LIBRJMC: AN OPEN-SOURCE GENERIC C++ LIBRARY FOR STOCHASTIC OPTIMIZATION

Mathieu Brédif and Olivier Tournaire

1 IGN, MATIS, 73 avenue de Paris, 94160 Saint-Mandé, France ; Université Paris-Est
2 CSTB, 84 avenue Jean Jaurès, 77447 Marne la Vallée, France
3 Imagine/LIGM, Université Paris-Est, 6 avenue Blaise Pascal, 77455 Marne la Vallée, France

Commission III/3

KEY WORDS: Stochastic Optimization, RJ-MCMC, Simulated Annealing, Generic C++ Library, Open-Source

ABSTRACT:

The librjmcmc is an open source C++ library that solves optimization problems using a stochastic framework. The library is primarily intended for but not limited to research purposes in computer vision, photogrammetry and remote sensing, as it has initially been developed in the context of extracting building footprints from digital elevation models using a marked point process of rectangles. It has been designed to be both highly modular and extensible, and have computational times comparable to a code specifically designed for a particular application, thanks to the powerful paradigms of metaprogramming and generic programming. The proposed stochastic optimization is built on the coupling of a stochastic Reversible-Jump Markov Chain Monte Carlo (RJMCMC) sampler and a simulated annealing relaxation. This framework allows, with theoretical guarantees, the optimization of an unrestricted objective function without requiring any initial solution. The modularity of our library allows the processing of any kind of input data, whether they are 1D signals (e.g. LiDAR or SAR waveforms), 2D images, 3D point clouds... The library user has just to define a few modules describing its domain specific context: the encoding of a configuration (e.g. object type in a marked point process context), reversible jump kernels (e.g. birth, death, modifications...), the optimized energies (e.g. data and regularization terms) and the probabilized search space given by the reference process. Similar to this extensibility in the application domain, concepts are clearly and orthogonally separated such that it is straightforward to customize the convergence test, the temperature schedule, or to add visitors enabling visual feedback during the optimization. The library offers dedicated modules for marked point processes, allowing the user to optimize a Maximum A Posteriori (MAP) criterion with an image data term energy on a marked point process of rectangles.

1 INTRODUCTION

Optimization of an objective function over a given search space is a very important and wide research topic. Optimization problems come however in very different flavors: the objective function may or may not exhibit nice properties like convexity or the search space may be a simple compact of $\mathbb{R}^n$, a (possibly infinite) set of discrete values, or even a combination of both: a (possibly infinite) union of spaces embeddable in $\mathbb{R}^n$. This article proposes an implementation of a stochastic optimization framework for optimizing arbitrary objective functions over the latter and more complex search spaces. Given the difficulty of these problems that mix combinatorial and variational aspects and the genericity of the proposed library, it does not provide a one-size-fits-all solution. Instead, it implements a theoretically sound stochastic framework that is easily and highly customizable for rapid prototyping and tuning of the optimization process. The librjmcmc is a generic C++ library (Abrahams and Gurtovoy, 2004) designed to be both heavily optimized and highly modular.

In this paper, we first briefly remind the mathematical foundations involved in the chosen stochastic framework (section 2). We then present our implementation choices to build a computationally efficient and generic library (section 3). Before concluding, we demonstrate sample uses of the library, including the discussion of the implementation of the building footprint extraction from satellite imagery proposed in (Tournaire et al., 2010) (section 4).

2 STOCHASTIC OPTIMIZATION

The optimization is performed using a coupling of a simulated annealing relaxation (Salamon et al., 2002) with a Reversible Jump Markov Chain Monte Carlo (RJMCMC) sampler (Hastings, 1970; Green, 1995). This framework enables the stochastic minimization of a large class of energies over complex search spaces, only requiring the evaluation of the objective function, rather than e.g. derivatives or convexity properties (Descombines, 2011).

2.1 Reversible-Jump Markov Chain Monte Carlo

A Markov Chain Monte Carlo (MCMC) sampler provides a series of samples according to a given unnormalized probability distribution function. RJMCMC is an extension of MCMC that allows the configuration space $\Omega$ to be the union of spaces of varying dimensions $\Omega = \bigcup_{n} \Omega_n$. Algorithm 1 (Green, 1995) consists in repeating stochastic proposition and acceptance steps to build a series of configurations (i.e. elements of the search space $\Omega$) which stationary distribution is the desired target distribution $\pi$. Note that after a sufficient number of iterations, the running configuration $X_t$ is independent of the initial configuration $X_0$.

A well-known property of interest is that the stationary distribution $\pi$ is not only never sampled directly (as it is the purpose of the algorithm) but also only evaluated up to a multiplicative constant, as it only appear in the term $\frac{f(x)}{\pi(x)}$ of the Green ratio. This thus enables the sampling of a probability $\pi$ defined as the normalization of a non-negative integrable function $f$: $\pi(x) = \frac{f(x)}{\int f(x)\,dx}$. 


Algorithm 1: RJMCMC sampler: Metropolis-Hastings-Green

- Initialize the configuration \( X_0 \) such that \( \pi(X_0) \neq 0 \)
- repeat
  - Sample \( i \sim q(i|X_1) \); // Select a reversible kernel \( Q_i \)
  - Sample \( X'_1 \sim Q_i(\cdot|X_1) \); // Sample the kernel \( Q_i \)
  - \( R \leftarrow \pi(X_1)Q_i(X_1|X'_1) / \pi(X'_1)Q_i(X'_1|X_1) \); // Green ratio
  - \( \alpha \leftarrow \min(1, R) \); // Acceptance rate
  - With probability \( \{ \alpha, 1 - \alpha \} \), \( X_{t+1} \leftarrow X'_t \)
- until convergence:

The kernel contribution to the Green ratio is then:

\[
Q_i(X'_1|X_1) = \frac{p_i(n'|n) \phi_i(u')}{p_i(n'|n) \phi_i(u)} \frac{\partial T_i(\theta, u)}{\partial (\theta, u)}
\]

### 2.2 Marked Point Process

A (possibly multi-)Marked Point Process (MPP) (van Lieshout, 2000) is a stochastic unordered set of geometric objects of one or many types (e.g. points, segments, circles, ellipses or rectangles in 2D, ellipsoids or spheres in 3D). The marked point denomination arises from the usual decomposition of the object parameterization into a point of its embedding space (usually its center) and a vector of values (the marks) that completes the geometric parameterization of the object (e.g. orientation, length or radius parameters), hence the name Marked Point Process.

The librjmcmc currently focuses on (multi) MPP in both sampling and optimization contexts. When direct sampling of the MPP is not possible, as when its probability distribution function (PDF) is not samplable and not normalized, the RJMCMC algorithm 1 may be used to build a Markov Chain which distribution converges to the desired target stationary distribution.
(such as the acceptance rate of each kernel) and is used through the following member function that modifies the current configuration in place (preventing useless copies), performing one iteration of algorithm 1.

```cpp
template<typename Configuration>
void sampler::operator()(Configuration & config, double temp);```

The Configuration type and its Modification associated type is highly problem dependent, section 2.2 will detail them in the MPP context. The sampler does only access them through the kernels, the reference Density and the evaluation of the energy difference $\Delta U$ they offer.

### 3.2.1 Density Strategy
This strategy provides the reference distribution, by exposing a double pdf_ratio(const Configuration&, const Modification&) member function that evaluates the distribution ratio $\pi(X_i)/\pi(X_0)$, where $X_0$ is provided as a Configuration and $X_i$ as a proposed Modification of the current configuration $X_0$. The poisson_distribution and uniform_distribution are discrete probabilities over the set of non-negative integers, which are handy as basic blocks to build a Density. Discrete distributions expose the following member functions:

- **metropolis_acceptance** $R_n e^{-\Delta U}$
- **smu_hartley_acceptance** $R_n / (1 + e^{-\Delta U})$
- **franz_hoffmann_acceptance** $R_n \left( c \left( 1 - \frac{1}{1 + q \Delta U} \right) \right)^{\frac{1}{\alpha}}$
- **tsallis_tsiariolo_acceptance** $R_n \left( c \left( 1 - \left( 1 - q \Delta U \right) \right) \right)^{\frac{1}{\alpha}}$
- **dueck_scheuer_acceptance** $\begin{cases} 1 & \text{if } R_n e^{-\Delta U} \geq R_0 \\ 0 & \text{otherwise} \end{cases}$

### 3.2.2 Acceptance Strategy
It encodes how the simulated annealing temperature $T$ is used to introduce the energy bias (through the $\Delta U$ term) into the reference PDF, $R_n$ denoting the Green ratio of the reference process. metropolis_acceptance is the most common choice, using the unnormalized target distribution $\pi e^{-\frac{U}{T}}$. Other acceptance rules depart from the formulation of a well-defined target distribution (Salamon et al., 2002).

![Available Acceptance Models](image)

### 3.2.3 Kernel Strategy
Pairs of reversible kernels are provided together using the following class, the forward and backward kernel being defined by swapping roles of Views, of Variates and inverting the Transform.

```cpp
template<typename View0, typename View1, typename Variate0, typename Variate1, typename Transform, class kernel;```

A kernel is used to propose atomic moves to explore the configuration space.

**Variate Concept** It provides the distribution $\phi_i$ that samples a fixed-length vector and evaluates its probability density.

**View Concept** It encodes the combinatorial aspects of the kernel and the function $\psi_i$ that extract an iterator over $\theta$ values when applied forwards and constructs the new configuration based on the vector $(\theta', u')$ resulting from the Transform when applied backwards.

**Transform Concept** Transformations encode the bijective differentiable mapping $T_i$ between spaces of equal dimensions. They are thus used to describe the bijection applied during a kernel proposal. Transformations must provide apply, inverse and abs_jacobian functions, which are used respectively in the forward kernel, the backward kernel and in the Green ratio computation.

### 3.3 MPP Concepts

#### 3.3.1 MPP Configurations
MPPs are commonly used to drive a stochastic exploration in an optimization context where the objective function, hereafter called an Energy is composed as the sum of per-object unary terms and per-pair of objects binary terms. For efficiency reasons, Configurations are thus tightly coupled with Energies in this library. A Configuration is a wrapper around a container of objects. The library offers vector, configuration and graph_configuration Configuration models. The former is based on a simple std::vector while the latter embeds a boost::graph that allows the caching of unary and non-zero binary energy terms, speeding-up $\Delta U$ computations.

The only requirement on the object type is that it is able to provide an iterator over a parameterisation vector. Multi-Marked Point Processes (i.e. processes of heterogeneous object types) are supported by supplying a boost::variant over the types of objects as the object template argument of the configuration. In this context, all library functions that handle explicitly the objects have them first dispatched through the boost::variant dispatching function so that functors like Energies are boost::variant-agnostic.

#### 3.3.2 MPP Energies
The library currently supports energies that are defined as sums of unary ($U_1$) and binary ($U_2$) energies:

$$U(X_i) = \sum_{x \in X_i} U_1(x) + \sum_{x,y \in X_i, x \neq y} U_2(x, y)$$

![Available Energy Models](image)

#### 3.3.3 MPP Density
The reference MPP distribution is defined using the class marked_point_process direct sampler, which combines a discrete distribution (section 3.2.1) for each object type, probabilizing the number of objects and a uniform object sampler that is used to get independent object samples.
...and its reverse kernel RJMCMC. Proposed new objects. The simplest MPP kernels that are sufficient for iterators to the death-proposed objects and a vector of birth-proposed new objects. The simplest MPP kernels that are sufficient for convergence are provided: RJMCMC::uniform_birth_kernel and its reverse kernel RJMCMC::uniform_death_kernel.

3.3.5 MPP Kernels For efficiency, the energy evolution $\Delta U$ of the modifications proposed by the kernels must be easy to compute. Thus MPP Kernels usually only delete and/or insert a small number of objects. A Modification class thus keeps a vector of iterators to the death-proposed objects and a vector of birth-proposed new objects. The simplest MPP kernels that are sufficient for convergence are provided: RJMCMC::uniform_birth_kernel and its reverse kernel RJMCMC::uniform_death_kernel.

3.4 Simulated Annealing Concepts
The main simulated annealing function is the following free function that performs the optimization loop until the convergence EndTest succeeds, advancing the temperature Schedule and calling the visitor function at each iteration:

```c++
template<class Configuration, class Sampler, class Schedule, class EndTest, class Visitor>
void simulated_annealing::optimize(
    Configuration& config, Sampler& sampler, Schedule& schedule, EndTest& end_test, Visitor& vis)
{
    double t = schedule.begin();
    for (; !end_test(config, sampler, t); t = schedule(t))
    {
        sampler(config, t);
        visitor.visit(config, sampler, t);
    }
    visitor.end(config, sampler, t);
}
```

3.4.1 Schedule concept Models of the Schedule concept are responsible for providing the evolution of the temperature throughout the simulated annealing process. The Schedule concept is a refinement of the InputIterator concept of the C++ Standard Template Library.

<table>
<thead>
<tr>
<th>Available Schedule Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>geometric_schedule $T_i = \alpha T_{i-1}$</td>
</tr>
<tr>
<td>logarithmetic_schedule $T_i = \frac{T_0}{\log(2+i)}$</td>
</tr>
<tr>
<td>inverse_linear_schedule $T_i = \frac{1}{1 + \Delta T}$</td>
</tr>
<tr>
<td>step_schedule $T^\prime &gt; T_i = T_{i+1}$</td>
</tr>
</tbody>
</table>

Whereas the logarithmetic schedule is the one that ensures convergence (Descombes, 2011), the geometric_schedule is usually used instead in practice for computational efficiency. The step_schedule wraps another schedule to offer constant temperature periods of $r$ iterations, enabling statistics computation through a visitor.

The evaluation of a suitable initial temperature is not trivial. On the one hand, too high a temperature will unnecessarily prolong the initial phase of the simulated annealing where the sampling is unbiased by the target distribution. On the other hand, too low a temperature prevents the exploration of the whole configuration space and results in the convergence to a local minimum only. (Salamon et al., 2002) suggests considering an estimation of the variance of the energy of configurations sampled according to the reference process to estimate the initial temperature. The corresponding salomon_initial_schedule $T_0$ estimation function is provided, which performs well in the absence of hardcore energy terms.

3.4.2 EndTest concept Models of the EndTest concept provide a simple predicate that informs the simulated annealing framework that the process has converged or whatever reason requiring to stop the simulated annealing iterations, such as a user-assisted cancellation. The only requirements of this concept is to provide the following member function:

```c++
template<typename Configuration, typename Sampler>
bool operator()(const Configuration& configuration, const Sampler& sampler, double temperature);
```

Provided models are max_iteration_end_test that fails after a given number of iterations and delta_energy_end_test that fails after a given number of iterations without energy change. They may be combined using the composite_end_test model.

<table>
<thead>
<tr>
<th>Available EndTest Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_iteration_end_test $t \geq t_{\text{max}}$</td>
</tr>
<tr>
<td>delta_energy_end_test $\forall i &gt; t - \tau, \Delta U_i = 0$</td>
</tr>
<tr>
<td>composite_end_test $0 &lt; \tau, \Delta U_i = 0$</td>
</tr>
</tbody>
</table>

3.4.3 Visitor concept Visitors are highly customizable objects passed to the simulated annealing process. They may be used to visualize or to gather statistics on the current state of the optimization. Available models offer console (figure 1) or GUI (based on Gilview and WxWidgets, figure 2) feedback, or shapefile saving of the current configuration. They may be combined using a composite_visitor. This extensibility mechanism offers the possibility to compile the same RJMCMC code with a GUI to build intuition and monitor the running algorithm and without for batch timing purposes.

<table>
<thead>
<tr>
<th>// called once at startup</th>
</tr>
</thead>
</table>
| template<typename Configuration, typename Sampler>
    void begin(const Configuration& configuration, const Sampler& sampler, double temperature); |
| // called once per iteration |
| template<typename Configuration, typename Sampler>
    void visit(const Configuration& configuration, const Sampler& sampler, double temperature); |
| // called once at the end |
| template<typename Configuration, typename Sampler>
    void end(const Configuration& configuration, const Sampler& sampler, double temperature); |

4 SAMPLES

4.1 Circle packing
This section details a simple example use of the librjmcmc library. The goal is to find a set of circles that minimizes a simple objective function. Each circle center lies in the unit square and its radius is constrained in an interval $[r_{\text{min}}, r_{\text{max}}]$. Notice that the cardinality of set itself is unknown. The minimized function awards a negative constant value for each circle and penalizes every pair of overlapping circles by its intersection area. Basicallly, it tries to pack as many circles as possible into the unit square while minimizing their overlap. First, one needs to define the object type of the MPP, with all its geometric methods. Then, the object container is defined as well as an objective function as the sum of a unary term for each object and a binary term for each pair of objects. The set of objects is then probabilized using a Poisson reference process and uniform birth. A basic birth and death RJMCMC sampler is then defined to enable the sampling of the MPP relative to the Poisson reference process, modulated by the score of the objective function and a temperature parameter. Then header files relative to the optimization of the objective function are included. A sufficiently slow temperature decrease morphology is defined to create a sort of bias away from the reference process to a combination of Dirac masses at the global mimima of the objective function, thus achieving its minimization. A geometric scheduling of the temperature decrease is sub-optimal but is more practical and thus more commonly used.
The main entry point starts by parsing the input parameters, providing default values. Once the reference Poisson process is instantiated, an empty configuration is created and initialized with the energy objects. Then, the RJMCMC sampler object samp is constructed. Finally, the simulated annealing objects are created and the optimization is performed in place on the configuration instance c. Figure 1 shows the output of the program, and the resulting circle packing. Its runtime is 11s for 356 circles and 10^7 iterations.

4.2 Building extraction

The method introduced in (Tournaire et al., 2010) has been implemented within the librjmcmc library, which extracts buildings...
from a Digital Surface Model (DSM) using a MPP of rectangles. This implementation, which has been included in the librjmcmc to foster reproducible research, amounts to provide the problem-dependent object: the rectangle type, the energies and the kernels. We show here results of this implementation on 4 different DSMs (figure 3), from 10 cm to 50 cm Ground Sample Distance (GSD), acquired using photogrammetry or Lidar and of various sizes. Figures 3.a and b are on the same area which is also contained in figure 3.d. The size of the dataset 3.d shows the scalability of the proposed approach, even in the absence of a proper acceleration (such as a quad tree).

<table>
<thead>
<tr>
<th>Input DSM</th>
<th>Type</th>
<th>GSD</th>
<th>Image size</th>
<th>MPP Rectangles</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.a Photo 50cm</td>
<td>650 x 650</td>
<td>76</td>
<td>177s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.b Lidar 50cm</td>
<td>650 x 650</td>
<td>109</td>
<td>179s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.c Photo 10cm</td>
<td>800 x 1400</td>
<td>31</td>
<td>104s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.d Photo 50cm</td>
<td>3634 x 2502</td>
<td>426</td>
<td>480s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Finally, the librjmcmc has now only be used within a 2D MPP context. Future interesting work would be to implement other Configuration, Kernel and Energy models, in embedding spaces of other dimensions (1D signals (Hernandez-Marin et al., 2007; Mallet et al., 2010), 3D ellipses (Perrin et al., 2006)...) or even in a non-MPP context, for instance to optimize a 3D triangulation over the noisy heightfield of photogrammetric Digital Surface Model. Our wish is that the open-source license of the librjmcmc library will enable reproducible research.

5 CONCLUSIONS AND FUTURE WORK

Generic programming is very efficient at run-time and this library showcases how modular high-level code performs well at runtime (Abrahams and Gurtovoy, 2004). Once accustomed to the necessary typedef clauses, using the library reduces to simply describing the problem at a high-level rather. Its development is however quite cumbersome, due to the extra burden allocated to the compiler, and its lack of native language support. Until concepts are accepted as part of C++, there are preliminary ways to better document them and to get more legible error reporting when the library is misused. librjmcmc now only checks the validity of the Schedule concept as an input iterator. Future work would be to systematically check template parameters for their intended concept.

Implementations of close relatives of the RJMCMC algorithm, such as jump diffusion or multiple birth and death (Descombes, 2011) would share many code parts. It would be interesting to implement them in a common library to assess their comparative strengths.

In the librjmcmc, complex binary or unary energy terms may be built using expressions on simpler energies. This technique is called a Domain-Specific Embedded Language (DSEL). It would be nice to extend its use the whole energy formulation, kernels, distributions, and even transforms so that, for instance, complex transforms would be able to generate their Jacobian computation code at compile-time. This would help hiding the complex type names produced by the generic programming and help offering a more user-friendly interface. boost::proto is a framework to define such DSELS, but its ease of use and runtime costs will have to be assessed.

References


