checkpoint files and start from those.

It is only possible to restart a job with most parameters equal to those in the checkpoint file. Exeptions (input parameters that may be different from the ones in the previous run, i.e. that are not written to the checkpoint file): `maxcycle`. If different parameters are to be used, the latest geometry should be used and the optimiser should be started from scratch (`restart=0`).

8 Collection ...

Force weighted internal coordinates: have a look at <http://www.molpro.net/molpro-user/archive/all/msg00071.html>

At the moment, I am experiencing problems with NEB in internal coordinates. While the structure may change continuously, the internals may not. Consider H₂CO separation: it has two dihedrals. In the bound configuration, both are 180°. In the dissociated configuration, one is 0 and one is 180, as H₂ moves to one side of CO. How to deal with that?

One possible solution: Improper are automatically put onto atoms with nearly planar configuration (what if more than 3 connections?). If an improper is placed on an atom, this one should not be in the middle of a dihedral that include these atoms. This, however, had to be removed again as it leads to an underdetermined system when not all atoms next to the planar atom are monovalent.

svn properties:

```
svn propset svn:keywords "URL Author Date Rev Id" dlf_util.f90
```

References

- [1] G. Henkelman and H. Jónsson: A dimer method for finding saddle points on high dimensional potential surfaces using only first derivatives. *J. Chem. Phys.* **111**, 7010 (1999).
- [2] A. Heyden, A. T. Bell, and F. J. Keil: Efficient methods for finding transition states in chemical reactions: Comparison of improved dimer method and partitioned rational function optimization method. *J. Chem. Phys.* **123**, 224101 (2005).
- [3] J. Kästner and P. Sherwood: Superlinearly converging dimer method for transition state search. *J. Chem. Phys.* **128**, 014106 (2008).
- [4] H. L. Woodcock, M. Hodošček, P. Sherwood, Y. S. Lee, H. F. Schaefer III, and B. R. Brooks: Exploring the quantum mechanical/molecular mechanical replica path method: a pathway optimization of the chorismate to prephenate Claisen rearrangement catalyzed by chorismate mutase. *Theor. Chem. Acc.* **109**, 140 (2003).
- [5] M. J. D. Powell: . *Math. Prog.* **26**, 1 (1971).
- [6] J. M. Bofill: Updated Hessian matrix and the restricted step method for locating transition structures. *J. Comput. Chem.* **15**, 1 (1994).
- [7] S. H. Brooks: A discussion of random methods for seeking maxima. *Operations Research* **6**, 244 (1957).
- [8] R. Luus, and T. H. I. Jaakola: Optimization by Direct Search and Systematic Reduction of the Size of Search Region. *AIChE Journal* **19**, (1973).
- [9] J. H. Holland: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, (1975).

- [10] D. Goldberg: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, New York, (1989).
- [11] R. L. Haupt, and S. E. Haupt: Practical genetic algorithms. Wiley, New York, (1998).
- [12] K. Müller and L. D. Brown: Location of saddle points and minimum energy paths by a constrained simplex optimization procedure. *Theor. Chim. Acta* **53**, 75 (1979).
- [13] S. R. Billeter, A. J. Turner, and W. Thiel: Linear scaling geometry optimization and transition state search in hybrid delocalised internal coordinates. *Phys. Chem. Chem. Phys.* **2**, 2177 (2000).
- [14] G. Henkelman and H. Jónsson: Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points. *J. Chem. Phys.* **113**, 9978 (2000).
- [15] E. Polak and G. Ribière: Note sur la convergence de directions conjuguées. *Ref. Fra. Inf. Rech. Op.* **3**, 35 (1969).
- [16] D. C. Liu and J. Nocedal: On the limited memory BFGS method for large scale optimization. *Math. Program.* **45**, 503 (1989).
- [17] J. Nocedal: Updating quasi-Newton matrices with limited storage. *Math. Comp.* **35**, 773 (1980).
- [18] C. J. Cerjan and W. H. Miller: On finding transition states. *J. Chem. Phys.* **75**, 2800 (1981).
- [19] J. Simons, P. Jørgensen, H. Taylor, and J. Ozment: Walking on potential energy surfaces. *J. Phys. Chem.* **87**, 2745 (1983).
- [20] A. Banerjee, N. Adams, J. Simons, and R. Shepard: Search for stationary points on surfaces. *J. Phys. Chem.* **89**, 52 (1985).
- [21] J. Baker: An algorithm for the location of transition states. *J. Comput. Chem.* **7**, 385 (1986).