requirements of achieving improvements in convergence rate. This may give a different picture of the actual efficiency of a sampling algorithm. Accelerated convergence at an extreme computational cost is obviously not optimal. Second, the computational requirements of different steps will vary greatly across platforms, depending on such factors as processor, memory architecture, use of efficient linear algebra packages, etc. Therefore, even if theoretical comparisons were extended to incorporate aspects of computation, the best block sampling scheme would remain platform-dependent. It is important to recognize that computational costs affect not only the proposal step – such as the cost of generating a multivariate normal proposal – but also model computations and density evaluations. Some parts of hierarchical models may inherently involve expensive computations, which can impact the relative efficiency of different blocking schemes. Third, existing theories and methods presume that some wise, manual selection of blocks may be feasible, based for example on an understanding of the model structure, which leads to understanding which posterior dimensions may be correlated. In general, however, it is difficult to know *a priori* which dimensions will be correlated, which is one purpose of automating a procedure like MCMC in the first place.

Here, we present a procedure for the automated exploration of MCMC blocking schemes, seeking a highly efficient MCMC algorithm specific to the hierarchical model and computing environment at hand. This represents a higher level of automated algorithm generation than is provided by existing software, which serve to produce "one size fits all" MCMC algorithms. The family of BUGS packages (WinBUGS, JAGS, and OpenBUGS; Lunn et al., 2000; Plummer, 2011; Lunn et al., 2012) assigns samplers based on local characteristics of each model parameter, using a combination of Gibbs sampling, adaptive rejection sampling, slice sampling, and, in limited cases, block sampling. Other MCMC packages including ADMB (Skaug and Fournier, 2006) and Stan (Stan Development Team, 2014) use Hamiltonian MCMC sampling (Neal, 2011), which may generally be more efficient but nevertheless represents a static approach to MCMC algorithm generation. Yet other promising methods such as Langevin sampling (Marshall and Roberts, 2012) are not incorporated into software commonly used by practitioners. For simplicity, we restrict our attention to univariate and blocked adaptive random walk sampling. However, the main concept of exploring the space of parameter blocks to improve MCMC efficiency generalizes to allow the use of other sampling methods.

In section 2, we examine the pros and cons of univariate versus block random walk sampling, both