Energy minimization techniques

- The apparently simple task of minimizing the energy of a set of atoms under a given interaction model is a very common, yet surprisingly complex problem to solve efficiently. It occurs e.g. in wanting to obtain the equilibrium shape of a protein, the ground state configuration of an atom cluster, a minimum-energy configuration of a defect etc.

- The are of course a large variety of energy minimization techniques in numerical mathematics. For large sets of atoms, one has to require that the memory requirement of the method scales as O(N), which rules out many efficient techniques which require O(N^2) memory.

- There are (at least) four basic approaches which can be taken in minimizing the energy of a group of atoms:

  1. Monte Carlo simulation: as a crude simplification: “let the atoms jump around randomly and accept the lowest-energy states.”.

  - Can be good e.g. in finding the equilibrium coordination in a liquid
  - Not very efficient in finding the closest local minimum.

  2. MD simulation: Do an MD-simulation letting the temperature -> 0 K.

  - Can be made more efficient by setting all $\boldsymbol{v} = 0$ if the energy grows, or by setting $\boldsymbol{v}_i = 0$ if the force $\boldsymbol{f}_i$ is in the opposite direction to $\boldsymbol{v}_i$
  - Sometimes quite efficient in finding a local minimum
  - Sometimes also good tool to find a global minimum: simulate at high T first, cooling down in cycles.

  3. Conjugate gradients

  - Very efficient method to find a local minimum

  4. Genetic algorithm

  - Probably best method to find a global minimum from a random initial configuration

- In this lecture package I present conjugate gradients and genetic algorithms for searching for global minima.

Conjugate gradients

- The conjugate gradients method is not by itself in any way related to atomistic simulations, but is a general method to minimize an *N*-dimensional function $f(\boldsymbol{x})$, where f can be any function of points $\boldsymbol{x}$ in *N*-dimensional space [Numerical Recipes, 2nd ed. ch. 10]

  For N atoms we can write their coordinates $\boldsymbol{r}$ as a 3N-dimensional vector $\boldsymbol{x}$ of the form

  $$\boldsymbol{x} = (r_{1,x}, r_{1,y}, r_{1,z}, r_{2,x}, ...)$$

  The function $f(\boldsymbol{x})$ corresponds now to a classical or quantum mechanical potential energy function $V(\boldsymbol{r})$. Using this form we can use the conjugate gradient approach to minimize the energy of a set of atoms.

- In the conjugate gradients (CG) method the gradient (force) of the function is used as a help in finding the minimum. The gradient tells in which direction the function changes the most rapidly, so it is clear that it can help in deducing in which direction the energy can be minimized. For a potential energy function this gradient is simply the (negative value of) the force between atoms.

- An obvious, but *not very efficient* way to minimize the energy is to always move in the direction of the negative gradient. This is the so called **steepest descent method**, which for atoms can be described as follows:

  0. Start from point $\boldsymbol{r_0}$, set i=0

  1. Calculate $\boldsymbol{F_i} = -\nabla V_i(\boldsymbol{r_i})$ and $V_i(\boldsymbol{r_i})$.

  2. If $V_{i-1} - V_i < \varepsilon$, end

  3. Minimize $V(\boldsymbol{r}_i + \alpha \boldsymbol{F})$ with respect to the scalar quantity α, i.e. move downwards in the direction of the gradient.

  4. Set $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i + \alpha \boldsymbol{F}$ and $i = i + 1$

  5. Return to stage 1.

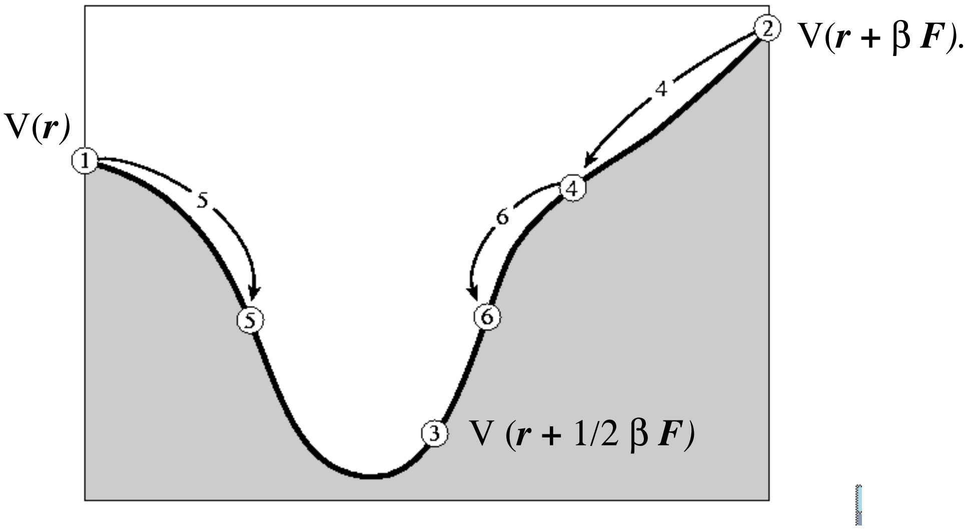
The algorithm somewhat resembles MD, but the chief difference is that there is no time, velocity or acceleration anywhere.

The line minimization in stage **3** is essentially a 1-dimensional operation in which the minimum of a function is sought by moving in a predetermined direction $\alpha \boldsymbol{F}$. Note that this is not the same thing as determining the minimum of the function, since it is very unlikely that ***F*** points to the minimum. The line minimization operation is a relatively straightforward operation which is carried out in two steps.

1. Make sure that there is a minimum and bracket it.

2. Search it with a given accuracy

Stage 1 is in principle easy to carry out. Starting from a point ***r*** and known direction ***F***, move forward some direction $\beta \boldsymbol{F}$. If $V(\boldsymbol{r} + \beta \boldsymbol{F}) > V(\boldsymbol{r})$ and in addition $V(\boldsymbol{r} + 1/2\, \beta \boldsymbol{F}) < V(\boldsymbol{r})$ and $V(\boldsymbol{r} + 1/2\, \beta \boldsymbol{F}) < V(\boldsymbol{r} + \beta \boldsymbol{F})$ the minimum is bracketed with the three points (*1*) $V(\boldsymbol{r})$ (*3*) $V(\boldsymbol{r} + 1/2\, \beta \boldsymbol{F})$ and (*2*) $V(\boldsymbol{r} + \beta \boldsymbol{F})$. If these criteria are not fulfilled, increase β and try again..

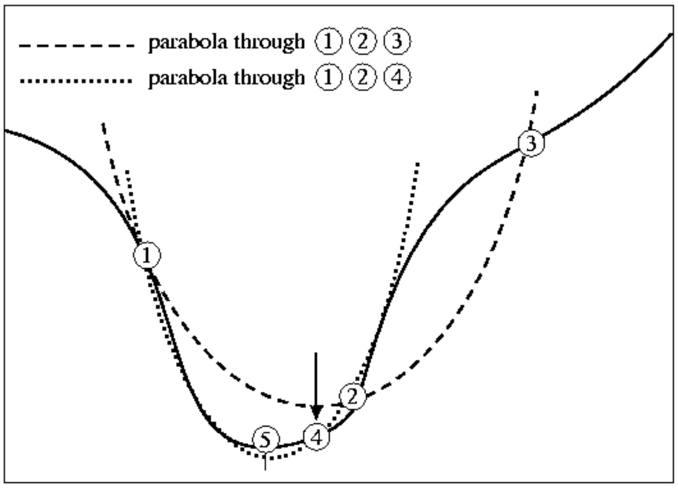


After the minimum has been bracketed, one could of course use ordinary binary search to find it. A slightly better method turns out to be to use a golden section, i.e. let the new minimum be 0.38197 from either end.

Often much better is to use so called inverse parabolic interpolation. In this method, a parabola is fit to the points *a, b* and *c* (corresponding to (1), (2) and (3) above), and the estimate of the minimum is the minimum of the parabola *x*. A bit of algebra shows that

$$x = b - \frac{1}{2}\frac{(b-a)^2[V(b)-V(c)]-(b-c)^2[V(b)-V(a)]}{(b-a)[V(b)-V(c)]-(b-c)[V(b)-V(a)]}$$

When the minimization is done once, either point *a* or *c* is replaced by point *x* (depending on which side of *b x* is), and the minimization step is repeated.



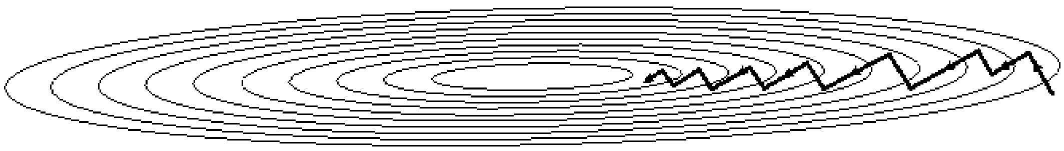
The iteration is continued until the minimum has been found with the desired accuracy.

- But this method can fail badly if the shape of the functional form is far from a parabola. In that case it is better to first use the golden section search, which always works. A combined method, which first tries the inverse parabolic search, but then reduces to the golden section if this fails, is probably the best general approach.

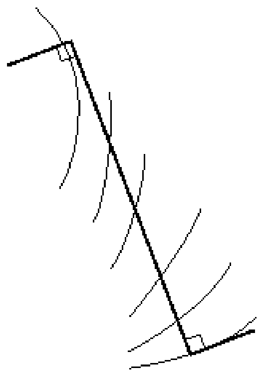
  One such method is the so called Brents method, which is presented in Numerical Recipes (program brent()).

- By combining the Steepest descent (SD)-algorithm and the Brent line minimization the energy of an atom system can be minimized. But this is still not very efficient in many dimensions. The reason is that the SD method easily winds up in a zig-zag pattern which does not move towards the minimum efficiently. This can be understood as follows.

  Imagine you walk in a long valley during a hiking tour in your favourite national park, and want to get to the bottom of the valley. If you are on one of the slopes of the mountains surrounding the valley, and walk down in the direction which goes most downwards (the direction of the gradient), you do get to the local bottom of the valley, but only slightly downwards along the length of the valley as you began. If you again chose the direction of the gradient to get down, you will start walking almost in the opposite direction of where you began for a short distance, then change direction again and so on. So you walk more in a zig-zag pattern than take the shortest route towards the minimum .



(a)



(b)

- In the Conjugate gradients (CG) method the problem is solved by choosing a new "conjugate" direction of movement so that it depends on the previous direction, and does not lead to the zig-zag-pattern above.

- What is really meant by two directions being conjugate to each other? Consider an arbitrary N-dimensional function $f(\boldsymbol{x})$, and construct its Taylor-series around a point $\boldsymbol{P}$:

$$f(\boldsymbol{x}) = f(\boldsymbol{P}) + \sum_i \frac{\partial f}{\partial x_i} x_i + \frac{1}{2} \sum_{i,j} \frac{\partial^2 f}{\partial x_i \partial x_j} x_i x_j + \ldots \approx c - \boldsymbol{b} \cdot \boldsymbol{x} + \frac{1}{2} \boldsymbol{x} \cdot \boldsymbol{A} \cdot \boldsymbol{x}$$

$$\text{where} \quad c \equiv f(\boldsymbol{P}) \qquad \boldsymbol{b} = -\nabla f\big|_{\boldsymbol{P}} \qquad \boldsymbol{A} = \left.\frac{\partial^2 f}{\partial x_i \partial x_j}\right|_{\boldsymbol{P}}$$

The matrix $\boldsymbol{A}$ is the so called Hessian matrix. In this approximation the gradient of f is

$$\nabla f = \boldsymbol{A} \cdot \boldsymbol{x} - \boldsymbol{b}$$

And a change in the gradient ∇f over some distance $\delta \boldsymbol{x}$ is again

$$\delta(\nabla f) = \boldsymbol{A} \cdot (\delta \boldsymbol{x})$$

Let us now assume that the previous direction in which we have moved is $\boldsymbol{u}$. We want to find a new direction $\boldsymbol{v}$ which takes us downwards as efficiently as possible. Since we have just performed a minimization in the direction $\boldsymbol{u}$, in the minimum the gradient $\boldsymbol{g}$ will be perpendicular to $\boldsymbol{u}$. We now want that for the new direction $\boldsymbol{v}$ the new gradient after the minimization, $\boldsymbol{g'}$, still is perpendicular to $\boldsymbol{u}$ -- this will prevent the zig-zag motion which is caused if we now would simply move in the direction of the gradient $\boldsymbol{g}$. This amounts to requiring that the change in the gradient $\delta(\nabla f)$ is perpendicular to $\boldsymbol{u}$. Thus we obtain from the previous equation the requirement

$$0 = \boldsymbol{u} \cdot \delta(\nabla f) = \boldsymbol{u} \cdot \boldsymbol{A} \cdot \boldsymbol{v}$$

If this is valid, the directions $\boldsymbol{u}$ and $\boldsymbol{v}$ are considered to be conjugated .

- In the conjugate gradients method two vectors $\boldsymbol{g}$ and $\boldsymbol{h}$ are used to calculate the new direction into which to move. $\boldsymbol{h}$ is the actual direction into which the line minimization is carried out.

  In solving linear equations, these are iterated as follows:

$$\boldsymbol{g}_{i+1} = \boldsymbol{g}_i - \lambda_i(\boldsymbol{A} \cdot \boldsymbol{h}) \quad \text{ja} \quad \boldsymbol{h}_{i+1} = \boldsymbol{g}_{i+1} + \gamma_i \boldsymbol{h}_i$$

  where

$$\lambda_i = \frac{\boldsymbol{g}_i \cdot \boldsymbol{g}_i}{\boldsymbol{h}_i \cdot \boldsymbol{A} \cdot \boldsymbol{h}_i} = \frac{\boldsymbol{g}_i \cdot \boldsymbol{h}_i}{\boldsymbol{h}_i \cdot \boldsymbol{A} \cdot \boldsymbol{h}_i} \quad \text{ja} \quad \gamma_i = \frac{\boldsymbol{g}_{i+1} \cdot \boldsymbol{g}_{i+1}}{\boldsymbol{g}_i \cdot \boldsymbol{g}_i}$$

  The vectors $\boldsymbol{g}$ and $\boldsymbol{h}$ fulfil the orthogonality and conjugation requirements:

$$\boldsymbol{g}_i \cdot \boldsymbol{g}_j = 0 \qquad \boldsymbol{h}_i \cdot \boldsymbol{A} \cdot \boldsymbol{h}_j = 0 \qquad \boldsymbol{g}_i \cdot \boldsymbol{h}_j = 0$$

  But now this is not yet enough, because we do not want to construct the N×N matrix $\boldsymbol{A}$ which would be required to get the new $\boldsymbol{g}$. The crucial, saving statement is the following: if we have just minimized f in the direction $\boldsymbol{h}$ to some point $\boldsymbol{x}_{i+1}$, the new $\boldsymbol{g}$ can be obtained simply with

$$\boldsymbol{g}_{i+1} = -\nabla f(\boldsymbol{x}_{i+1})$$

  and the end result corresponds to the above equations!

  In principle this iteration algorithm gets one to an energy minimum in a system of N atoms with a memory requirement $O(N)$ and a number of iteration steps $O(N)$. This sounds like a problem for large numbers of atoms: if say $N =$ 100000 we definitely do not want to iterate 100000 times. But in practice the atom motion in large systems is almost always strongly correlated, and much fewer iteration steps are enough to get to a minimum.

- My personal experience for atomic systems is that typically ~ 200 steps in periodic systems and ~ 1000 steps in systems with a surface is enough to find an energy minimum with 15 digits of accuracy regardless of system size.

- Using these equations we obtain the following algorithm for conjugate gradient energy minimization:

0. Start at point $\boldsymbol{r_0}$, set $i = 0$, $\boldsymbol{x_0} = -\nabla V(\boldsymbol{r_0})$, $\boldsymbol{g_0} = \boldsymbol{x_0}$, $\boldsymbol{h_0} = \boldsymbol{x_0}$

1. Minimize $V(\boldsymbol{r}_i + \alpha \boldsymbol{x_i})$ with respect to the scalar α (using Brents method), then set $\boldsymbol{r_{i+1}} = \boldsymbol{r}_i + \alpha \boldsymbol{x_i}$ and evaluate $V_{i+1}(\boldsymbol{r_{i+1}})$

2. If $V_{i+1} - V_i < \varepsilon$, **quit.**

3. Calculate $\boldsymbol{x_i} = \nabla V(\boldsymbol{r_{i+1}})$ and $V_i = V(\boldsymbol{r}_{i+1})$.

4. Calculate $\gamma = \dfrac{\boldsymbol{x}_i \cdot \boldsymbol{x}_i}{\boldsymbol{g}_i \cdot \boldsymbol{g}_i}$

5. Set $\boldsymbol{g_{i+1}} = -\boldsymbol{x}_i$

6. Set $\boldsymbol{h_{i+1}} = \boldsymbol{g_{i+1}} + \gamma \boldsymbol{h_i}$ and $\boldsymbol{x}_{i+1} = \boldsymbol{h_{i+1}}$

7. Set $i = i + 1$ and return to phase 1.

The above is the original, so called Fletcher-Reeves - algorithm. In some cases it is more efficient to use the so called Polak-Ribiere- version, which is identical to the above except that step 4 is:

4. Calculate $\gamma = \dfrac{(\boldsymbol{x}_i + \boldsymbol{g}_i) \cdot \boldsymbol{x}_i}{\boldsymbol{g}_i \cdot \boldsymbol{g}_i}$

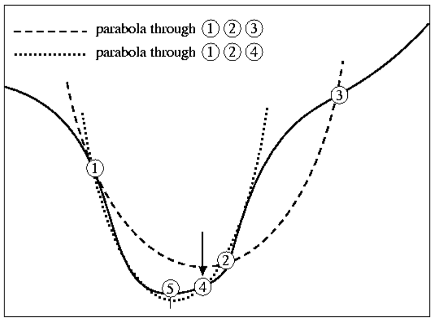
- The above algorithm is already a very efficient way to look for a local minimum. It also does not have anything specific to atomistic simulations; the function $V(\boldsymbol{r})$ can be any N-dimensional function $f(\boldsymbol{x})$ which has a well-defined gradient.

- In typical atomistic simulations there are special features (especially the knowledge that the atoms do have a smooth minimum) which can be utilized to optimize the algorithm, at the possible expense of generality.

- In atomistic simulations with modern potentials, not to mention quantum mechanical models, the calculation of the potential energy *V(**r**)* is very slow, and the calculation of forces even slower. In the above algorithm the line minimization-step **1** is the only step where forces are actually calculated. This step had two parts (see above):

  1. Make sure there is a minimum, and bracket it.

  2. Search it with the desired accuracy.

  The bracketing requires at least 3 evaluations of the potential, and the Brent method line minimization typically 5-10 evaluations. In addition, we still need to construct neighbour lists every now and then, which makes things even slower.

  In atomistic systems we know, however, that the length scale is rather limited. Unless the initial atom positions are really unphysical, the atoms are almost certain to be ~ 0.2 Å from the ground state position, or even closer. If we simply assume that the minimum is newer say farther than 0.5 Å, we can simply get rid of step **1.** But this is clearly a bit dangerous, and still does not gain us more than 20 % or so of the efficiency.

- It would be even better if we could get rid of the 5-10 potential evaluations needed in the Brent method. This can be achieved rather simply. So simply in fact that I invented the solution myself, but I am sure many other atomistic simulators have done the same numerous times.

  (A parenthesis on re-inventing the wheel. It is of course not a good idea to spend two months re-inventing the wheel, if you could get the answer by spending two days in the library. If, however, it only takes half a day to re-invent the wheel (like it took me in this case) it may at least be more fun than spending two days in dusty libraries).

- In this speed-up method (called ACG for reasons apparent below) we start by assuming that the ‘minimum is out there’. In atomic systems this is usually a reasonable assumption, unless we have purely repulsive potentials and at the same time open boundary conditions (in which case there is no sense in using CG in any case, as there is no minimum).

- The main point in the ACG method is my observation that when the original CG method line minimization of $V(\boldsymbol{r}_i + \alpha \boldsymbol{F}_i)$ with respect to the scalar α, for most steps the optimal value of the scalar α is about the same, ~ 0.05. (This is of course no natural constant, but seems to be valid for common Si and metal potentials). If the scalar α is almost the same in any case, it does not seem sensible to optimize it separately every time.

  So the method is as follows. Set initially $\alpha = 0.05$. For every step move forwards by $\alpha \boldsymbol{F}_i$. If the potential energy goes down, increase optimistically α a bit. If the potential energy goes up, disregard the previous step, decrease α and repeat the same iteration. Because of the optimization of α I call the method adaptive conjugate gradients, ACG.

- In algorithmic form we can write the ACG as:

0. Start from $\boldsymbol{r}_0$, set $i = 0$, $\boldsymbol{F}_0 = -\nabla V(\boldsymbol{r}_0)$, $\boldsymbol{g}_0 = \boldsymbol{F}_0$, $\boldsymbol{h}_0 = \boldsymbol{F}_0$, α=0.05

1. Store old $\boldsymbol{r}_i \to \boldsymbol{r}_i^{prev}$

2. Set $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i + \alpha \boldsymbol{F}_i$

3. Calculate $V_{i+1}(\boldsymbol{r}_{i+1})$ and $\boldsymbol{F}_i = -\nabla V(\boldsymbol{r}_i)$

4. If $V_{i+1} > V_i$: return $\boldsymbol{r}_i^{prev} \to \boldsymbol{r}_i$, set $\alpha = \alpha \times 0.5$, return to step 2.

5. If $V_{i+1} - V_i < \varepsilon$, **quit**.

6. Calculate $\gamma = \dfrac{(\boldsymbol{x}_i + \boldsymbol{g}_i) \cdot \boldsymbol{x}_i}{\boldsymbol{g}_i \cdot \boldsymbol{g}_i}$ (Polak-Ribierre)

7. Set $\boldsymbol{g}_{i+1} = -\boldsymbol{F}_i$

8. Set $\boldsymbol{h}_{i+1} = \boldsymbol{g}_{i+1} + \gamma \boldsymbol{h}_i$ and $\boldsymbol{F}_{i+1} = \boldsymbol{h}_{i+1}$

9. Increase $\alpha = \alpha \times 1.05$, set $i = i + 1$ and return to step 1.

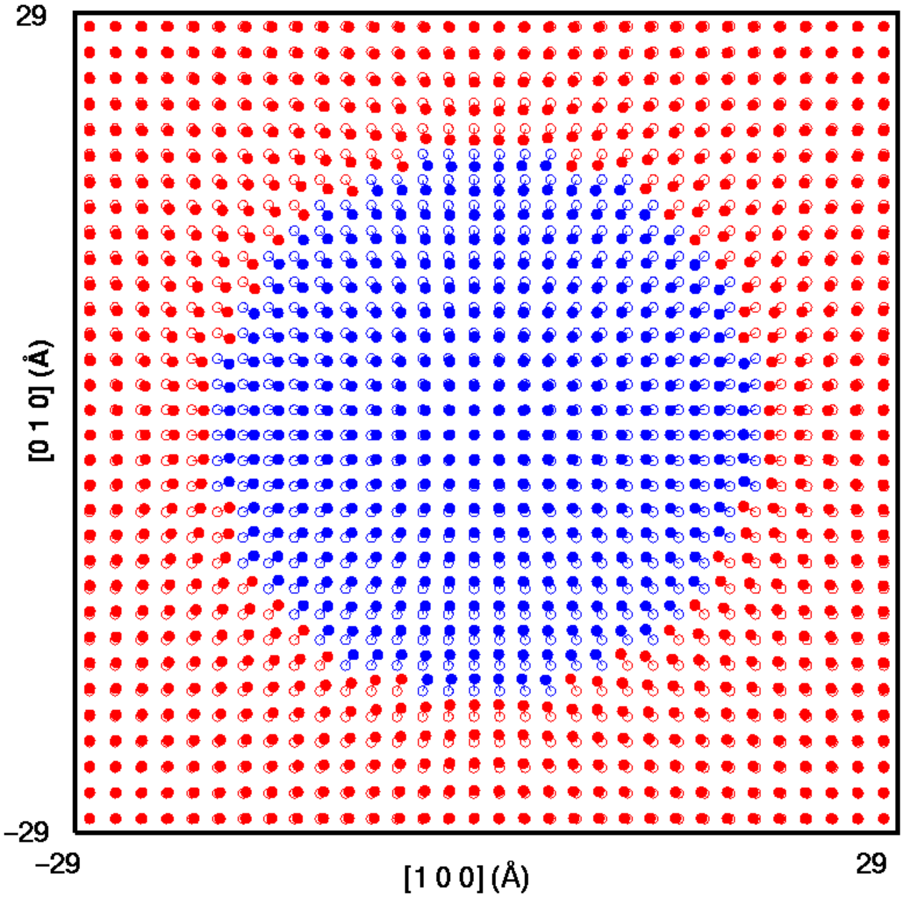
- The constants 0.5 and 1.05 were optimized for Stillinger-Weber Si.

- Written in this way the method usually needs only one potential evaluation per iteration step, except when the energy increases. In practice the energy decreases almost always, so on average the number of potential evaluations still is only about 1.1 / iteration. In the ordinary CG method this value is about 10, so in the ACG each iteration step is about 10 times faster than in CG!

  On the other hand, the ACG loses the perfect match of conjugate directions, so it needs more iterations. Still, in my experience the overall speedup of ACG vs. CG is almost always a factor of ~ 3-5.

- As an example I created 40 Å diameter Co-nanocluster in a 16×16×16 unit cell periodic Cu cell, and relaxed this system with EAM potentials with different methods. These calculations (in larger cells) are useful in understanding the energetics of Co nanoclusters.

  The adjacent figure shows the atom displacements due to the minimization, but so that the displacements have been exaggerated by a factor of 3. The open circles are the original atom positions, the closed circles the final positions after minimization.



- The blue atoms are Co, red Cu. Because Co has a smaller equilibrium nearest-neighbour distance than Cu, the atoms move inwards.

The simulation results were as follows (computer ~ 400 Mhz Pentium Linux):

(SD= Steepest Descent, CG = Conjugate Gradients, MD = Molecular dynamics. Et is the energy tolerance, Niter the number of iterations)

Method	Et (eV)	Niter	Final E (eV)	Simulation time (s)
SD Plain	0.001	227	-59927.160	2684.20
SD Adaptive	0.001	172	-59927.052	323.56
CG Plain	0.001	27	-59927.193	363.03
CG No bracketing	0.001	27	-59927.193	251.98
ACG = CG Adaptive	0.001	70	-59927.194	128.34
MD btctau=70 fs	-	250	-59927.169	390.25

We see that all methods give essentially the same result, as they should. The 0.1 eV differences may be shifts in the position of a single atoms, and hence not likely to be a significant problem.

- The SD method with line minimization is very slow, as expected. The number of iterations is clearly the smallest in the CG methods, but they are still ~ 3 times slower than ACG.

  A bit surprising is that the adaptive SD method is in fact faster than straight CG, and that ordinary MD is almost as fast as straight CG or adaptive SD.

- But the ACG method clearly beats all the others by a factor of 3 or more.

- Despite this, my recommendation is that in a new minimization problem (atomistic or not) it is best to first implement the full CG method, as it is the most general and reliable one. After that, one can check whether it can be optimized for the particular range of problems, e.g. by a scheme similar to the one above.

  But in every new class of systems is best to first compare full CG and your own method to verify that the assumptions made in your own approach are still valid.

Genetic algorithms

- Genetic algorithms (GA) are a popular method for looking for a global minimum, which have not been used too much in the physical sciences. They are, however, well suited at least for looking for the minimum of a fairly large set of atoms.

- Groups of atoms typically have a fairly large set of energy minima, so the ordinary methods are not well applicable for looking for a global minimum: CG only looks for the closest local minimum, and MD and Monte Carlo (simulated annealing) are fairly easily stuck to one minimum or a local region.

  But the GA method is well suited for looking for global minima, at least for dilute atom systems.

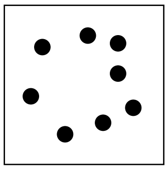
- Genetic algorithms have obtained their inspiration from Darwin's theory of evolution. The idea is to perform natural selection for some group of parameters $\boldsymbol{G}$ which describes well the real system. The group is allowed to breed by mating, after which natural selection is carried out (i.e. the poorest adapted species are killed).

  The parameters $\boldsymbol{G}$ can be considered to correspond to a gene sequence, DNA.

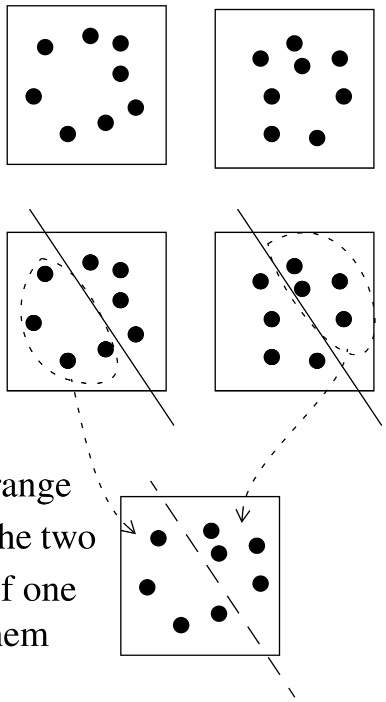
- Here we present the Deaven and Ho approach to genetic algorithms for atoms [Deaven and Ho, Phys. Rev. Lett. 75 (1995) 288]. This version is somewhat simpler than the ordinary GA, but seems like the best implementation to atoms. After this we also present the more conventional GA method.

- Let us state the problem as follows. We have N atoms in free space, and want to find their minimum-energy configuration. The parameter set $\boldsymbol{G}$ is now simple the set of atom coordinates $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ... \boldsymbol{x}_N\}$. Let us assume we also have some interaction model for the atoms.

  We illustrate there the algorithm with 2D figures; in reality it of course usually is in 3D. The difference between 2D and 3D is trivial.



Deaven and Ho genetic algorithm:



0. Start. Create random initial positions for p structures, each with exactly N atoms.

1. Mating and breeding. Select two well-adjusted parents for breeding. This is done by selecting a given parent i with state $\boldsymbol{G}_i$ with the probability

$$P(\boldsymbol{G}) \propto e^{-E(\boldsymbol{G})/T_m}$$

where the mating 'temperature' T_m is selected as the range of energies among the whole population $\{\boldsymbol{G}_i\}$. Split the two parent structures along the same line. Take one half of one parent, and another half of another parent, and join them together.

Here the added complication that a child may have a different number of atoms than the parents comes in. In this case, the lines creating the two parents are moved in opposite directions until a state where the child has equal numbers of parents is found.

2. Mutation. With a probability $\mu \geq 0$ perform a mutation on the child. There are two possible kinds of mutations:

a. Move atoms in a random direction by a random distance a random number of times. The distance is of the order of the bond length, and the number of times ~ 5 - 50.

b. Move an atom *up* along the potential energy function (opposite direction to the force). The aim is to move over potential barriers, i.e. "watersheds".

3. Minimize the energy of the child to the closest local minimum. This is done by CG or MD.

4. Natural selection. If the child has lower energy than any of the parents, allow it to stay alive. Then check that its energy does not match the energy of any parent within an energy range δE. If this is true, include it in the population, and kill the least-well adapted parent (the one with the highest E).

5. If convergence has not been reached, return to stage **1.**

- Some comments:

  - The energy range δE is included to prevent the population from having several identical or very similar structures.

  - The mutation operation can sometimes be completely left out.

  - The mating temperature reduces the probability that poorly adjusted parents get to breed. Hence they are more likely to die without giving rise to any offspring.

    If $T_m >> E_{\mathrm{max}}$ all parents get to breed by about the same probability. If on the other hand $T_m << E_{\mathrm{min}}$ only the best adjusted parents get to breed. (Even this can be seen to have a biological interpretation, although not a very good one: in warm climates it is easier to survive, whereas in harsher, colder climates only the best adjusted individuals can survive and breed...)

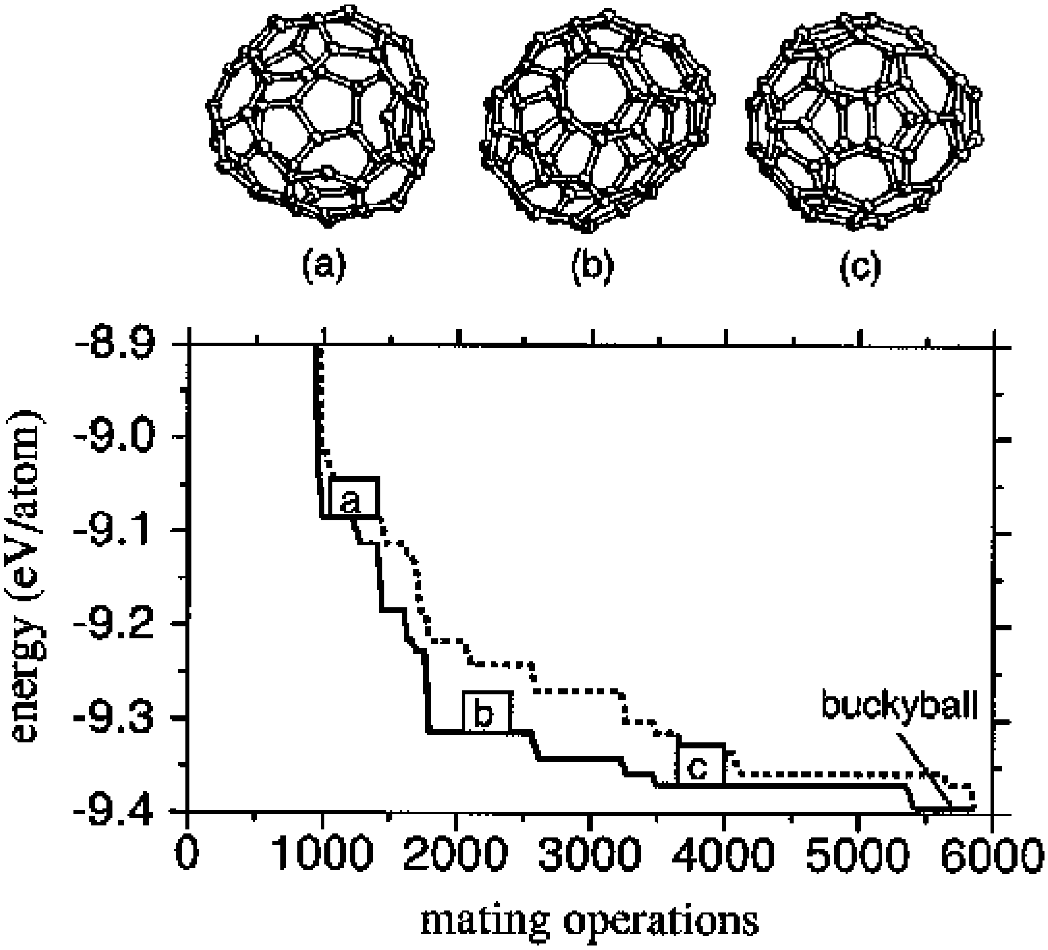
  - The size of the population does not have to be very large. With Deaven and Ho, who used TB, had it usually at 4. Jura Tarus found that somewhat larger numbers work better for the Tersoff C potential.

- Deaven and Ho used their code to find the equilibrium structure for a fullerene C_{60} and other small carbon clusters, starting from random atom coordinates. No other simulation method had at that time been able to produce a fullerene 'from scratch' . Chelikowsky got close with MD [Phys. Rev. Lett. **67** (1991) 2970], but using a to-say-the least suspicious bond-bending-part in his potential. Simulated annealing (a Monte Carlo method) can find the structure of molecules of the order of C_{20} , but not larger than that.

- Deaven and Ho used a Tight Binding force model, which was known to describe fullerenes well.

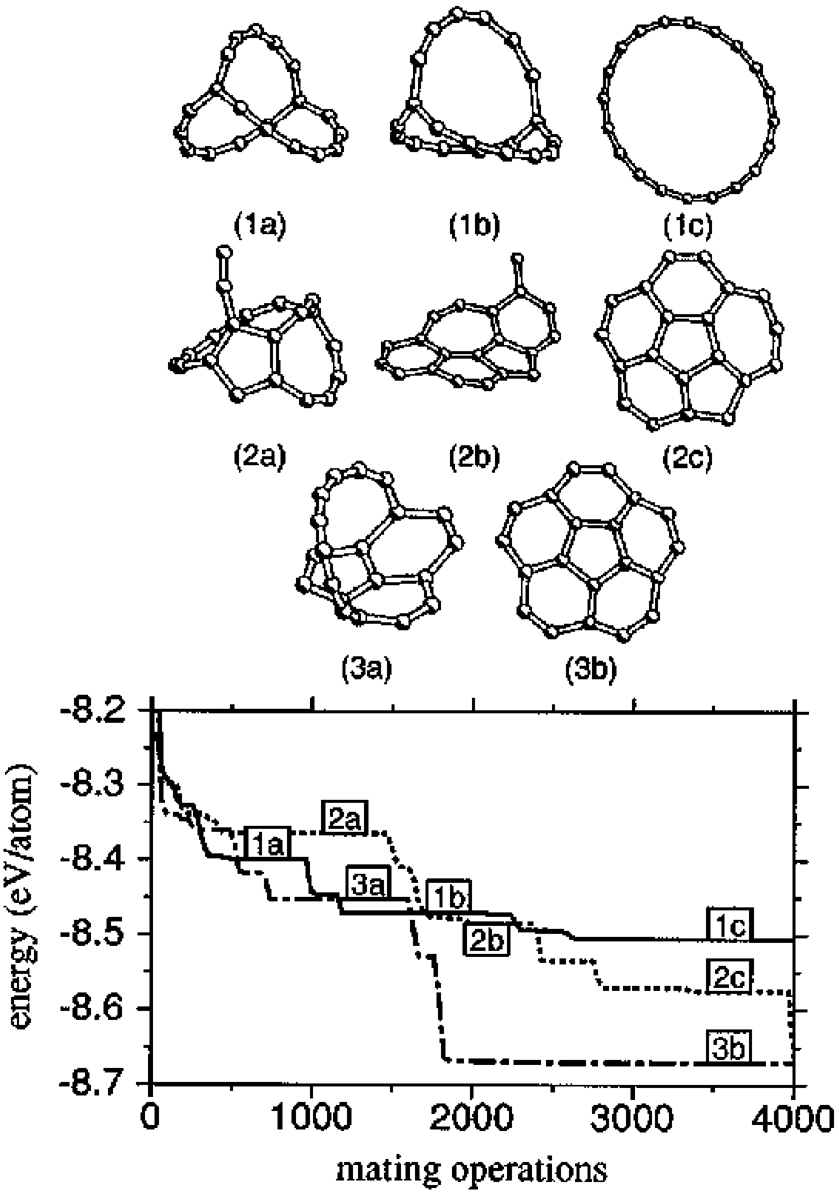
- Parameters: mating temperature $T_m = 0.2$ eV/atom, energy resolution $\delta E = 0.01$ eV, population $p = 4$.

- **Fullerene C_{60} .** The algorithm finds a perfect fullerene after about 6000 mating operations starting from random coordinates, without mutations ($\mu = 0$) :



The upper curve is the maximum energy of the population, the lower the minimun. We see that fairly fast (1000 mating operations) a fairly well-adjusted state (a) which still has defects (a 12-membered atom ring and two 7-membered atom rings). A large fraction of the time, about 5000 mating operations, goes to removing the last defects. In stage (b) there is still left a 7-atom ring, and in state (c) there already is the correct amount of pentagons and hexagons, but two pentagons adjacent to each other.

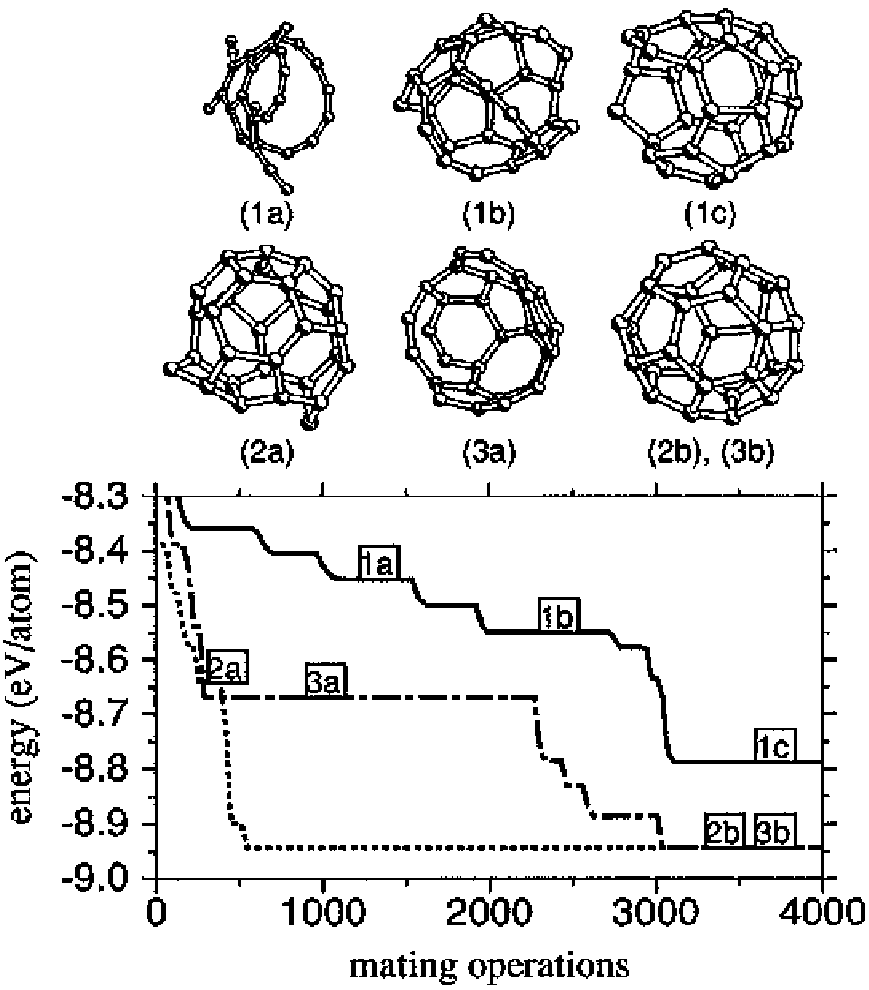
- **Carbon cluster C_{20}.** When the genetic algorithm is run for 20 carbon atoms, the effect of mutations becomes apparent:



States 1 a-c and the solid line describe the results when the code is ran without mutations. The structure is stuck in a round carbon circle. But some $\mu = 0$ - states do find the correct structure.

In states 2 a -c and 3 a-b $\mu = 0.05$. Now the structures find fairly fast the lowest-energy bowl form. State 2 c is already close to the ground state, but the rings on the side have 5 or 7 atoms State 3 b is the correct ground state, with only 6-membered atom rings.

- **Carbon cluster C_{30}**



Most runs end up in the correct state, but some of the $\mu = 0$ states do not in 4000 mating operations found the ground state, but get stuck in state (1c). With $\mu = 0.05$ almost all states end up in the correct cage structure (2b) and (3b). The intermediate configurations (2a) and (3a) show that the correct final state can be reached in several different ways.

- Only mutation, with no mating, does not lead to the correct state.

- But here it is important to realize that the development of the GA minimization process does not necessarily contain any physically meaningful information: the real path to the ground state probably has no relation to the GA path. So only the ground state found by GA may correspond to real life (in case the experimental situation has had time to reach the ground state).

- **The original way of realizing** GA is even closer to nature. In it, the information on the state is coded in a binary “gene sequence” which corresponds to DNA. Let us consider the interaction between two molecules A and B [Xiao and Williams, Chem. Phys. Lett. 215 (1993) 19]. Both molecules can be described with a position and rotation angle, so the information needed ($x, y, z, \alpha, \theta, \phi$) . If we now discretize the possible positions and angles, using e.g. 16 possibilities for each dimension, the state of the molecule can be described with 24 bits of information, for instance

  (4.5 Å,5.0 Å,9.0 Å,120^o, 100^o,60^o)=(1001:1010:1110:0110:0101:0011).

- The breeding operation is defined such that the binary string is exchanged from some point forward (“crossover”). So if we have two parents

  P1 = (1001:1010:1110:0110:0101:0011)
  P2 = (1001:1010:1110:0100:1011:1110)

  and the exchange position is chosen to be 17, we get the children

  C1 = (1001:1010:1110:0110:0 | 011:1110)
  C2 = (1001:1010:1110:0100:1 | 101:0011)

  In this case stage **1.** in the above algorithm simply becomes.

  1. Mating and breeding. Exchange the gene sequence of a parents with another starting from a random position.

- The mutation operation now becomes simply

  2. Mutation. With a given probability μ exchange the state of a bit (0 -> 1 or 1->0) for all bits in all individuals.

- Because a bit corresponds to a position or rotation angle, this directly changes the state of the individual.

- Otherwise the algorithm is essentially as that of Deaven and Ho.

- Xiao used the algorithm to search for the ground state configurations for simple hydrocarbon molecules such as the benzene dimer. He used a population of 100 and 8 bits to code each position or angle.

- But this approach has the problem that during the mating and mutation the state of the molecule can change quite radically, and the properties of the parents are not transferred to the children. Hence Deaven and Ho say that their method is better for optimizing atomic structure.