

SATFO version 4.0
Simulated **A**nnealing **T**oolbox **F**or **O**ptimization

User Manual
Version 3.5

A.H.C. van Kampen
L.M.C. Buydens

University of Nijmegen
Subfaculty of Chemistry
Department of Analytical Chemistry
Toernooiveld 1
NL-6525 ED Nijmegen, the Netherlands
(email: avkampen@sci.kun.nl)

12-2-96

Contents

1	Copyright	4
2	Introduction	5
2.1	References	5
3	Simulated Annealing	5
3.1	The principle	5
3.2	Statistical mechanics and simulated annealing	6
4	General description of SATFO	8
4.1	Algorithms	8
4.2	Extensions	8
4.3	SATFO files	9
4.4	Using SATFO	9
4.5	Compilation	11
4.6	Executing	11
4.7	Setting file	11
4.7.1	Explanation of setting file	16
4.8	Output	19
4.9	Display of the data	20
5	Measuring	20
5.1	Equilibrate the system	20
5.2	Uncorrelated measurements	21
5.2.1	Determining the equilibrium state	22
5.2.2	Determining the acceptance ratio	22
5.2.3	Determining the mean function value and standard deviation	22
5.2.4	Calculating the specific heat	22
5.3	An example	22
6	Details of features of SATFO	23
6.1	Acceptance ratio	23
6.2	Specific Heat	23
6.3	Generating algorithms	23
6.4	Heating algorithms	24
6.5	Cooling algorithms	24
6.6	Acceptance algorithms	24
6.7	Ending Markov chain	25
6.8	Ending simulated annealing	25
6.9	Boltzmann annealing	25
6.10	Fast Simulated Annealing	25
6.11	Generalized Simulated Annealing	25
7	Tutorial and Example	26

1 Copyright

This documentation is part of Simulated Annealing Toolbox for Optimization (SATFO). SATFO is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SATFO is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with SATFO. If not, see <http://www.gnu.org/licenses/>.

2 Introduction

SATFO (**S**imulated **A**nneling **T**oolbox **F**or **O**ptimization) is a simulated annealing (SA) toolbox written in standard ANSI C. It can be used to setup a simulated annealing algorithm application. Different forms of SA were implemented according to literature resources. Most functions that are contained in SATFO were tested (except the Dynamic geometric cooling schedule), but no guarantees are given. SATFO is used (and tested) by several people, and consequently the possibility for serious bugs is diminished. SATFO will be improvements and extended in the future.

2.1 References

Readings on SA not cited in the text: [1, 2, 3, 4]. An extensive annotated bibliography is available from [5]. A mailing list is available for SA. To subscribe send an email to *anneal-request@chaco.com* with the message **subscribe**.

3 Simulated Annealing

Kirkpatrick et al. [2] pointed out the correspondence between statistical mechanics and problems of combinatorial optimization. Physical systems may be coaxed into a minimum energy conformation (e.g., a crystal) by a slow annealing process. The reduction of the temperature confines the system to a smaller and smaller region of the phase space, but is carried out slow enough to allow the system to pass out of local energy minima. In this way, the system arrives at the global minimum configuration. This process cannot only be used for combinatorial optimization, but can also be applied to numerical optimization problems.

3.1 The principle

The principle of simulated annealing is very simple. Consider the problem of finding the minimum of the function $f(\bar{P})$ (which may represent a combinatorial or numerical optimization problem). The problem is solved by finding the values of the parameters P_i ($i = 1...n$, where n is denotes the problem dimensionality) that minimize the cost function $f(\bar{P})$.

Starting from a set of parameters \bar{P} that were initialized at random, a sequence of new sets is generated by making modifications of the previous set. These modifications are made by a so-called **parameter generation function** G , i.e. $\bar{P}_{t+1} = G(\bar{P}_t)$. The sequence of sets of parameters is called a **Markov chain**. If a new set of parameters is assembled this new set is accepted if it decreases the error, i.e. $f(\bar{P}_t) < f(\bar{P}_{t+1})$. However, if the new set of parameters increases the error then it is only accepted with a probability P_{accept} . This probability is calculated according to a pre-defined distribution (e.g., Boltzmann, $P_{accept} = e^{\frac{\Delta f}{T}}$), and depends on the difference in the function values at iteration (Markov chain number) t and $t + 1$, and may depend on the control parameter c , which is called the **temperature**.

This temperature is initially set at some high value, causing high values for the probability P_{accept} . Consequently, almost every set of new parameters (improvements or not) will be accepted, i.e., this stage of the optimization process corresponds to a random search.

After generating a sequence of sets of parameters at constant temperature, the temperature will be decreased according to a pre-define **cooling schedule**. Subsequently, a new Markov chain is generated. Since the temperature is decreased, the acceptance probability P_{accept} will also decrease, and therefore, the chance for accepting a parameter set that increases the error diminishes.

By decreasing the temperature after each Markov chain, the search becomes biased to local search. If the temperature is low enough, no further changes of the parameters will be accepted unless it is an improvement.

3.2 Statistical mechanics and simulated annealing

This section comprises a brief summary of [2]

Statistical mechanics is the central discipline of condensed matter physics, a body of methods for analyzing aggregate properties of large number of atoms to be found in samples of liquid or solid matter. The most probable behavior of such a system in *thermal equilibrium* at a given temperature can be characterized by the average and small fluctuations about the average behavior of the system, when the average is taken over the ensemble of identical systems introduced by Gibbs. In this ensemble, each configuration, defined by the set of atomic positions $\{r_i\}$, of the system is weighted by its Boltzmann probability factor, $e^{\frac{-E(\{r_i\})}{k_B T}}$, where $E(\{r_i\})$ is the energy of the configuration, k_B is Boltzmann's constant, and T is the temperature.

For finding the ground state of the system, a low temperature ($T = 0$) is not a sufficient condition. Physical experiments that determine the low-temperature state of a material (e.g., growing a single crystal from a melt), are done by careful annealing, first melting the instance, then lowering the temperature slowly, and *spending a long time at temperatures in the vicinity of the freezing point* (which is not necessarily at $T = 0$). If this is not done, the resulting substance will be meta-stable with only locally optimal structures.

Metropolis et al. [6] introduced a simple algorithm that can be used to provide an efficient simulation of a collection of atoms in equilibrium at a given temperature. In each step of this algorithm, an atom is given a small random displacement and the resulting change, ΔE , in the energy of the system is calculated. If $\Delta E \leq 0$, the displacement is accepted, and the configuration with the displaced atom is used as the starting point of the next step. The case $\Delta E > 0$ is treated probabilistically: the probability that the new configuration is accepted is $P(\Delta E) = e^{\frac{-\Delta E}{k_B T}}$. By repeating the basic step many times, one simulates the thermal motion of atoms in thermal contact with a heat bath at temperature T . The choice of $P(\Delta E)$ has the consequence that

the system evolves into a Boltzmann distribution.

Using a *cost function* in place of the energy, and defining configurations by a set of parameters $\{x_i\}$, the Metropolis procedure can be used to generate a population of configurations of a given optimization problem at some temperature. The temperature has of course no physical meaning in optimization, it is merely used as an (important) control parameter. The *simulated annealing* process consist of first 'melting' the system being optimized at a high temperature, then lowering the temperature by slow stages until the system 'freezes'. *At each temperature the simulation must proceed long enough for the system to reach a steady state.* The sequence of temperatures and the number of rearrangements of the $\{x_i\}$ attempted to reach equilibrium at each temperature can be considered an annealing schedule.

Statistical mechanics can be used for extracting properties of a macroscopic system from microscopic averages. Ensemble averages can be obtained from a single generating function, the *partition function* Z (a sum over all possible configurations),

$$Z = \sum_{\{x_i\}} e^{\frac{-E}{k_B T}}$$

The logarithm of Z , called the *free energy*, $F(T)$, contains information about the *average energy* $\langle E(T) \rangle$, and the *entropy*, $S(T)$, which is the logarithm of the number of configurations contributing to the ensemble at T :

$$-k_B T \ln Z = F(T) = \langle E(T) \rangle - TS$$

Boltzmann-weighted ensemble averages are easily expressed in terms of derivatives of F . Thus the average energy is given by:

$$\langle E(T) \rangle = \frac{-d \ln Z}{d(1/(k_B T))}$$

The rate of change of the energy with respect to the control parameter T , is related to the size of typical variations in the energy by:

$$C(T) = \frac{d \langle E(T) \rangle}{dT}$$

$$C(T) = \frac{[\langle E(T)^2 \rangle - \langle E(T) \rangle^2]}{k_B T^2}$$

In statistical mechanics $C(T)$ is called the specific heat. A large $C(T)$ signals a change in the state of order of a system, and *can be used in optimization context to indicate that freezing has begun, and hence that very slow cooling is required.*

In optimization $E(T)$ represents the cost function value for a specific set $\{x_i\}$ at temperature T . The analogy between cooling a fluid and optimization may fail in one important respect. In ideal fluids all the atoms are alike and the ground state is a regular crystal. A typical optimization problem will contain many distinct, noninter-changeable elements, so a regular solution is unlikely.

The use of these concepts are shown in the example section of this manual.

4 General description of SATFO

4.1 Algorithms

Most algorithms (concerning simulated annealing) contained in SATFO were programmed after literature resources. Furthermore, SATFO makes use of several functions and macro's that were originally programmed by C.B. Lucasius for the genetic algorithm toolbox GATES [7, 8]. These functions were modified to suite SATFO and are contained in the files `SArandom.c`, `SArandom.h`, `SAtypedef.h`. The first two files contain functions and a macro for the generation of random numbers, which make use of a Fibonacci VLP (Very Long Period) pseudo-random number generator [9, 10]. The primitive of this generator generates random 24-bit mantissas - fractional parts of a single-precision real number.

The file `SAtypedef.h` contains several type definitions used by SATFO.

The available macro in `random.h` is used for generating random real numbers:

- **RANDOM_BOUND_DOUBLE** - Generates a random value of double precision format. This macro takes two arguments: a upper bound (*Bound*), and a (global) variable which is called the random fraction (*RandomFraction*). This variable is changed on every call of the macro, and ensures that a sequence of different random numbers is generated.

Example: $X = \text{RANDOM_BOUND_DOUBLE}(100.0, \&\text{RandomFraction})$ generates a double precision real value between 0 (inclusive) and 100 (exclusive).

SATFO also makes use of the file `calloc.h` that contains macro's for memory allocation that are based on the `alloc` C-function. These macros were written by A.D. Dane:

- **CALLOCxD** - Allocation of an x-dimensional array, where $x = 1, 2, 3$ or 4 . The macro takes $x + 1$ arguments: P , which is the address of the x-dimensional array, and x-times N , which is the number of desired array elements in each dimension. For example: $\text{CALLOC1D}(P, N)$ allocates memory for a 1D-array P , which has N elements. The array elements are of type identical to P ; $\text{CALLOC2D}(Q, N1, N2)$ allocates memory for a 2D-array Q of size $N1 \times N2$.
- **FREExD** - Counterpart of CALLOCxD . This macro frees the memory allocated with CALLOCxD . For example: $\text{FREE1D}(P)$ and $\text{FREE2D}(Q)$ frees the above allocated arrays.

4.2 Extensions

Although many common instances of the SA algorithm can be assembled with this toolbox it is not yet complete. Possible extensions are:

1. **Polynomial time cooling schedule** [11, 12, 13]

2. **Heating schedule** $c_0 = \frac{\overline{\Delta C}^+}{\ln \chi_0^{-1}}$ where c_0 is the initial temperature, $\overline{\Delta C}^+$ is the average increase in cost and χ_0 is the initial acceptance ratio.

An extension that will not be added is *very fast simulated reannealing* developed by L. Ingber [14, 15, 16, 17]. This algorithm is available by ftp from `ftp.caltech.edu` in `/pub/ingber` as `ASA`.

4.3 SATFO files

The following files comprise the SA toolbox (SATFO):

- **SATFOv31.c**: main file, containing SA algorithm.
- **SATFOv31.h**: include file for SA.c. This file contains a definition for the machine precision of the computer. This value should be adjusted in order to be compatible with you computer.
- **SATFOv31.set**: setting file (see below).
- **SA_User.c** (or any other user file): file containing the cost function and the user defined parameter generating function.
- **SArandom.c**: random number generator.
- **SArandom.h**: random number generator macro and type definitions.
- **SAtypedef.h**: type definitions used within the other files
- **calloc.h**: memory allocation macros.

4.4 Using SATFO

Figure 1: Basic framework of SATFO.

Figure 1 denotes the basic framework of SATFO. It is used by including your own *cost function* in a separate file (user file). This function must return a positive value, which represents a measure for the error. The simulated annealing algorithm will minimize this error. In addition it is possible to include your own *parameter generating function*, but this is not strictly necessary because SATFO contains several standard functions (see below). In your own written user file you call the main simulated annealing function $SA(argc, argv)$, where $argc$ and $argv$ are arguments that are passed to $SA()$ when it is executed (see below). This function is contained in the file `SATFOV40.c`.

An example program for finding the minimum in a hyperparabole is given below. The search strategy moves through the search space by modifying the set of problem parameters (i.e., the array *Parameters*) according to the user defined parameter generator function. This last function adjust the current set of parameters by adding a real number (between -1 and +1) to the current set of parameters.

```
void main(int argc, char *argv[])
{
    SA(argc, argv);
}

double EvaluateCostFunction(int      NumberOfParameters,
                           double   *Parameters)
{
    int i;
    double y;

    y=0.0;

    for(i=0;i<NumberOfParameters;i++)
        y=y+SQR(Parameters[i]);

    return(y);
}

GenerateNewParametersUser(double *ParametersNew,
                          double *Parameters,
                          double *UpperBounds,
                          double *LowerBounds,
                          int      NumberOfParameters,
                          double   ControlParameter)
{
    /* uniform(-1,1) is user defined random generator */
    for(i=0;i<NumberOfParameters;i++)
        ParametersNew[i]=Uniform(-1,+1)+Parameters[i];
}
```

Remark: the functions `EvaluateCostFunction` and `GenerateNewParametersUser()` are restricted to these names and parameter arguments.

4.5 Compilation

If any modifications are made to the main file `SA.c` then it has to be recompiled with:

```
gcc -c SA.c
```

```
gcc -c SArandom.c (only if SArandom.o does not exist)
```

Compiling your own module:

```
gcc -o simanneal <own filename> SA.o SArandom.o -lm
```

This results in the executable `simanneal`

4.6 Executing

```
simanneal <random seed> <input setting file> <output file>
```

or

```
simanneal <random seed> <input setting file> <output directory> <output file>
```

Example 1: `simanneal 34323 SA.set SA.out`

Example 2: `simanneal 34323 SA.set program/data/ SA.out`

4.7 Setting file

All parameters that control the simulated annealing algorithm can be set via an external setting file. This is a easy way to adjust the SA parameters without having to recompile the program after a change in its configuration.

Although the file `SA.set` has a predefined format, formatting restrictions are minimal. These are, that the entries should each be preceded by a colon character (`:`) and should be specified in a predefined order; thereby, compound entries – i.e. entries which are designated to take the form of a list of entries – are also considered as entries.

Next, we discuss an exemplary transcript of a configuration file for a 11-dimensional problem. Thereby, for the sake of clarity, the successive entries are indicated by serial indices – 00 to 43 – before the colon, thus not violating mentioned formatting restrictions.

```
#####  
# SATFO setting file #  
#####
```

```
#####  
# WARNING -- UNPREDICTABLE BEHAVIOR IS LIKELY TO OCCUR WHEN #  
# - You edit this file with a non-'flat-ASCII' editor #  
# - You edit other text parts besides the entries (preceded by a colon) #  
# - You specify inconsistent entries #  
#####
```

PROBLEM PARAMETERS

NUMBER OF (= \$)

00: 11

FLOORS		
[\$ reals]		01: -10.0 -10.0
		-10.0 -10.0
		-10.0 -10.0
		-10.0 -10.0
		-10.0 -10.0 -10.0
CEILINGS		
[\$ reals]		02: 10.0 10.0
		10.0 10.0
		10.0 10.0
		10.0 10.0
		10.0 10.0 10.0
CENTERS		
[\$ reals]		03: 0.0 0.0 0.0 0.0 0.0
		0.0 0.0 0.0 0.0 0.0 0.0
GENERATING FUNCTION		
GENERATING MODE		
[1 = Gauss		
2 = Cauchy		
3 = Uniform		
4 = User]		05: 3
STEP SIZE		
[real, >0.0		
ignored for mode 4]		06: 0.01
HEATING		
HEATING MODE		
[1 = Geometric		
2 = Constant]		07: 2
STARTING TEMPERATURE		
[real, >0.0		
ignored for heating mode 2]		08: 10.0
HEATING FACTOR GEOMETRIC		
[real, >1.0		
ignored for heating mode 2]		09: 1.0
INITIAL ACCEPTANCE RATIO		
[real, >=0.0, <=1.0		
ignored for heating mode 2]		10: 0.90
PROPOSED TRANSITIONS		
[real, >=1		
ignored for heating mode 2]		11: 150

CONSTANT TEMPERATURE
 [real, >0.0
 ignored for heating mode 1] 12: 5.3

COOLING

COOLING MODE
 [1 = Geometric
 2 = Dynamic geometric
 3 = Linear
 4 = Geometric2
 5 = Logarithmic
 6 = Exponential] 13: 1

COOLING FACTOR GEOMETRIC
 [real, >0.0, <=1.0
 ignored for cooling modes 2,3,4,5,6] 14: 0.90

COOLING FACTOR EXPONENTIAL
 [real, >1.0,
 ignored for cooling modes 1,2,3,4,5] 15: 0.9

DECREMENT STEP
 [real, >0.0,
 ignored for cooling modes 1,2,4,5,6] 16: 0.0001

DISTANCE PARAMETER
 [real, >0.0,
 ignored for cooling modes 1,3,4,5,6] 17: 1.0

SETTING EQUILIBRIUM

LENGTH MARKOV CHAIN 18: 5000

EXIT MARKOV CHAIN

EXIT MODE
 [1 = Minimum transitions
 2 = Maximum transitions
 3 = Constant function values
 4 = Maximum rejections
 Can be used in compound mode] 19: 12

MINIMUM NUMBER TRANSITIONS
 [integer, >1,
 ignored for exit modes 2,3,4] 20: 5000

MAXIMUM NUMBER TRANSITIONS
 [integer, >1,
 ignored for exit modes 1,3,4] 21: 5000

NUMBER OF CONSTANT FUNCTION VALUES
 [integer, >1
 ignored for exit modes 2,3,4] 22: 35

MAXIMUM NUMBER REJECTIONS	
[integer, >1	
ignored for exit modes 1,2,3]	23: 25
EXIT SIMULATED ANNEALING	
EXIT MODE	
[1 = Minimum control parameter	
2 = Minimum acceptance	
3 = Constant function values	
4 = Machine precision	
5 = Minimum value cost function	
Can be used in compound mode]	24: 12
MINIMUM CONTROL PARAMETER	
[real, >0.0,	
ignored for exit modes 2,34,5	25: 0.00001
MINIMUM ACCEPTANCE	
[real, >0.0,	
ignored for exit modes 1,3,4,5]	26: 0.001
NUMBER OF CONSTANT FUNCTION VALUES	
[integer, >1	
ignored for exit modes 1,2,3,5]	27 [SET AT 16]
MINIMUM VALUE COST FUNCTION	
[real, >0.0,	
ignored for exit modes 1,2,3,4]	28: 0.0001
ACCEPTANCE CRITERION	
ACCEPT MODE	
[1 = Metropolis criterion	
2 = Sigmoid criterion	
3 = GSA]	29: 1
Optimal Value Error Function	
[real, >= 0.0	
ignored for accept modes 1 and 2]	30: 0.0
Factor GSA	
[real, >0.0	
ignored for accept modes 1 and 2]	31: 10.0
SKIP STEPS	
EQUILIBRIUM	
[integer, >1]	32: 25
FUNCTION VALUES	
[integer, >1]	33: 25

SPECIFIC HEAT	
[integer, >n*Function Values]	34: 500
RUNTIME OUTPUT	
LOG OUTPUT	
[1 = To output file	
2 = To screen	
3 = No log output]	35: 2
INTERMEDIATE	
[1 = Print intermediate output	
0 = Don't print intermediate output]	36: 0
CONTROL PARAMETER OUTPUT	
[1 = To output file	
2 = To screen	
3 = No log output]	37: 1
ACCEPTANCE RATIO OUTPUT	
[1 = To output file	
2 = To screen	
3 = No log output]	38: 1
COST FUNCTION VALUE OUTPUT	
[1 = To output file	
2 = To screen	
3 = No log output]	39: 1
PARAMETER OUTPUT	
[1 = To screen	
2 = No output]	40: 1
MEAN FUNCTION VALUE OUTPUT	
[1 = To output file	
2 = To screen	
3 = No log output]	41: 1
SPECIFIC HEAT VALUE OUTPUT	
[1 = To output file	
2 = To screen	
3 = No log output]	42: 1
WORK VALUE OUTPUT	
[1 = To output file	
2 = To screen	
3 = No log output]	43: 1

4.7.1 Explanation of setting file

1. PROBLEM PARAMETERS:

This section defines the number of parameters to be optimized in your application. For each parameter a lower and upper bound must be defined. The parameters will be initialized within these bounds and the parameter generating functions will only make changes within the bounds (unless you don't check on the bounds in your own defined generating function).

- (a) **Number of parameters:** number of parameters in your cost function.
- (b) **Lower bounds:** for each parameter a lower bound must be defined.
- (c) **Upper bounds:** for each parameter a upper bound must be defined.
- (d) **Centers:** *not used*. This will be removed in a future version.

2. PARAMETER GENERATING FUNCTIONS:

Four possibilities are available to make steps through the search space. Three standard functions are defined. In addition the user can include his own function.

If one of the standard functions are used, these parameter generator functions adjust the array of problem parameters \bar{p} at iteration t according to

$$\bar{p}_{t+1} = \bar{p}_t + \text{step size} \times \bar{D}$$

\bar{D} is a unit vector in the search space spanned by the parameters \bar{p} , and indicates the direction of the move. The elements of this vector are $\frac{D(i)}{\|D\|}$ ($i = 1 \dots N$), where $D(i)$ represents a value generated from one of the three standard distributions.

- (a) **Gauss:** $D(i)$ is generated from the Gaussian distribution $N(0, 1)$.
- (b) **Cauchy:** $D(i)$ is generated from the Cauchy distribution $C(0, 1)$.
- (c) **Uniform:** $D(i)$ is generated from the uniform distribution $U(-1, +1)$.
- (d) **User:** Parameters are adjusted according to a user defined function.
- (e) **Step size:** this value sets the magnitude of the step taken in the direction of the generated unit vector \bar{D} . This value can also be used for the the user defined function.

3. SETTING INITIAL CONTROL PARAMETER:

Before starting the simulated annealing process, an initial value for the control parameter c must be set. SATFO offers two possibilities to set this value. First of all it is possible for automatically setting this value according to some predefined criterion. Secondly it is possible for the user to set the initial value.

- (a) **Heating mode:** this parameter defines which method is used for setting the value of the initial control parameter. If the **geometric** mode is chosen, then the control parameter is automatically set to the an optimal value, i.e., c is increased from an initial value to its final value according to a geometric schedule, i.e., $c = \beta \times c$

The magnitude of the increment is set by the parameter **heating factor geometric** (β). The control parameter c is increased until a predefine acceptance ratio is reached (typically within the range of 0.7 to 0.9). This acceptance ratio is set by the parameter **initial acceptance ratio**. The number of function evaluations from which this acceptance ratio is calculated is set by the parameter **length Markov chain for heating**. The heating process starts at the value set by the parameter **starting temperature**.

- (b) **Starting temperature:** initial value for control parameter c for heating mode **geometric**.
- (c) **Heating factor geometric:** increment factor β for the control parameter in the geometric schedule.
- (d) **Initial acceptance ratio:** heating is terminated if this predefined value is reached.
- (e) **Length Markov chain for heating:** number of monte Carlo trials from which the acceptance ratio is calculated during heating. A longer chain gives a more reliable estimate of the acceptance ratio.
- (f) **User defined initial temperature:** initial value for control temperature set by user. No heating is used

4. DECREASING CONTROL PARAMETER:

A cooling schedule for decreasing the control parameter c must be chosen in order to 'freeze' the system during the annealing process. Several standard schedules are available, and are discussed and below.

- (a) **Cooling schedule:** select type of cooling schedule
- (b) **Cooling factor geometric:** control parameter
- (c) **Cooling factor exponential:** control parameter
- (d) **Decrement step:** control parameter
- (e) **Distance parameter:** control parameter

5. SETTING EQUILIBRIUM

The use of the first Markov chain is to equilibrate the system. This is explained in more detail below. This length should be zero for a normal optimization process. Only for 'measurements' on the system, this value should be set.

- (a) **Length Markov Chain:** Length of the first Markov chain.

6. EXIT MARKOV CHAIN

Four options are available to exit a Markov chain, whereafter the control parameter is decreased (or simulated annealing is terminated).

- (a) **Exit mode:** exit criterion. All modes can be combined. For example if the minimum number of desired transitions is 100 but no more than 5000 attempts must be made to make a move, the exit mode can be set to 12.
- (b) **Minimum number of transitions:** Minimum length Markov chain.
- (c) **Maximum number of transitions:** Maximum length Markov chain.
- (d) **Number of constant function values:** if the error function value does not change for this number of iterations, the current Markov chain is terminated.
- (e) **Maximum number of rejections:** if a proposed moves (new parameter sets) are rejected this number of times, the Markov chain is terminated.

7. EXIT SIMULATED ANNEALING

Five options are available to terminate the simulated annealing process.

- (a) **Exit mode:** termination criterion. All modes can be combined. For example if the exit mode is set to 12 then simulated annealing is terminated if either the control parameter or the acceptance ratio drop below the specified values.
- (b) **Minimum control parameter:** simulated annealing is terminated if the temperature drops below this value. Remark: this criterion must not be selected in case of GSA (Generalized simulated annealing).
- (c) **Minimum acceptance:** simulated annealing is terminated if the acceptance ratio drops below this value.
- (d) **Constant function values:** If the error values do not change for this number of iterations, then simulated annealing is terminated.
- (e) **Machine precision:** if the difference between two succeeding error values drops below the machine precision, the simulated annealing is terminated. Remark: the machine precision is defined in the header file `SATFOV31.h`.
- (f) **Minimum value cost function:** if the error value drops below this threshold, then simulated annealing is terminated.

8. ACCEPTANCE CRITERION

A move that result in an increased error value is only accepted with a probability that is obtained from a pre-specified criterion. Three options are available.

- (a) **Accept mode:** select the **Metropolis**, **Sigmoide** or **GSA** (Generalized Simulated Annealing) criterion. This are explained in more detail below.
- (b) **Optimal value error function:** expected minimum error value. This is used in combination with GSA.

- (c) **Factor GSA:** control parameter for influencing the acceptance probability for inferior moves.

9. SKIP STEPS

The use of this parameters is explained in more detail in the chapter about 'measuring'. The values for the next three parameters should be 1 for a normal optimization procedure. For a 'measurement' on the system, alternative values must be set.

- (a) **Equilibrium**
- (b) **Function values**
- (c) **Specific heat**

10. RUNTIME OUTPUT

Several file can be generated during the simulated annealing process, or the output may simply be written to the screen. The files are explained in more detail below.

- (a) **Log output:** if selected, run time information is printed to the screen or the file that was given as command line argument.
- (b) **Intermediate**
- (c) **Control parameter output**
- (d) **Acceptance ratio output**
- (e) **Cost function value output**
- (f) **Parameter output**
- (g) **Mean function value output**
- (h) **Specific heat value output**
- (i) **Work value output**
- (j) **Evaluate output:** not implemented yet.

4.8 Output

Output generated by SATFO can be written to file or screen, or can be suspended. The output that can be generated is:

1. **log file:** this file contains information about the progress of the run. The file name is the name of the output file with which you executed the program. This file can grow very large.
2. **Accept.out:** contains acceptance ratios [11, 12, 13] of subsequent Markov chains [18].
3. **Temp.out:** contains values of the control parameter of subsequent Markov chains.

4. **Func.out**: contains cost function values for the best function value ever generated for each Markov chain. Chain.
5. **Mean.out**: contains mean function values and the standard deviation for each Markov Chain (see section 'measuring').
6. **Heat.out**: contains the mean specific heat and its standard deviation for each Markov Chain (see section 'measuring').
7. **Eval.out**: This file contains information about the number of function evaluations. It contains four entries: the first is the markov chain number, the second is the length of the Markov chain, the third is the number of transitions, and the last is the total number of function evaluations.
8. **Work.out**: contains two columns with information about the equilibrium state of the system. The first column represents the first Markov chain (non-equilibrium) and the second column represents the second Markov chain (equilibrium) (see section 'measuring').

4.9 Display of the data

An matlab function (`Display.m`) is available which plots the obtained data. This program does not visualize the standard deviations for the mean function value or specific heat. For visualizing the standard deviations use the matlab function `Display2.m`. These two functions are available in the `tutorial_data` directory.

5 Measuring

During the simulated annealing process several quantities, like the acceptance ratio, average function value for each Markov chain and the specific heat can be calculated. In other words, 'measurements' are made to the system. The results of these measurements may help to improve the performance of the simulated annealing process, or just to verify if the optimization strategy is operating as expected.

5.1 Equilibrate the system

In real physical or chemistry experiments, measurements are often made after the system to be measured is in equilibrium. For example, before measuring the pH of a solution, this solution is stirred for some time to be certain that the solution is in equilibrium, i.e., homogeneous. A measure the viscosity of a solution at a specific temperature, the solution is, for example, placed in a heat bath until a thermal equilibrium is reached.

In order to make measurements to the optimization system, an equilibrium should be reached. This means that at each temperature T the uphill-moves (increase in

error value) and the down-hill moves (decrease in error value) cancel, i.e., $\sum f_j = \sum [f_{increase} - f_{decrease}] = 0$. Here $f_{increase}$ represents a transition to a state of an increased error value, and $f_{decrease}$ a transition to a state of an decreased error, and the sum is taken over all configurations (i.e., parameter sets $\{x_i\}$) in a single Markov chain. After each decrease of the temperature, there will be a bias for down-hill moves until equilibrium is reached. Only then, measurements on the system are meaningful. To obtain this equilibrium, SATFO makes use of the first Markov chain (see Figure 1). This must be long enough to reach equilibrium, and the function values generated in this chain are not used. The second Markov chain is used to make the actual 'measurements' by sampling from the equilibrium distribution at temperature T . To check the equilibrium, SATFO writes $\sum f_j$ for Markov chain 1 and 2 at each temperature to the file `Work.out` (this sum actually is the work performed by the system). The work for Markov chain 1 should be larger than for Markov chain 2. Ideally, the work in Markov chain 2 should be zero. However, this probably requires very long Markov chains in the first stage.

5.2 Uncorrelated measurements

A second requirement to make measurements on a physical system is that the measurements must be uncorrelated.

In simulated annealing subsequent states (i.e., parameter sets $\{x_i\}$) may be correlated because a subsequent state is generated from the current state (unless, for example, a random parameter generation function is used). To prevent a bias in the measurements, SATFO provides three control parameters to obtain uncorrelated (less correlated) measurements. These control parameters are the three **skip steps** parameters. Their use is shown in Figure 2.

Figure 2: Measuring uncorrelated samples

5.2.1 Determining the equilibrium state

To obtain an indication of the work (i.e., the equilibrium state of the system in the first and second Markov chain), the parameter *equilibrium* is used. In the sum $\sum[f_{increase} - f_{decrease}]$ not every subsequent function value is used. After each used function evaluation the specified number of evaluations by the parameter *equilibrium* is skipped.

5.2.2 Determining the acceptance ratio

As shown in Figure 2, in the first Markov chain no other quantities are calculated. In the second Markov chain, however, the acceptance ratio, mean function value of each Markov chain and the specific heat are calculated. The acceptance ratio is calculated from the same function values that are used to determine the equilibrium state of the system, and it thus influences by the parameter *equilibrium*.

5.2.3 Determining the mean function value and standard deviation

To obtain a measure for the mean function value, it is possible to sum the function values of all subsequent states. However, to obtain a unbiased measure and a value for the standard deviation, the parameter *function values* is used. This parameter determines how many function values are used to calculate the mean and standard deviation. These two values are written to the file `Mean.out`. Note that calculation of the mean and standard deviation is only possible if the parameter *function values* is larger than 1. The parameters *equilibrium* and *function values* may have equal values, and consequently, the same function values are used for determining whether the system is in equilibrium and for calculating the mean and standard deviation.

5.2.4 Calculating the specific heat

For the calculation of the specific heat we need the average function value. This quantity is calculated by using the function values according to the parameter *function values*, which is explained above. The parameter *specific heat* determines how many function values are calculate the mean in order to calculate a specific heat. This parameter must be an integral number times the parameter *function values*. From the obtained mean value a specific heat can be calculated. Subsequently, the next set of function values is used to calculate a second specific heat. Finally, the average specific heat and its standard deviation is calculated from the obtained set of individual specific heats. The values for *function values* and *specific heat* determine how many specific heats are determined during one Markov chain. The result is written to the file `Heat.out`.

5.3 An example

In Figure 2 the first Markov chain has a length of 36 steps. The parameter *equilibrium* was set to 4, and subsequently 9 values are used to determine the equilibrium

state of the system. Thereafter, the second Markov chain starts, which has a length of 124 steps. In this chain 30 values are used to determine the equilibrium state and the acceptance ratio. The parameter *function values* was set to 8, and consequently 15 function values are used to calculate the mean and standard deviation at the end of the second Markov chain. The parameter *specific heat* was set to 4 times *function values*, and consequently 4 values are used to calculate a specific heat. At the end of the second Markov chain, four specific heats are calculated from which an average specific heat and standard deviation is obtain.

The values presented here for the skip parameters do not represent values that should be used in real simulations. There, their values should be much larger.

6 Details of features of SATFO

6.1 Acceptance ratio

The acceptance ratio χ is defined as $\chi = \frac{\text{accepted transitions}}{\text{proposed transitions}}$. $0 \leq \chi \leq 1$.

The number of proposed transitions is equal to the length of the Markov chain (which may be different for each temperate). Furthermore, if the **SKIP STEPS** parameters are not equal to 1, the number of accepted and proposed transitions must be calculated as explained in the section 'measuring'.

6.2 Specific Heat

The specific heat is calculated as: $\frac{\langle f^2 \rangle - \langle f \rangle^2}{c^2}$ where $\langle .. \rangle$ denotes the expected value [19].

6.3 Generating algorithms

There are three methods implemented to generate new parameters. The fourth option is to use a user defined function. The three standard methods all use the same procedure. First, for each parameter, a random number is drawn from a specific distribution. This results in a n dimensional vector (n represents the number of parameters). This vector is normalized to an unit vector. Then new parameters are generated by:

$$\overline{\mathbf{p}}_{t+1} = \overline{\mathbf{p}}_t + \text{step size} \times \overline{\mathbf{Direction}}$$

The following distributions are implemented:

1. **Gauss** [20, 17]: *Gauss*(0, 1) : $x \sim e^{x^2}$
2. **Cauchy** [20, 17]: *Cauchy*(0, 1) : $x \sim \frac{1}{1+x^2}$
3. **Uniform** [21]: *Uniform*(-1, +1)
4. **User defined**

6.4 Heating algorithms

The following heating algorithms are implemented in SATFO:

1. **Geometric** [11, 12, 13, 22]: $c_{k+1} = \alpha * c_k$ with $\alpha > 1.0$
A typical value for $\alpha = 2.0$, the iteration is terminated if a minimum acceptance ratio is reached.
2. **Constant**: the control parameter is simply set to an initial value determined by the user.

6.5 Cooling algorithms

Remark: the following nomenclature for the schedules is not consistent in literature. Therefore these schedules may be named different than the names given in SATFO. The following cooling algorithms are implemented:

1. **Geometric** [11, 12, 13]: $c_{k+1} = \alpha * c_k$, $\alpha < 1.0$
2. **Dynamic geometric** (not tested yet) [11, 12, 13]: $c_{k+1} = \frac{c_k}{1 + \frac{c_k \ln(1+\delta)}{3\sigma c_k}}$
3. **Linear** [23]: $c_{k+1} = c_k - \delta$, $\delta = \text{stepsize}$
4. **Geometric 2** [20]: $c_{k+1} = \frac{c_0}{1+t}$,
 c_0 is initial value of c , t = number of Markov chain.
5. **Logarithmic** [20]: $c_{k+1} = \frac{c_0}{\log(1+t)}$
6. **Exponential** [21]: $c_{k+1} = \alpha^t$

6.6 Acceptance algorithms

Transition is accepted if $U(0, 1) < x$.

The following acceptance algorithms are implemented:

1. **Metropolis criterion** [11, 12, 13]: $x = e^{-\frac{f_{new} - f_{old}}{c_k}}$, $f_{new} > f_{old}$.
2. **Sigmoide criterion** [24]: $x = \frac{1}{1 + e^{-\frac{f_{new} - f_{old}}{c_k}}}$
3. **Generalized simulated annealing** [25, 26]. $x = e^{-\beta \frac{c_{new} - c_{current}}{c_{current} - c_{opt}}}$. c_{opt} denotes the expected minimum function value.

6.7 Ending Markov chain

The following stop criteria are implemented to exit a Markov chain:

1. Minimum number of transitions exceeded [11, 12, 13]
2. Maximum number of steps in Markov chain exceeded [11, 12, 13]
3. Constant function values in last n steps [19]
4. Maximum number of rejections in Markov chain exceeded [13]

6.8 Ending simulated annealing

The following stop criteria are implemented to exit simulated annealing:

1. Control parameter less than minimum value [11, 12, 13]
2. Constant function values in last n steps [19]
3. Control parameter or function value less than machine precision
4. Cost function value less than minimum value
5. Acceptance ratio less than minimum value [23]

6.9 Boltzmann annealing

The standard Boltzmann annealing algorithm can be set by choosing the logarithmic cooling algorithm and the Gauss generating function [11, 20, 12, 13].

6.10 Fast Simulated Annealing

Fast simulated annealing can be set by choosing the second geometric cooling algorithm and a Cauchy generating function [11, 20, 12, 13]. The use of the Cauchy distributions ensures that large steps are more likely to occur than in case of the Gaussian distribution. Therefore, cooling can occur faster.

6.11 Generalized Simulated Annealing

Generalized simulated annealing does not use a cooling scheme. Instead it makes use of the function values themselves. The acceptance probability is given by $P = e^{-\beta \frac{c_{new} - c_{current}}{c_{current} - c_{opt}}}$, where c_{opt} denotes the expected minimum function value. β is a constant that must be such that the acceptance probability is $0.5 \leq P \leq 0.9$. Probabilities close to zero lead to inefficient searches (almost every transition is accepted), whereas probabilities much below 0.5 require too many function evaluations to escape from local minima. The probability of accepting an detrimental step depends on the magnitude of the change in function values $c_{new} - c_{current}$. If the GSA approaches C_{opt} ,

the probability P decreases, and therefore less detrimental steps are accepted. If the estimated value c_{opt} is too large, then $c_{current} - c_{opt}$ will eventually become zero, and a new estimate is necessary.

7 Tutorial and Example

The simulated annealing algorithm was applied to conformational analysis of a peptide. The cost function comprised terms for restraint violations and Vanderwaals overlap.

The resulting data can be found in the `tutorial_data` directory. This directory also contains the configuration file `SA.set`.

For this *measuring* experiment, a standard *Gauss parameter generating function* was used with a the *step size* parameter set to 0.05. The *initial temperature* was set manually to 3.0. This gave an initial acceptance ratio close to one. The length of the *first Markov chain* to equilibrate the system was set to 5000. Since the skip parameter *equilibrium* was set to 25, this resulted in 200 function values used for determining this equilibrium. The *second Markov chain* was set to a minimum of 10000 transitions with a maximum of 20000 attempts. The skip parameters *function values* and *specific heat* were set to 25 and 250 respectively. The temperature was cooled by using the *geometric* scheme with α set to 0.95. The *Metropolis acceptance criterion* was used for accepting detrimental steps. Simulated annealing was terminated if the acceptance ratio or temperature dropped below the specified thresholds.

The results are shown in Figure 3, 4, and 5.

Figure 3: Output of SATFO visualized by the matlab function `Display.m`

Figure 4: Mean function value and standard deviation for each Markov chain. Visualized with the matlab function Display2.m

8 Acknowledgements

The authors acknowledge N.M. Faber for his suggestions with respect to the measurement part of the toolbox.

Figure 5: Specific heat and standard deviation for each Markov chain. Visualized with the matlab function Display2.m

References

- [1] B. Hajek. Cooling Schedules for Optimal Annealing. *ACM Transactions on Mathematical Software*, 13(2):311, 1988.
- [2] KirkPatrick S., Gelatt, C.D., Vecchi, M.P. Optimizing by Simulated Annealing. *Science*, 220(4598):671, 1998.
- [3] Groot, R.D., Eerden, J.P. van der, Faber, N.M. The Direct Correlation Function in Hard Sphere Fluids. *J. Chem. Phys.*, 87(4), 1987.
- [4] Corana, A., Marchesi, M., Martini, C., Ridella, S. Minimizing Multimodal Functions of Continuous Variables with the 'Simulated Annealing' Algorithm. *ACM Transactions on Mathematical Software*, 13(3):262, 1987.
- [5] Collins, N.E., Eglese, R.W., Golden, B.L. Simulated Annealing - An Annotated Bibliography. *American Journal of Mathematical and Management Sciences*, 8(3-4):209–307, 1988.
- [6] Metropolis, N., Rosenbluth, A., Rosenbluth, A., Teller, E., Teller, J. *J. Chem. Phys.*, 21:1087, 1953.
- [7] Lucasius, C.B., Kateman, G. GATES towards evolutionary large-scale optimization: A software-oriented approach to genetic algorithms. Part 1: General perspective. *Computers & Chemistry*, 18(2):127, 1994.
- [8] Lucasius, C.B., Kateman, G. GATES towards evolutionary large-scale optimization: A software-oriented approach to genetic algorithms. Part 2: Toolbox description. *Computers & Chemistry*, 18(2):137, 1994.
- [9] James, F. A Review of pseudorandom number generators. *Computer Physics Communications*, 60:329–344, 1990.

- [10] Marsaglia, G., Zaman, A., Tsang, W.W. Toward a Universal Random Number Generator. *Statistics and Probability Letters*, 8:35–39, 1990.
- [11] Aarts, E., Korst, J. *Simulated Annealing and Boltzmann machines*. John Wiley & Sons, Chichester, 1990.
- [12] Laarhoven, P.J.M., Aarts, E.H.L. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, Dordrecht, 1987.
- [13] P.J.M. Laarhoven. *Theoretical and computational aspects of simulated annealing*. PhD thesis, Erasmus Universiteit Rotterdam, 1987.
- [14] L. Ingber. *Adaptive Simulated Annealing (ASA)*. ftp.caltech.edu in /pub/ingber.
- [15] L. Ingber. Very Fast Simulated Re-Annealing. *J. Mathl. Comut. Modelling*, 12:967–973, 1989. ftp.caltech.edu in /pub/ingber.
- [16] L. Ingber. Simulated Annealing: Practice Versus Theory. ftp.caltech.edu in /pub/ingber, 1993.
- [17] L. Ingber. Genetic Algorithms and Very Fast Simulated Reannealing: a Comparison. *Mathl. Comput. Modelling*, 16(11):87–100, 1992.
- [18] A.T. Bharucha-Reid. *Elements of the Theory of Markov Processes and their Applications*. McGraw-Hill Book Company, Inc., New York, 1960.
- [19] S. Kirkpatrick. Optimization by Simulated Annealing: Quantitative Studies. *Journal of Statistical Physics*, 34:975, 1984.
- [20] Szu, H., Hartley, R. Fast Simulated Annealing. *Physics Letters A*, 122(3,4):157, 1987.
- [21] F. Daalmans. Simulated annealing: een overzicht van theorie en toepassingen. Scriptie 382, University of Nijmegen, Department for Analytical Chemistry, 1992.
- [22] S. Kirkpatrick. Optimization by Simulated Annealing. Research Report RC9355, IBM, 1992.
- [23] Barakat, M.I., Dean, P.M. Molecular Structure Matching by Simulated Annealing. A Comparison Between Different Cooling Schedules. *Journal of Computer-Aided Molecular Design*, 4:295–316, 1990.
- [24] P.C. Schuur. Classification of Acceptance Criteria for the Simulated Annealing Algorithm. Technical report COSOR 89-29, Vakgroep Besliskunde en Stochastiek, TU Eindhoven, 1989.
- [25] Sutter, J.M., Kalivar, J.H. Convergence of Generalized Simulated Annealing with Variable Step Size with Application Toward Parameter Estimations of Linear and Non-Linear Models. *Analytical Chemistry*, 63:2383–2386, 1991.

- [26] Bohachevsky, I.O., Johnson, M.E., Stein, M.L. Generalized simulated annealing for function optimization. *Technometrics*, 28(3):209, 1986.