Exact optimization by means of sequentially adaptive Bayesian learning

John Geweke Bart Frischknecht*

July 28, 2014

Abstract

Simulated annealing is a well-established approach to optimization that is robust for irregular objective functions. Recently it has been improved using sequential Monte Carlo. This paper presents further improvements that yield the global optimum with accuracy constrained only by the limitations of floating point arithmetic. Performance is illustrated using a standard set of six test problems in which simulated annealing has had mixed success. Our approach reliably finds the exact global optimum in all six cases, and with fewer function evaluations than competing simulated annealing algorithms. This approach is a specific case of the sequentially adaptive Bayesian learning algorithm, which uses feedback from particles to the design of the algorithm. The feature of this algorithm most critical to exact optimization is targeted tempering, a new technique developed in this paper.

*Geweke: University of Technology Sydney and Erasmus Unviersity Rotterdam, John.Geweke@uts.edu.au; Frischknecht: Vennli, bart.frischknecht@vennli.com. Both authors acknowledge the support of Australian Research Council grant DP130103356. Geweke acknowledges support through the ARC Centre of Excellence for Mathematical and Statistical Frontiers of Big Data, Big Models, New Insights, ARC grant CD140100049. Frischknecht was at UTS when most of the research reported was conducted. We thank Enlu Zhou for kindly providing the code accompanying Zhou and Chen (2013) and William J. McCausland for useful discussions.

1 Introduction

Irregular global optimization problems arise routinely in business and applied science. The simulated annealing algorithm (Kirkpatrick et al., 1983; Černý, 1985) has proved competitive in many global optimization problems in $\mathbb{R}^d$ (Černý, 1984; Vanderbilt and Louie, 1984; Geman and Hwang, 1986; Aluffi-Pentini et al., 1985; Dekkers and Aarts, 1991; Belisle, 1992). Variants of the algorithm using multiple values of the argument vector x simultaneously emerged only slightly later (Aarts et al., 1986a, 1986i; Xue, 1994; Hamma et al., 2000). More recent contributions (Molvalioglu et al., 2007, 2009; Zhou and Chen, 2013) also utilize multiple values of x, known as particles.

The methods introduced in this paper use particles, and they also draw on sequential Monte Carlo methods (Liu and Chen, 1998; Doucet et al., 2000; Del Moral et al., 2006), as did Zhou and Chen (2013). What is fundamentally different in our approach is the recognition that the particles themselves constitute a rich source of information about the function $h(x)$ for which $x^* = \arg\max_{x \in X} h(x)$ is the desideratum. This information can be used to adapt the algorithm continually, leading to much more efficient execution than could ever be achieved in this class of algorithms with design fixed ex ante. The adaptive scheme set forth here is one component of a yet more general class of algorithms that subsume both integration problems and optimization problems of various kinds. We refer to this class of algorithms as sequentially adaptive Bayesian learning (SABL), implemented in software of the same name. This approach results in two major advances relative to previous approaches using simulated annealing.

One advance is that it frees the investigator from the time consuming burden of retuning the algorithm manually each time it is applied to a new problem. For example, a major challenge in using simulated annealing of all kinds is the choice of an effective cooling schedule (Gidas, 1985; Hajek, 1988; Hajek and Sasaki, 1989; Kohn and Fielding, 1999). The adaptation in SABL for optimization incorporates targeted tempering, introduced in this paper for the first time. Early in each cycle the algorithm uses the existing particles to determine the incremental cooling that guarantees a stated effective sample size target. Thus the specification of the cooling schedule reduces to the choice of a single number and, moreover, the algorithm is robust to a wide range of effective sample size targets. This is one of a handful of algorithm hyperparameters, and the SABL software has default values for each. This paper uses the same test problems as Zhou and Chen (2013), and with a single exception detailed subsequently the SABL default values performed well. We spent almost no time in problem-specific tuning.

The second major advance is somewhat more surprising – at least, we did not anticipate it at the start of the research reported here. For the six test problems the SABL optimization algorithm determines $h^* = \max_{x \in X} h(x)$ with h^* having the smallest approximation error that is possible given machine representation of floating point numbers. In the common IEEE 754-1985 binary64 (double precision) standard this error is $\varepsilon \cdot |h(x)|$, $\varepsilon = 2^{-52} \approxeq 2.22 \times 10^{-16}$. The algorithm produces a set $X_L^* \subseteq X$, with all $x^* \in X_L^*$ having exact floating point representation and $h(x^*) = h^*$. In the test problems h is twice-differentiable in a neighborhood of its maximum and the number of

such x^* is very large – easily greater than the number of particles that can reasonably be employed. Repeated executions of the algorithm, with different seeds for the random number generator, always lead to exactly the same h^* but not exactly the same X_L^*.

The organization of the paper is straightforward. Section 2 develops the analytical framework for optimization using SABL and includes a self-contained overview of non-adaptive sequential Monte Carlo algorithms. Section 3 introduces the adaptation that is essential to the effectiveness of the optimization algorithm. Section 4 reports the performance of the SABL optimization algorithm in the test problems and compares it with the performance of the algorithm of Zhou and Chen (2013), which in turn shows that algorithm compares favorably with a number of other variants of simulated annealing. There is a short concluding section.

2 Theory

This section develops the theoretical foundations of our approach, which has much in common with both sequential Monte Carlo and simulated annealing but is not a special case of either.

2.1 Tempered distributions, upper contour sets and the global maximum

The objective considered here and in the relevant literature is to maximize the function $h(x)$ defined on a set $X \subseteq \mathbb{R}^d$.

Condition 1 *Core conditions:*

1. *The support X of $h(x)$ has finite Lebesgue measure, $M(X) < \infty$.*
2. *The objective function h is Lebesgue-measurable on X.*
3. $\overline{h} = \sup_{x \in X} h(x) < \infty$.

In both simulated annealing and SABL the set X is also the support of a probability distribution from which the particles are drawn initially.

Condition 2 *The initial probability distribution is absolutely continuous. The associated probability density $\pi(x)$ is boundde below and above: $0 < \underline{\pi} < \pi(x) < \overline{\pi} < \infty \ \forall \ x \in X$.*

Our research strategy has its roots in the close similarity of this environment and that of Bayesian inference. Suppose we formally regard $\pi(x)$ as the prior density of x and the Boltzman transformation of h at temperature T, $\exp[h(x)/T]$, as the likelihood function in x. Then Conditions 1 and 2 imply the conditions of Proposition 2 in Durham and

Geweke (2015) stating that as the number of particles in the SABL algorithm increases their distribution converges to that of the kernel density

$$f(x) = \pi(x) \exp[h(x)/T]. \tag{1}$$

While there are no data, as such, in optimization problems, decreasing temperature T is formally analogous to incorporating additional observations. By making T quite small the distribution can be concentrated on those values of x corresponding to the largest values of $h(x)$. If there is a global mode x^* then the distribution concentrates around x^*. We now develop these ideas explicitly.

Index the upper contour sets of the function h by $\varepsilon > 0$:

$$X(\varepsilon) = \left\{x : h(x) > \overline{h} - \varepsilon\right\},$$

$\overline{h}$ having been defined in Condition 1. Let P_T denote the probability measure on X defined by T in the kernel density (1). Condition 1 implies that $X(\varepsilon)$ is both Lebesgue- and P_T-measurable.

Condition 3 $M[X(\varepsilon)] > 0 \ \forall \ \varepsilon > 0$.

Let S^c denote the complement of a set $S \subseteq X$ with respect to X.

Proposition 1 *Given conditions 1 through 3 and any $\varepsilon > 0$,*

$$\lim_{T \to 0} P_T[X(\varepsilon)] = 1.$$

Proof. The statement is trivially true if $M[X(\varepsilon)^c] = 0$, so suppose $M[X(\varepsilon)^c] > 0$ and thus $P_T(X(\varepsilon)^c) > 0$.

$$\begin{aligned}
\frac{P_T[X(\varepsilon)]}{P_T[X(\varepsilon)^c]} &\geq \frac{P_T[X(\varepsilon/2)]}{P_T[X(\varepsilon)^c]} \geq (\underline{\pi}/\overline{\pi}) \cdot \frac{M[X(\varepsilon/2)]}{M([X(\varepsilon)]^c)} \cdot \frac{\exp\left[\left(\overline{h} - \frac{\varepsilon}{2}\right)/T\right]}{\exp\left[\left(\overline{h} - \varepsilon\right)/T\right]} \\
&= (\underline{\pi}/\overline{\pi}) \cdot \frac{M[X(\varepsilon/2)]}{M([X(\varepsilon)]^c)} \cdot \exp\left(\frac{\varepsilon}{2T}\right) \longrightarrow \infty.
\end{aligned}$$

■

Thus under the stated conditions T can be chosen sufficiently low to make the posterior distribution with density kernel (1) concentrate almost all of its probability on an arbitrarily exclusive upper contour set of h. Given somewhat stronger conditions this is also true of a neighborhood of the global mode.

Condition 4 *The function h has a unique global mode x^* on X: if $x \in X$ and $x \neq x^*$ then $h(x) < h(x^*)$.*

Denote the open ball $B(x^*, \delta) = \left\{x : (x - x^*)'(x - x^*) < \delta^2\right\}$.

Condition 5 $\forall \ \delta > 0 \ \exists \ \varepsilon > 0 : X(\varepsilon) \subseteq B(x^*, \delta)$.

Proposition 2 *Given conditions 1 through 5,*

$$\lim_{T \to 0} P_T[x \in B(x^*, \delta)] = 1.$$

2.2 Non-adaptive sequential Monte Carlo

A nonadaptive sequential Monte Carlo algorithm generates random vectors x_i $(i = 1, \ldots, n)$ that are ergodic for a probability distribution with probability density kernel

$$\pi(x)\, k(x) \tag{2}$$

with respect to Lebesgue measure. The function $\pi(x)$ is a proper probability density. In Bayesian inference $\pi(x)$ is a proper prior probability density, $k(x)$ is a likelihood function, and (2) constitutes the proper kernel of the distribution of a parameter vector x conditional on data. In this work $\pi(x)$ is the probability density of the initial distribution and $k(x) = \exp[h(x)/T]$. Sequential Monte Carlo evolved from the particle filtering literature and for this reason the random vectors x_i $(i = 1, \ldots, n)$ are known as particles.

The algorithm begins with independent and identically distributed particles $x_i^{(0)} \sim \pi(x)$. It proceeds through cycles $\ell = 1, \ldots, L$, applying the correction component $k_\ell(x)$ and mutation component G_ℓ in cycle ℓ. Each cycle consists of three phases. At the start of cycle ℓ denote the particles $x_i^{(\ell-1)}$ $(i = 1, \ldots, n)$.

The correction phase of cycle ℓ constructs a weight for each particle

$$w\left(x_i^{(\ell-1)}\right) = k_\ell\left(x_i^{(\ell-1)}\right) / k_{\ell-1}\left(x_i^{(\ell-1)}\right) \quad (i = 1, \ldots, n). \tag{3}$$

The weights may well be regarded as importance sampling weights when the source density is $k_{\ell-1}$ and the target density is k_ℓ. The function w must be bounded on X. The specification of the sequence k_ℓ, with $k_0(x) = 1$ and $k_L(x) = k(x)$, is one of two components that constitute the design of the sequential Monte Carlo algorithm.

The selection phase of cycle ℓ next assigns the probability function

$$\rho_i = w\left(x_i^{(\ell-1)}\right) \Big/ \sum_{j=1}^{n} w\left(x_j^{(\ell-1)}\right) \quad (i = 1, \ldots, n)$$

to the particles. It samples n particles with replacement from this distribution, using residual resampling (Douc et al., 2005). This constitutes a new set of particles $x_i^{(\ell,0)}$.

The mutation phase of cycle ℓ next subjects each particle to a transition

$$x_i^{(\ell)} \sim dG_\ell\left(\ \cdot\ \mid x_i^{(\ell,0)}\right).$$

The transition density dG_ℓ must be invariant with respect to the kernel (2):

$$\int_X \pi(y)\, k_\ell(y)\, dG_\ell(x \mid y)\, dy = \pi(x)\, k_\ell(x). \tag{4}$$

The specification of the transition densities dG_ℓ is the second of the two components that constitute the design of the sequential Monte Carlo algorithm.

The final set of particles is $x_i = x_i^{(L)}$ $(i = 1, \ldots, n)$. These particles are ergodic for the distribution with probability density kernel (2): i.e., if

$$E\left[g\left(x\right)\right] = \frac{\int g\left(x\right)\pi\left(x\right)k\left(x\right)dx}{\int \pi\left(x\right)k\left(x\right)dx}$$

exists then

$$\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n} g\left(x_i\right) = E\left[g\left(x\right)\right]$$

almost surely. This result is established by the work of Chopin (2002, 2004).

The sequential Monte Carlo simulated annealing algorithm of Zhou and Chen (2013) is almost a specific case of a non-adaptive sequential Monte Carlo algorithm. The initial density $\pi\left(x\right)$ is uniform on X, which for the test problems is the hypercube $\left[-50, 50\right]^d$. The sequence k_ℓ of the correction component is defined recursively:

$$\begin{aligned} k_1\left(x\right) &= \exp\left[\lambda\frac{h\left(x\right)}{T_1}\right] + 10^{-30}, \\ k_\ell\left(x\right) &= k_{\ell-1}\left(x\right)\cdot\exp\left[\lambda h\left(x\right)\left(\frac{1}{T_\ell} - \frac{1}{T_{\ell-1}}\right)\right] + 10^{-30} \quad \left(2 = 1, \ldots, L\right), \end{aligned} \tag{5}$$

where

$$T_\ell = \frac{\max_i\left|h\left(x_i^{(\ell-1)}\right)\right|}{10\log\left(\ell+1\right)} + 10^{-10}. \tag{6}$$

Both λ and L must be chosen and are problem-specific in practice. The transition density of the mutation component is a Metropolis random walk with candidates

$$\widetilde{x}_i^{(\ell)} \sim N\left(x_i^{(\ell,0)},\ \alpha\beta^\ell I_d\right). \tag{7}$$

Both α and β must be chosen and are problem-specific in practice. Finally the number of particles n must be chosen and is also problem-specific in practice.

This algorithm is not quite a specific case of a non-adaptive sequential Monte Carlo algorithm because the particles themselves enter into the correction component of the algorithm in (6).

3 Adaptation

For the ergodicity of the non-adaptive sequential Monte Carlo algorithm to be useful, the correction and mutation components must be designed so that a reliable approximation of the distribution with probability density kernel (2) is achieved in real time. Sensible choices of these components, coupled with a number of particles n consistent with reasonable execution time, more often than not lead to approximations that are

unreliable: that is, results from repeated executions of the algorithm lead to unsatisfactorily disparate results. This problem is familiar in the simulated annealing literature in the choice of the cooling schedule, which corresponds to the choice of the correction component in the non-adaptive sequential Monte Carlo algorithm. Experienced and skilled users often devote considerable time to tuning these algorithms to the problem at hand in order to achieve satisfactory results. The sequential Monte Carlo simulated annealing algorithm of Zhou and Chen (2013) has five tuning parameters: the number of cycles L, the correction component parameter λ in (6), the parameters α and β in the mutation component (7), and the number of particles n. In Zhou and Chen (2013) these choices are problem specific.

Multi-particle algorithms afford the opportunity for algorithmic rather than manual tuning. This idea goes back at least to the notion of effective sample size introduced in Liu and Chen (1995). In the context of the sequential Monte Carlo algorithm described in Section 2.2, this amounts to using information in the particles $x_i^{(\ell-1)}$ to choose the kernel k_ℓ. Section 3.1 shows how the temperature in the correction component can be selected to guarantee a stipulated effective sample size. This is a new idea that removes the major headache of stipulating a cooling schedule for each optimization problem.

Similar strategies have been developed for the mutation component. Section 3.2 reviews one such strategy that was proposed in Durham and Geweke (2015), incorporated in SABL, and turns out to be quite effective in the test problems of Zhou and Chen (2013).

The existing literature, including Chopin (2002, 2004) and papers that have followed, cannot be used to demonstrate ergodicity of these adaptive algorithms. Feedback from particles to the correction and mutation components is not permitted in most of this theory – the notable exception being the work of Del Moral et al. (2012) on using effective sample size for the correction component. Section 3.3 outlines a procedure that fully addresses this problem, introduced in Durham and Geweke (2015).

All of these developments presume the specification of the final temperature T in (1), carried forward to (2) in the previous section. Successful manual specification of T is problem-specific. If T is too high then the particles will not cluster sufficiently about the true mode for practical purposes. On the other hand, for T sufficiently low, 64-bit floating point representation becomes a poor approximation of the actual distributions, calling into question the relevance of any of the approximation theory. Moreover, this range is problem specific. Section 3.3 demonstrates how to use the distribution of $h\left(x_i^{(\ell)}\right)$ to identify the point at which the limitations of machine precision have been reached, and therefore the global maximum has been determined to the greatest precision possible with 64-bit floating point representation. This procedure is generic, not problem specific.

3.1 Targeted tempering

In a tempered correction component the sequence of kernels in the sequential Monte Carlo algorithm is

$$k_\ell(x) = \exp\left[h(x)/T_\ell\right].$$

If the sequence $\{T_\ell\}$ is monotonically decreasing from $T_0 = \infty$ to $T_L = T$, then $k_0(x) = 1$, $k_L(x) = k(x)$ and $k_\ell(x)/k_{\ell-1}(x)$ is bounded on X, consistent with the requirements of Section 2.2 for the correction component. The weight function in (3) is

$$w(x) = k_\ell(x)/k_{\ell-1}(x) = \exp\left[\left(\frac{1}{T_\ell} - \frac{1}{T_{\ell-1}}\right)\cdot h(x)\right] = \exp\left[r_\ell h(x)\right], \tag{8}$$

which defines the derived sequence $\{r_\ell\}$.

The effective sample size in the correction phase of cycle ℓ is the function

$$ESS = \frac{\left[\sum_{i=1}^n w\left(x_i^{(\ell-1)}\right)\right]^2}{\sum_{i=1}^n w\left(x_i^{(\ell-1)}\right)^2} \tag{9}$$

of the particle weights (3). Relative effective sample size is $RESS = ESS/n$. If all the weights are the same then $ESS = n$ and $RESS = 1$: effective and actual sample size are the same. If only one weight is positive then $ESS = 1$ and $RESS = 1/n$. In applications of sequential Monte Carlo to Bayesian inference problems values of $RESS$ in the range $(0.1, 0.9)$ have been found effective. Adaptive algorithms often specify a target value η^0 that is used in the algorithm that determines the next kernel k_ℓ. Only target values in the range $(0, 1)$ are sensible. (Throughout we use the superscript 0 to designate the hyperparameters of the SABL optimization algorithm. SABL assigns default values to all of these hyperparameters, and any or all of them can easily be changed if desired.)

From (8) and (9),

$$RESS(r_\ell) = \frac{\left[\sum_{i=1}^n w\left(x_i^{(\ell-1)}\right)^{r_\ell}\right]^2}{n\sum_{i=1}^n w\left(x_i^{(\ell-1)}\right)^{2r_\ell}} = \frac{\left\{\sum_{i=1}^n \exp\left[r_\ell h\left(x_i^{(\ell-1)}\right)\right]\right\}^2}{n\sum_{i=1}^n \exp\left[2r_\ell h\left(x_i^{(\ell-1)}\right)\right]}.$$

With $\eta^0 \in (0, 1)$ the equation $RESS(r_\ell) = \eta^0$ has exactly one solution, $r_\ell > 0$, and the solution can be computed quickly and reliably. Thus it is possible to choose r_ℓ so that relative effective sample size in the correction component is exactly the target value η^0. The temperature in cycle ℓ is $T_\ell = \max\left[T_{\ell-1}/\left(1 + r_\ell T_{\ell-1}\right),\ T\right]$. If $T_\ell = T$ then $\ell = L$ and the algorithm terminates at the end of the cycle at which point the particles constitute the final values x_i $(i = 1, \ldots, n)$.

This defines an adaptive correction component that we term targeted tempering. It reduces the manual selection of kernels k_ℓ, and the choice of a cooling schedule T_ℓ in simulated annealing, to a single choice η^0 of the relative effective sample size criterion. The value $\eta^0 = 0.5$, the default value in SABL, works well for the six test problems discussed in Section 4.

3.2 Adaptive mutation

Following resampling in the selection phase the particles $x_i^{(\ell,0)}$are identically but not independently distributed. In particular some particles, those with high weights in the correction phase, have multiple copies. A rule of thumb is that if the particles $x_i^{(\ell-1)}$ are all distinct at the start of the correction phase, then the number of distinct particles at the end of the selection phase is roughly equal to the effective sample size. Thus given $\eta^0 = 0.5$ there will be about $n/2$ distinct particles.

The role of the mutation phase is to break the dependence induced by multiple copies of the same particle. It is not enough that the essential condition (4) be satisfied. If the dependence is not broken sufficiently then multiple copies of many particles will be passed on to the next cycle $\ell + 1$, and dependence cumulates. A completely ineffective mutation phase leaves all particles unchanged, and then given sufficiently low T all particles are copies of $x = \underset{i=1,\ldots,n}{\arg\max} h\left(x_i^{(0)}\right)$. Durham and Geweke (2015) discusses how to monitor the effectiveness of the mutation phase.

SABL uses a Metropolis random walk in the mutation phase, as does Zhou and Chen (2013). But rather than use a fixed candidate distribution like (7), SABL uses the information in the particles about the shape of the probability density kernel $\pi(x) \cdot k_\ell(x)$ to construct the variance matrix in the normal source distribution of a Metropolis random walk. It repeats the Metropolis random walk, measuring the dependence amongst the particles at the end of each step, and the mutation phase ends when the degree of dependence achieves a prespecified value. The remainder of the section describes these two procedures in turn.

Index the Metropolis steps in the mutation phase by $s = 1, \ldots, S_\ell$ and denote the particles at the end of step s by $x_i^{(\ell,s)}$, with $x_i^{(\ell)} = x_i^{(\ell,S_\ell)}$. In each step s construct the sample variance matrix $V^{(\ell,s)}$ of the particles $x_i^{(\ell,s-1)}$. Then draw the candidates

$$\widetilde{x}_i^{(\ell,s)} \sim N\left(x_i^{(\ell,s-1)}, c^{(\ell,s)} V^{(\ell,s)}\right),$$

where $c^{(\ell,s)}$ is a scaling factor. Denote the acceptance rate by $\alpha^{(\ell,s)}$ and let α^0 be the target acceptance rate. For $s = 1$, $c^{(1,1)} = \gamma_0^0$. The scaling factor is

$$c^{(\ell,s)} = \begin{cases} \min\left(c^{(\ell,s-1)} + \delta^0, \overline{\gamma}^0\right) & \text{if } \alpha^{(\ell,s-1)} > \alpha^0 \\ \max\left(c^{(\ell,s-1)} - \delta^0, \underline{\gamma}^0\right) & \text{if } \alpha^{(\ell,s-1)} \leq \alpha^0 \end{cases} \tag{10}$$

If $s = 1$ then $(\ell - 1, S_{\ell-1})$ replaces the superscripts $(\ell, s-1)$ in (10). The default values of the hyperparameters in SABL are $\gamma_0 = 0.5$, $\alpha^0 = 0.25$, $\delta^0 = 0.1$, $\underline{\gamma}^0 = 0.1$ and $\overline{\gamma}^0 = 2$. These values were used in all of the problems taken up in Section 4.

The decision to terminate Metropolis steps in the mutation phase is based on relative numerical efficiency (RNE), a measure of the accuracy of each of the approximations

$$\frac{1}{n}\sum_{i=1}^{n} x_{i,j}^{(\ell,s)} \approxeq \int_X x_j \cdot \pi(x) \exp\left[h(x)/T_\ell\right] dx \quad (j = 1, \ldots, d). \tag{11}$$

RNE is a fraction in which the denominator is the sampling variance of $x_{i,d}$, divided by n. This would be the variance of the approximation error if the particles were independent. The numerator of the fraction is an approximation of the actual approximation error. This is constructed by organizing the particles into J^0 groups of N^0 particles each ($J^0 \cdot N^0 = n$). Independence across groups is enforced by executing the selection phase separately for each group. The approximation (11) is constructed separately for each group, and the variance across groups is then used to infer the approximation error of the approximation using all of the particles. The Metropolis steps terminate if RNE exceeds the value RNE^0, and always terminates if s has attained the upper bound S^0. The default values in SABL are $J^0 = 2^4$, $N^0 = 2^{10}$ (and thus there are $n = 2^{14}$ particles), $RNE^0 = 0.4$ and $S^0 = 100$. These values were used in all of the test problems discussed in Section 4.

SABL provides a number of variants for the mutation phase. One of these is a blocked Metropolis random walk. In this procedure the vector x is partitioned into B blocks, $x' = \left(x'_{(1)}, \ldots, x'_{(B)}\right)$. In step s of the mutation phase the Metropolis step is executed for just one of the blocks, with the candidate distribution constructed from the variance of the particles in that block, the previous scaling factor, and the acceptance rate. The user can specify the blocks, or SABL can construct the blocks randomly. The default in SABL is random blocks, with the number of blocks based on the dimension d of x. This option is often effective when the shape of the density $\pi(x) \exp[h(x)/T_\ell]$, or large d, leads to very low acceptance of candidates for the entire vector. (This, in turn, is indicated by execution of all S^0 steps of the mutation phase with very low values of RNE at the end.) This circumstance arose in test problem 5, taken up in Section 4. The blocked Metropolis random walk with default random blocking proved effective in this problem.

3.3 Convergence

The algorithm as constructed to this point provides particles x_i ($i = 1, \ldots, n$) drawn from the probability distribution with density kernel (2). The sequential Monte Carlo literature does not provide a rigorous justification for this procedure because of the feedback from particles to the algorithm itself described in Sections 3.1 and 3.2. This difficulty can be overcome using the two-pass procedure developed in Durham and Geweke (2015). The first pass proceeds as described, retaining all of the algorithm design parameters: the sequence of temperatures T_ℓ ($\ell = 1, \ldots, L$) of the correction component and the variance matrices $c^{(\ell,s)}V^{(\ell,s)}$ of the mutation component. The second pass then takes these algorithm design parameters as given, with new seeds for the random number generator used to produce the particles. The second pass thus conforms to the non-adaptive sequential Monte Carlo literature for which the theory has been completely developed. SABL facilitates this two-pass procedure.

For optimization there are some particular conceptual problems and practical considerations. The conceptual problems stem from the fact that the objective is to determine

$x^* = \arg\max_{x \in X} h(x)$, not to integrate with respect to the probability density kernel (2). Given Conditions 1 through 5 of Section 2.1, once T is sufficiently small then yet smaller values increase the probability that x^* is within a stated distance of the mean of the particles. While the accuracy of the numerical approximation to the mean can be evaluated as outlined in Section 3.2, this does not address the accuracy of the approximation of the mode. Even to report the mean as the approximate solution is unsatisfactory, given that there are very likely particles x_i for which $h(x_i)$ is larger than h evaluated at the mean of the particles. This is a generic conceptual problem for simulated annealing, including Zhou and Chen (2013).

This leaves the practical consideration of choosing the final temperature T. Recall that that the probability density kernel (2), wih $k(x) = \exp[h(x)/T]$, is a motivating device to permit the application of theory and methods developed in the sequential Monte Carlo literature to multi-particle simulated annealing. Moreover, the sequence of temperatures $\{T_\ell\}$ in the adaptive algorithm is determined sequentially based on the particles $x_i^{(\ell-1)}$ $(i = 1, \ldots, n)$ from the previous cycle.

There is no reason to stop this sequence at any pre-selected temperature. As a practical matter there has to be a stopping rule for the algorithm, but that stopping rule need not be stated in terms of T. In fact a final temperature T is an awkward criterion to use because the implications of any stated T will not be apparent until the algorithm is executed anyway. Our experience thus far suggests two different strategies that are more effective.

The first strategy is to execute the algorithm until the limitations of 64-bit floating point arithmetic are manifest. If these limitations amounted to rounding $h(x)$ to the nearest real number with exact machine representation, then for functions h that are smooth the fraction of particles $x_i : h(x_i) = \max_{j=1,\ldots,n} h(x_i)$ would increase as this limit is approached. This suggests a convergence criterion f^0 stated in terms of this fraction, and in fact that is an option in the SABL software where the default value is $f^0 = 0.5$. Values of this criterion sufficiently close to 1 will eventually encounter the problem that the variance matrix of particles computed for the M phase becomes non-positive definite or is such a poor approximation to particle distribution that there is little further cooling in the C phase.

Beyond the simplest problems the limitations of 64-bit arithmetic are not much like rounding. The longer and more complex the arithmetic required for function evaluation, the greater is the opportunity for errors of evaluation to exceed that implied by a single bit. (Good practice, supported by numerical analysis, seeks to minimize the problem, but the phenomenon exists even with best practice.) The exact outcome can depend on software, on hardware, and even differ from one execution to the next in standard environments with multiple cores and multi-pass compilers. But the objective functions in the test problems in Zhou and Chen (2013), including the detailed illustration, are sufficiently simple that the first strategy is effective. Thus SABL finds solutions of the test problems to machine accuracy.

The second strategy applies when $h(x)$ is expressed in known units of measurement

– very likely to be the case in applications, though it is not in test problems like the ones taken up in Secton 4. For example, if the objective function is measured in US dollars, then there is little point in continuing the algorithm once

$$\max_{j=1,\ldots,n} h\left(x_j\right) - \min_{j=1,\ldots,n} h\left(x_j\right) < \delta^0. \tag{12}$$

with $\delta^0 = 1$. If the problem is maximum likelihood, then $h\left(x\right)$ is the log likelilhood function and a criterion of (say) $\delta^0 = 10^{-6}$ in (12) would likely be regarded as more than adequate by most analysts. Criteria of this form avoid the cost of pointless additional iterations. Save objectives on the scale of US GDP in the first example, they are well safe of the $\left(2.2 \times 10^{-16}\right) \cdot \left|h\left(x\right)\right|$ standard. Of course any problem can in principle be so complex and/or so badly coded that problems arise. This is an issue for any line of attack on on the global optimization problem and the relevant considerations will be specific to the application.

4 Performance

This section assesses the performance of the SABL optimization algorithm and compares it with the performance of the simulated annealing sequential Monte Carlo algorithm of Zhou and Chen (2013), which that paper in turn demonstrates is superior to a number of alternative simulated annealing algorithms. The vehicle for these six exercises is the set of six test problems used in Zhou and Chen (2013). We use the same initial distribution as that paper, uniform on $X_d = \left[-50, 50\right]^d$ in every test problem.

In the first test problem $d = 2$, whereas in the others $d = 10$ or $d = 20$. The two-dimensional x makes it easier to demonstrate some features of the algorithm, and Section 4.1 does so. Section 4.2 undertakes systematic evaluation and comparison for all six test problems.

4.1 A detailed illustration

Test problem 1 is Dejong's 5^{th} function for dimension $d = 2$,

$$h\left(x\right) = -\left[0.002 + \sum_{i=1}^{25} \frac{1}{i + \sum_{j=1}^{d}\left(x_j - a_{ij}\right)^6}\right] \tag{13}$$

with $a_{1\cdot} = e \otimes v$ and $a_{2\cdot} = v \otimes e$, where $e = \left(1, 1, 1, 1, 1\right)$ and $v = \left(-32, -16, 0, 16, 32\right)$. The function has 25 local modes, each near – but not exactly the same as – a point of the form $\left(v_i, v_j\right)$. The modes are not equal and the unique global mode is near the point $\left(-32, -32\right)$. Figure 1 provides contours of $h\left(x\right)$ on X_d in the upper left panel. Successive panels show finer structure on shrinking neighborhoods of the global mode.

The initial distribution is uniform on X_d, as it is in all of the six test problems both in this paper and in Zhou and Chen (2013). The SABL optimization algorithm

terminated after 34 cycles. (This number is random, but varies little on re-execution.) Figure 2 depicts the distribution of the particles at selected cycles. In each panel the axes coincide with the full range of particles so all particles are shown. The particles in the set $X_\ell^* = \left\{ x_i : h\left(x_i^{(\ell)}\right) = \max_{j=1,\ldots,n} h\left(x_j^{(\ell)}\right) \right\}$ are overplotted in red. The similarity of the particle distributions in Figure 2 to the contour plots in Figure 1 is apparent: cycles 2 and 4 evoke the upper left panel, cycle 8 is similar to the upper right panel, cycle 16 exhibits the shape shown in the lower left panel, cycle 24 the lower right panel.

By cycle 28 the function $h(x_i)$ attains its maximum possible value (to machine precision) for some of the particles x_i. These are the particles overplotted in red in Figure 2 for cycles 28 and beyond. The remaining cycles gradually eliminate particles x_i for which $h(x_i)$ is less than this value. By cycle 34 they account for fewer than half the particles and the algorithm terminates. Figure 3 provides greater detail on

$$X_L^* = \left\{ x_i^{(L)} : h\left(x_i^{(L)}\right) = \max_{j=1,\ldots,n} h\left(x_j^{(L)}\right) \right\},$$

the set of particles for which the function attains its maximum value in the last cycle, in four independent runs of the algorithm. The distributions are quite similar in all runs, reflecting the fact that the unconventional form of the distribution arises from machine representation of x_i and h and not from the randomness of particles in the algorithm. Table 1 provides an alternative presentation of this fact.

In moving from the initial distribution of particles to the final one the algorithm moves through three distinct stages that are characteristic of its performance in all of the test problems. Figure 4 documents the adaptation in SABL. The horizontal axis in all 8 panels is cycle number ℓ. Numbering panels from top to bottom and then left to right, panel 1 shows the cooling schedule T_ℓ set in the correction phase and panel 2 shows the corresponding increment to power $T_\ell^{-1} - T_{\ell-1}^{-1}$. Panel 3 documents the ratio of unique to total particles. Panel 4 counts the number of iterations of the Metropolis random walk in the mutation phase, which is limited to a maximum of 100 by the SABL default hyperparameters. Panel 5 shows the average relative numerical efficiency of arguments 1 and 2 at the end of the mutation phase. Panels 6 and 7 report the standard deviation of the particles representing the first argument and the corresponding values $h(x_i)$, respectively. The last panel shows the fraction of the particles that correspond to $\max_{i=1,\ldots,n} h(x_i)$.

In the first stage, cycles 1 through 6, particles represent multiple modes, with the number of modes diminishing. SABL eliminates particles in modes that are not global steadily by means of targeted tempering. This swiftly reduces the weights of particles near modes that are local but not global and leads to multiple copies of particles near the global mode in the selection phase. In computations that are intense relative to other phases, multiple iterations in the mutation phase scattering the particles in that neighborhood of local modes. When these iterations reach their upper bound of 100, RNE drops below its target value of 0.4.

At the end of the first stage the particles in the local modes have been eliminated. This means that temperature can be then be reduced substantially without reducing

effective sample size below its targeted value. The second stage, cycles 7 through 14, is characterized by this rapid cooling. Correspondingly the standard deviation of $h(x_i)$ also drops rapidly. The accomopanying decrease in the standard deviation of x_i is not as great due to the fact that the behavior of $h(x)$ is smooth near the global mode. Scattering of particles in the mutation phase to meet the RNE criterion of 0.4 takes little effort.

At the end of the second stage the process of decreasing temperature from the higher level appropriate to multiple local modes to the lower level for a single mode is complete. In the third stage, cycles 15 through 25, the decrease in temperature is steady but slower as the particles converge toward the mode of the regular distribution represented in the lower panels of Figure 1. Effective scattering of particles in the mutation phase continues to require just a few iterations of the Metropolis random walk.

The final stage of the algorithm, cycles 26 through 34, is characterized by the emergence and increasing dominance of particles for which the machine representation of $h(x_i)$ attains its maximum possible value. These are indicated by the red overplots for cycles 28 and beyond in Figure 2. As temperature decreases in this stage the function $h(x)$ loses the smooth character of its analytical representation (lower right panel of 1) and increasingly assumes the form of a step function. This compromises the effectiveness of the normal source distribution in the Gaussian random walk and it becomes an increasingly poor representation of the local behavior of $h(x)$ evaluated to machine precision. This is reflected in the mutation phase running to the maximum number of iterations in the Metropolis random walk and RNE dropping below its target value. The fraction of particles corresponding to the maximum possible machine representation of $h(x)$ increases steadily until it exceeds one-half and the algorithm terminates.

4.2 Performance of the SABL optimization algorithm

The functions $h(x)$ of the test problems used here and in Zhou and Chen (2013) are stated in the appendix. In all of these problems the elements of x^* always have the same value. In test problems 2 through 6 the values of x^* are known exactly. The first panel of Table 2, rows 1 through 3, documents these values and the dimension d of x.

We began by executing the SABL optimization algorithm using the default algorithm design hyperparameters in SABL. These are documented in Section 3. To recapitulate, there are eight. The algorithm uses $J^0 = 16$ groups of $N^0 = 1,024$ particles each, for a total of 16,384 particles. For the correction component the relative effective sample size criterion is $\eta^0 = 0.5$. For the Metropolis random walk in the mutation component the initial scaling factor is $\gamma^0 = 0.5$, the acceptance rate target is $\alpha^0 = 0.25$, the scaling factor increment factor is $\delta^0 = 0.1$, and the lower and upper bounds for the scaling factor are $\underline{\gamma}^0 = 0.1$ and $\overline{\gamma}^0 = 2$ respectively. These settings proved satisfactory for five of the six test problems. For test problem 5 we used the blocked Metropolis random walk with accompanying SABL default hyperparameters.

In fact, the SABL optimization algorithm performs equally well in determining x^* and $h(x^*)$ with $J^0 = 4$ and $N^0 = 256$, i.e. using one-sixteenth the number of particles.

Execution is faster, but not 16 times faster because there are more cycles and more steps in the mutation phase. The purpose of this observation is to suggest that choosing the number of particles does not seem a pressing issue for these test problems. The main emphasis here is on the fact that the SABL default parameters performed almost perfectly, and that the SABL optimization algorithm reduces almost to the point of elimination the need to engage in tedious problem-specific algorithm tuning exercises. All results reported here used 16,384 particles.

The second panel of Table 2, consisting of rows 4 through 8, documents the accuracy and speed of the SABL optimization algorithm developed in this paper. In test problems 3, 4 and 5 $h^* = h(x^*)$ has an exact 64-bit floating point representation and SABL determines h^* without error. The error bound, computed as the difference between the largest and next-largest values of $h(x_i)$ in the final set of particles, is always the positive distance from h^* to the next larger in magnitude floating point number of the same precision. In the conventional implementation of double precision arithmetic the ratio of the bound to $|h^*|$ is 2.22×10^{-16}. The latter is sometimes called "machine epsilon" and can be determined using the Matlab function `eps`. The error bound exceeds the actual error in the five test problems for which actual error is known.

Row 6 reports the largest range of the arguments x in the set X_L^*. These ranges are from 10^7 to 10^{15} times greater than the error bound for h^*. This reflects the fact that over the range of particles in the final cycles of the SABL optimization algorithm the test functions $h(x)$ are all quite smooth despite the fact that globally they are not.

Row 7 reports the final temperature T_L of the algorithm. This is the temperature T in the final cycle $\ell = L$ for which X_ℓ^* first contained more than $n/2$ particles. Its value varies somewhat from one execution of the algorithm to another. It is interesting to note that T_L is somewhere between 5% and 20% of the value of the h^* error bound (machine epsilon) in every case. Row 8 reports the number of times $h(x)$ was evaluated during the execution of the algorithm, the standard measure of computational efficiency. The number of evaluations varies from one replication to the next, but rarely by more than 10%.

4.3 Performance of the SA/SMC algorithm

The third panel of Table 2, consisting of rows 9 through 20, documents the specific variants of the SA/SMC algorithm used for the test problems in Zhou and Chen (2013) and the performance of the algorithm. The information in the table cannot all be gleaned from the published paper. It is taken from our examination and execution of the code graciously provided by the authors. Our results are consistent with the findings reported for the test problems in Zhou and Chen (2013).

Rows 9 through 13 provide the values of the hyperparameters for these results: the number of particles, the number of cycles, the value of λ in (5), and the values of α and β in (7). These are, in turn, the same as those reported in Zhou and Chen (2013). The vector of hyperparameters is different for each of the test problems. It seems reasonable to infer that some effort was devoted to tuning these hyperparameters. It also seems

reasonable to conjecture that there is a single setting for all the test problems that would provide similar accuracy, but this setting would lead to many more function evaluations in most cases. We did not attempt to explore these issues further by experimenting with these values.

The algorithm as documented in rows 9 through 13 is executed 100 times using different seeds of the random number generators, just as in Zhou and Chen (2013). The results of these 100 executions are the basis of the statistics reported in rows 14 through 19 of Table 2. Each of the 100 executions provides a vector x^* that corresponds to the largest value of $h^* = h(x^*)$ from among all of the particles in the final cycle of the SA/SMC algorithm. Means and standard deviations of h^* and x_1^* are reported in rows 14-15 and 17-18. Standard deviations appear to vary greatly over the six test problems, but this does not have the same substance for artificial problems as it would in a real problem where, for example, h might be measured in dollars or millions of dollars.

Rows 16 and 19 are based on x^{**}, the single particle across all 100 repetitions of the algorithm that provides the largest value $h^{**} = h(x^{**})$. Both x_1^{**} and h^{**} are approximations of the true mode that are much more accurate than the standard errors in rows 15 and 18 would lead one to believe. But, we know the true accuracy only because we know the true solution of the optimization problem, which is never the case in a real problem. This bears out the conceptual problems, raised in Section 3.3, in assessing numerical accuracy of approximations of optima in the presence of simulation noise.

The number of function evaluations in row 20 reflects the fact that the entire algorithm is executed 100 times: each entry is 100 times the product of the entries in rows 9 and 10 for the same column.

4.4 Comparison

The most obvious contrast in the two algorithms is the most important. The SABL optimization algorithm solves the problem to machine precision whereas approximation errors in the SA/SMC algorithm are dominated by simulation noise. The error of approximation of $h(x^*)$ with SABL is machine epsilon, whereas it is impossible to assess the size of this error in the SA/SMC algorithm even as a conceptual matter. However it is clear that the error in the SA/SMC algorithm is many orders of magnitude larger in all of the test problems.

5 Conclusion

It has been generally recognized that sequential Monte Carlo methods provide an approach to optimization, in principle, for at least a decade. Recent efforts in this direction (Molvalioglu et al. (2007, 2009); Zhou and Chen (2013)) are implementations of this principle using algorithms with fixed design. The work here uses a similar approach, but with the critical difference that it incorporates the flexibility and adaptation inherent

in sequentially adaptive Bayesian learning (SABL). An important novel component of adaptation is targeted tempering, introduced in Section 3.1.

This SABL optimization algorithm has several striking advantages compared with these previous attempts.

1. The need for problem-specific tuning of algorithm hyperparameters, like the cooling schedule, is greatly reduced. In the test problems taken up here and in Zhou and Chen (2013) it was virtually eliminated. In previous approaches using simulated annealing of all kinds, considerable effort is typically devoted to tuning the algorithm.

2. Accuracy is limited only by the machine representation of real numbers. With conventional double precision floating point representation the accuracy of the ratio of the error of approximation of $h^* = \max_{x \in X} h(x)$ to h^* itself is about 2.2×10^{-16}. The corresponding approximation error for the argument x^* is problem-specific: for a function that is locally twice differentiable at the mode, it is inversely proportional to the second derivative evaluated at x^*. This elimination of simulation noise in the approximation renders the approximations of x^* and h^* orders of magnitude more accurate than achieved using variants of simulated annealing, including Zhou and Chen (2013).

3. Comparison of the number of function evaluations with previous approaches using simulated annealing depends on the particular approach and problem. For four of the six test problems used to evaluate the SA/SMC algorithm in Zhou and Chen (2013), the number of SABL function evaluations is from 7.2% to 28% the number in SA/SMC; for one problem the number is the same; and for the problem requiring the fewest function evaluations, SABL evaluates $h(x)$ 2.75 times as often.

In real applied problems the computations required to evaluate $h(x)$ will generally be more extensive than those in the test problems examined here. This leads to two important additional considerations. The first is grounded in the fact that the correction and mutation phases of the algorithm are embarrassingly parallel. The selection phase is not, but this phase constitutes a very small fraction of total execution time. Thus the algorithm is well suited to execution on graphics processing units (GPUs), and SABL software implements this option in an efficient and accessible manner. When execution time is bound by the floating point operations in the evaluation of $h(x)$, GPU implementation can decrease execution time by factors between 10 and 100. Conversely when the evaluation of $h(x)$ is relatively trivial, as is the case with the test problems, GPU implementation is in fact slower.

The second consideration for real applied problems is that the limitations of machine arithmetic are more severe and complex than rounding to the nearest bit in floating-point representation would suggest. It becomes impossible to evaluate $h(x)$ to machine precision defined in this way, and the precision that can be achieved is problem-specific.

This limitation cannot be overcome by any algorithm implemented using conventional floating point arithmetic.

Taken together, these factors imply that the SABL optimization algorithm has many attractions in the solution of optimization problems that arise routinely in business and applied science but are sufficiently irregular that conventional steepest ascent algorithms cannot be used. We are currently pursing this potential.

6 Appendix

- Test problem 1: (13) in Section 4.1
- Test problem 2: Powel singular function ($d = 20$)

$$\begin{aligned} h(x) &= -\sum_{i=2}^{d-1} \left[(x_{i-1} + 10x_i)^2 + 5(x_{i+1} - x_{i+2})^2 \right. \\ &\quad \left. + (x_i - 2x_{i+1})^4 + 10(x_{i-1} - x_{i+2})^4 \right] - 0.01 \end{aligned}$$

- Test problem 3: Rosenbrock function ($d = 20$)

$$h(x) = -\sum_{i=1}^{d-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2 \right] - 1$$

- Test problem 4: Griewank function ($d = 20$)

$$h(x) = -\left[\frac{1}{4000} \sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} \cos\left(i^{-1/2} x_i\right) + 1 \right]$$

- Test problem 5: Trigonometric function ($d = 10$)

$$h(x) = -1 - \sum_{i=1}^{d} \left[8\sin^2\left(7(x_i - 0.9)^2\right) + 6\sin^2\left(14(x_i - 0.9)^2\right) + (x_i - 0.9)^2 \right]$$

- Test problem 6: Pintér's function ($d = 10$)

$$\begin{aligned} h(x) &= -\left[\sum_{i=1}^{d} i x_i^2 + \sum_{i=1}^{d} 20i \cdot \sin^2(x_{i-1}\sin x_i - x_i + \sin x_{i+1}) \right. \\ &\quad \left. + \sum_{i=1}^{d} i \log_{10}\left(1 + i\left(x_{i-1}^2 - 2x_i + 3x_{i+1} - \cos x_i + 1\right)^2\right) \right] - 10^{-15} \end{aligned}$$

References

Aarts, EHL, de Bont, FMJ., Habers, JHA. and van Laarhoven, PJM. 1986a. A Parallel Statistical Cooling Algorithm. Proc. STACS 86, Springer Lecture Notes in Computer Science, 210, 87-97.

Aarts, EHL., de Bont, FMJ., Habers, JHA. and van Laarhoven, PJM. 1986b. Parallel Implementations of the Statistical Cooling Algorithm. Integration 4: 209-238.

Belisle CJP. 1992. Convergence theorems for a class of simulated annealing algorithms on $\mathbb{R}^d$. Journal of Applied Probability 29: 885 - 895.

Černý V. 1984. Minimization of Continuous Functions by Simulated Annealing, Research Institute for Theoretical Physics, University of Helsinki, preprint No. HU-TFT-84-51.

Černý V. 1985. Thermodynamical approach to the travelling salesman problem: An efficient solution algorithm. Journal of Optimization Theory and Applications 45:41 - 51.

Chopin N. 2002. A sequential particle filter method for static models. Biometrika 89: 539 - 551.

Chopin N. 2004. Central limit theorem for sequential Monte Carlo methods and its application to Bayesian inference. Annals of Statistics 32: 2385 - 2411.

Cohn H, Fielding M. 1999. Simulated annealing: Searching for an optimal temperature schedule. SIAM Journal on Optimization 9: 779–802.

Collett P, Eckmann JP. 1980. Iterated Maps on the Interval as Dynamical Systems. Boston: Birkhauser.

Dekkers A, Aarts E. 1991. Global optimization and simulated annealing. Mathematical Programming 50: 367 - 393.

Del Moral P, Doucet A, Jasra A. 2006. Sequential Monte Carlo samplers. Journal of the Royal Statistical Society Series B 68: 411 - 436.

Del Moral P, Doucet A, Jasra A. 2012. On adaptive resampling strategies for sequential Monte Carlo methods. Bernoulli 18: 252-278.

Douc R, Cappe P, Moulines E. 2005. Comparison of resampling schemes for particle filtering. 4th International Symposium on Image and Signal Processing and Analysis `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.126.9118`

Doucet A, Godsill S, Andrieu C. 2000. On sequential Monte Carlo sampling methods for Bayesian filtering. Statistics and Computing 10: 197 - 208.

Durham G, Geweke J. 2015. Adaptively sequential posterior simulators for massively parallel computing environments. In Poirier D, Tobias J (eds.) *Advances in Econometrics* (forthcoming).

Geman S, Hwang CR. 1986. Diffusions for global optimization. SIAM Journal on Control and Optimization 24: 1031–1043.

Gidas, B. 1985. Nonstationary Markov chains and convergence of the annealing algorithm. Journal of Statistical Physics 39: 73–131.

Hajek B. 1988. Cooling schedules for optimal annealing. Mathematics of Operations Research 13: 311–329.

Hajek B, Sasaki G. 1989. Simulated annealing - to cool or not. Systems and Control Letters 12: 443–447.

Hamma B, Viitanen S, Törn A. 2000. Parallel continuous simulated annealing for global optimization. Optimization Methods and Software 13: 95-116.

Kirkpatrick S, Gelatt CD, Vecchi MP. 1983. Optimization by simulated annealing. Science 220: 671 - 680.

Kroese DP, Porotsky S, Rubinstein RY. 2006. The cross-entropy method for continuous multi-extremal optimization. Methodology and Computing in Applied Probability 8: 383–407.

Liu JS, Chen R. 1998. Sequential Monte Carlo methods for dynamic systems. Journal of the American Statistical Association 93: 1032 - 1044.

Lozano JA., Larranaga P, Inza I, Bengoetxea E. 2006. Towards a new evolutionary computation: Advances on estimation of distribution algorithms, vol. 192. Springer.

Molvalioglu O, Zabinsky ZB, Kohn W. 2007. Models and algorithms for global optimization in: Optimization and its Applications, pp 215-222. New York: Springer.

Molvalioglu O, Zabinsky ZB, Kohn W. 2009. The interaction particle algorithm with dynamic heating and cooling. Journal of Global Optimization 43: 329 - 356.

Rubinstein R. 1999. The cross-entropy method for combinatorial and continuous optimization. Methodology and Computing in Applied Probability 1: 127–190.

Vanderbilt, D, Louie SG. 1984. A Monte Carlo simulated annealing approach to optimization over continuous variables. Journal of Computational Physics 36: 259-271.

Xue, G. 1994. Molecular conformation on the CM-5 by parallel two-level simulated annealing. Journal of Global Optimization 4:187-208.

Zhou E, Chen X. 2013. Sequential Monte Carlo simulated annealing. Journal of Global Optimization 55: 101 - 124.

Table 1: Properties of the set X_L^* in four executions of the algorithm

Property*	Run 1	Run 2	Run 3	Run 4
$\text{Mean}\,(x_1 \in X_L^*) + c$	2.45e-8	7.82e-8	-4.05e-8	-1.88e-8
$\text{Mean}\,(x_2 \in X_L^*) + c$	1.74e-8	-2.14e-8	7.96e-8	1.88e-8
$\text{Max}\,(x_1 \in X_L^*) - \text{Min}\,(x_1 \in X_L^*)$	3.29e-6	3.31e-6	3.26e-6	3.26e-6
$\text{Max}\,(x_2 \in X_L^*) - \text{Min}\,(x_2 \in X_L^*)$	3.31-6	3.31e-6	3.29e-6	3.29e-6

*$c = 31.978334315250328$

Table 2: Performance of the SA/SMC and SABL optimization algorithms

	Test problem	1	2	3	4	5	6
1	d	2	20	20	20	10	10
2	h^*	$\approxeq -0.998$	-0.01	-1	0	-1	-1e-15
3	x^*	$\approxeq -31.978$	0.00	1	0	0.9	0
	SABL optimization						
4	h^* error	*	2e-19	0	0	0	8.0e-32
5	h^* error bound	2.2e-16	1.7e-18	2.2e-16	2.2e-16	2.2e-16	2.0e-31
6	Max x_i^* range	3.2e-6	1.0e-7	2.6e-8	1.0e-6	1.9e-9	5.3e-16
7	Final $T = T_L$	4.1e-17	1.3e-19	1.2e-17	1.1e-17	3.5e-17	1.8e-32
8	Evaluations	1.1e7	3.9e7	7.3e7	2.8e7	3.3e7	2.9e7
	SA/SMC optimization						
9	# particles	200	200	1000	200	1000	200
10	# cycles	200	2000	5000	5000	4000	20,000
11	Annealing λ	0.1	0.001	0.001	0.001	0.001	0.001
12	Metropolis α	10	10	10	10	10	10
13	Metropolis β	0.995	0.995	0.998	0.998	0.998	0.998
14	Mean h^* error	-1.2e-8	-0.0025	-3.657	-0.0010	-0.446	-0.648
15	Std h^* error	2.8e-8	0.0016	0.768	0.0103	0.386	4.559
16	h^{**} error	<1e-8	-0.0005	-0.0008	-1.1e-7	-0.0002	-3.8e-12
17	Mean x_1^* error	-0.0181	-0.0091	-0.3589	-0.062	0.0205	-0.0003
18	Std x_1^* error	0.0302	0.0703	0.7680	0.628	0.1776	0.0073
19	x_1^{**} error	-0.0011	-0.0105	-0.0004	-0.0001	0.0066	6.6e-8
20	Evaluations	4.0e6	4.0e7	5.0e8	1.0e8	4.0e8	4.0e8

*No exact (analytical) solution avialable for comparison

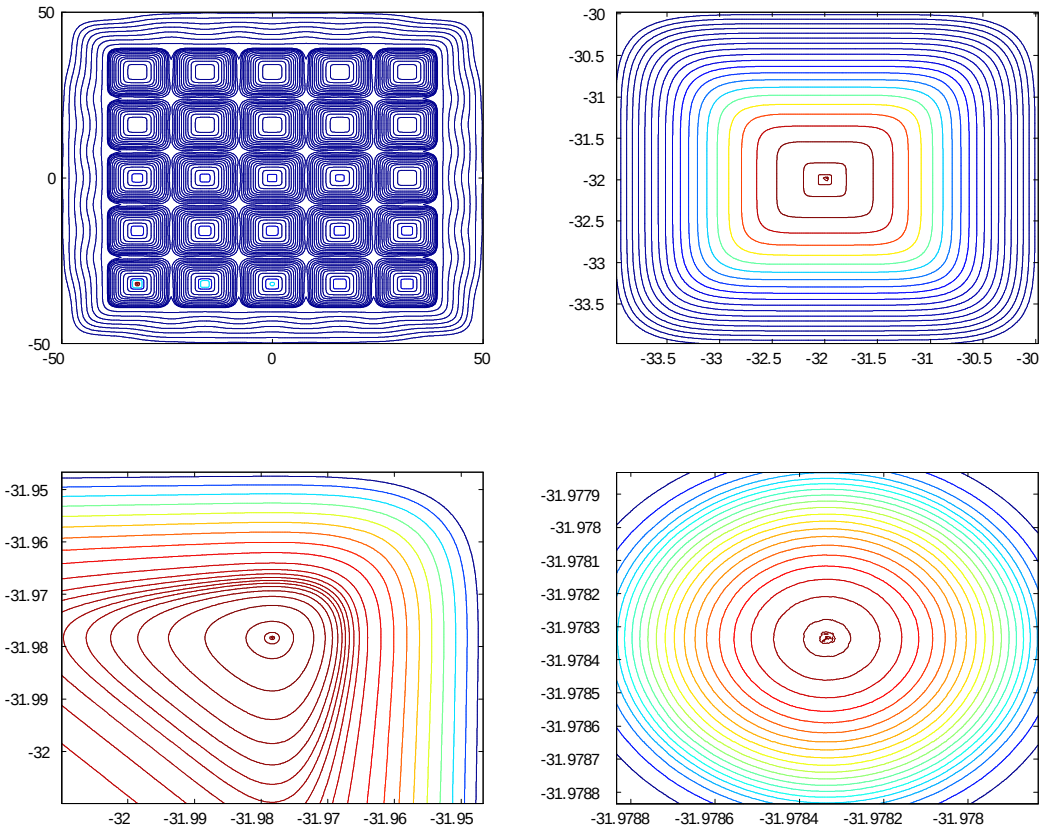

Figure 1: Contours for the function h in test problem 1

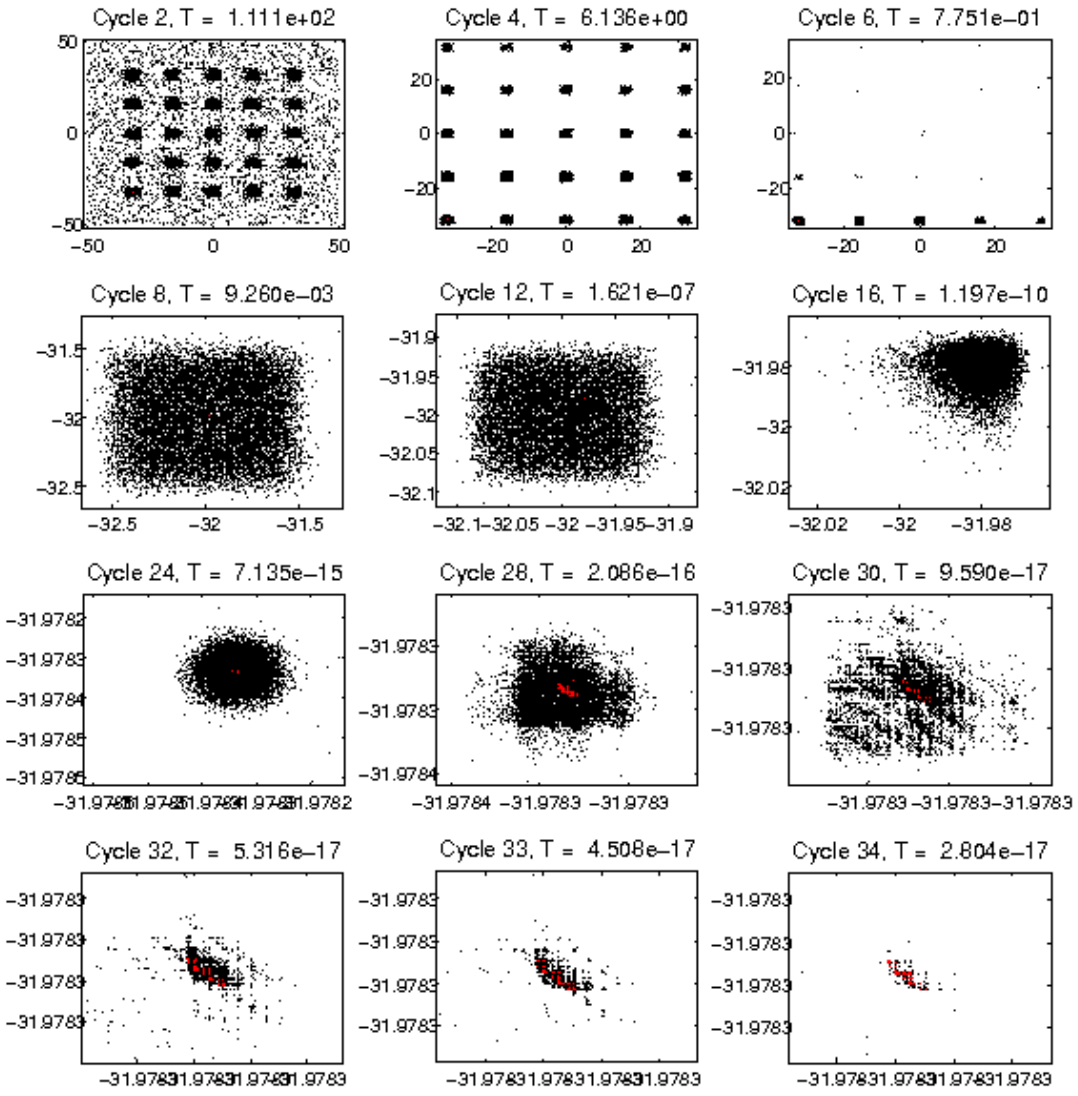

Figure 2: Distribution of particles in selected cycles in test problem 1

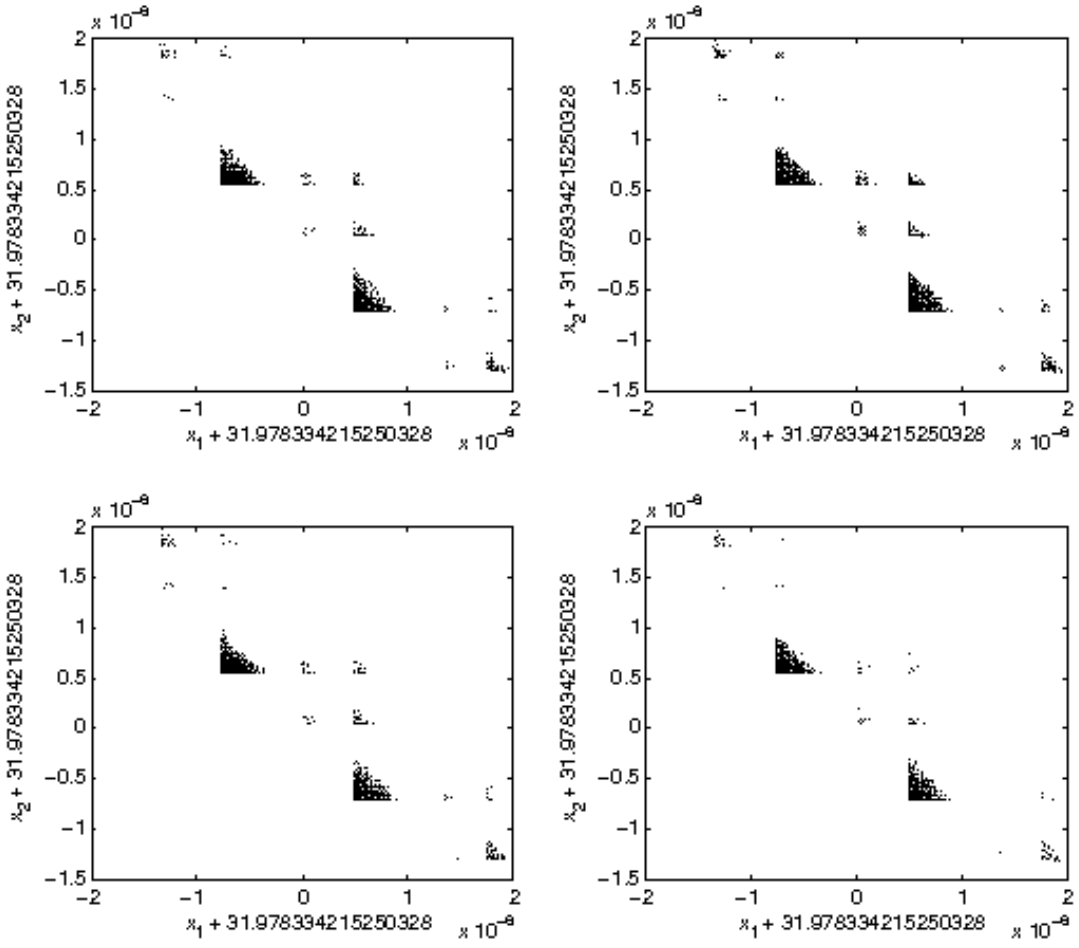

Figure 3: The set X_L^* in four executions of the algorithm in test problem 1

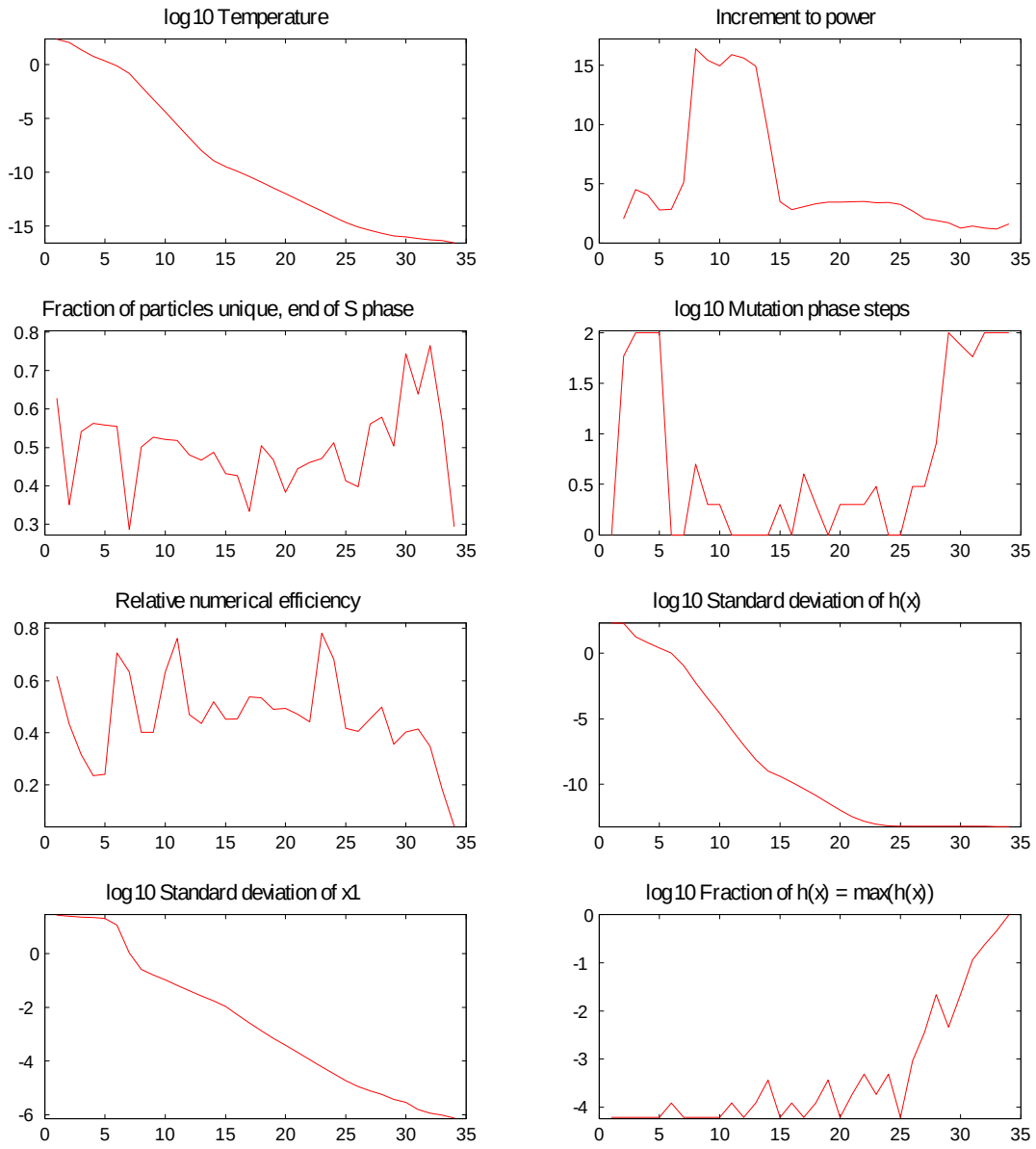

Figure 4: Some aspects of the SABL optimization algorithm (vertical axis) by cycle (horizontal axis) in test problem 1