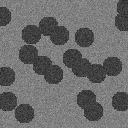
Speeding up Jump Diffusion Markov Chain Monte Carlo (JD-MCMC) algorithm in Object Detection using MPI

**Tuan Nguyen and Matthew Gaalswyk**

# Abstract

Markov Chain Monte Carlo (MCMC) is an important problem in statistical computation. In essence, MCMC uses a “random walk” to draw random samples (sampling) from an unknown distribution. Jump-Diffusion (JD) techniques provide a mixed mechanism to draw samples from disconnected state spaces. Combining the JD and MCMC algorithms (JD-MCMC) can help us estimate parameter values as well as the parameter size (order) of the unknown distributions.

A typical application of JD MCMC is object detection. Given an image with targets as shown in the figure to the right, we can detect the number of objects and the locations of each target using JD-MCMC. The sampling progress is shown in a video at <https://youtu.be/JvN2QN9T5uQ>.

However, JD-MCMC in this specific problem as well as Monte Carlo simulation in general takes a long time with a single thread to generate good results. The “random walk” should be run long enough to go through the distribution: the process also jumps back and forward many times in disconnected spaces (number of objects) to detect the correct order.

In this project, we propose a parallel implementation of the JD-MCMC algorithm in object detection, based on MPI and parallel design. The basic idea of our implementation is to simulate several random walks at the same time (parallel sampling). An existing MATLAB JD-MCMC program will be ported to C++ and run to get baseline performance results. The program will then be parallelized, utilizing MPI, and run on the High Performance Cluster. Performance results will be compared with the single-threaded C++ program as well as the MATLAB program. The anticipated challenges are correctly segmenting and efficiently distributing the workload across multiple MPI threads. Significant progress has already been made in porting the MATLAB program.

# An overview of JD-MCMC

# Problems

# Parallel MCMC algorithms

# Experimental Results

# Conclusion