Parameter Sensitivity Analysis of the SparTen High Performance Sparse Tensor Decomposition Software

Jeremy M. Myers  
College of William and Mary  
Sandia National Laboratories  
jermyer@sandia.gov

Daniel M. Dunlavy  
Sandia National Laboratories  
dmdunla@sandia.gov

Keita Teranishi  
Sandia National Laboratories  
knteran@sandia.gov

D.S. Hollman  
Sandia National Laboratories  
dshollm@sandia.gov

Abstract—Tensor decomposition models play an increasingly important role in modern data science applications. One problem of particular interest is fitting a low-rank Canonical Polyadic (CP) tensor decomposition model when the tensor has sparse structure and the tensor elements are nonnegative count data. SparTen is a high-performance C++ library which computes a low-rank decomposition using different solvers: a first-order quasi-Newton or a second-order damped Newton method, along with the appropriate choice of runtime parameters. Since default parameters in SparTen are tuned to experimental results in prior published work on a single real-world dataset conducted using MATLAB implementations of these methods, it remains unclear if the parameter defaults in SparTen are appropriate for general tensor data. Furthermore, it is unknown how sensitive algorithm convergence is to changes in the input parameter values. This report addresses these unresolved issues with large-scale experimentation on three benchmark tensor data sets. Experiments were conducted on several different CPU architectures and replicated with many initial states to establish generalized profiles of algorithm convergence behavior.

Index Terms—tensor decomposition, Poisson factorization, Kokkos, Newton optimization

I. INTRODUCTION

The Canonical Polyadic (CP) tensor decomposition model has garnered attention as a tool for extracting useful information from high dimensional data across a wide range of applications [1]–[5]. Recently, Hansen et al. developed two highly-parallelizable Newton-based methods for low-rank tensor factorizations on Poisson count data in [6], one a first-order quasi-Newton method (PQNR) and another a second-order damped Newton method (PDNR). They were first implemented in MATLAB Tensor Toolbox [7] as the function cp_apr, referring to this approach as computing a CP decomposition using Alternating Poisson Regression (i.e., CP-APR). These methods fit a reduced-rank CP model to count data, assuming a Poisson error distribution. PDNR and PQNR are implemented in SparTen, a high-performance C++ library of CP-APR solvers for sparse tensors. SparTen improves on the MATLAB implementation to provide efficient execution for large, sparse tensor decompositions, exploiting the Kokkos hardware abstraction library [8] to harness parallelism on diverse HPC platforms, including x86-multicore, ARM, and GPU computer architectures.

SparTen contains many algorithmic parameters for controlling the optimization subroutines comprising PDNR and PQNR. To date, only anecdotal evidence exists for how best to tune the algorithms. Parameter defaults in SparTen were chosen according to previous results using the MATLAB implementations described by Hansen et al. [6]. However, their analysis was limited to a single real-world dataset, and thus may not be optimal for computing decompositions of more general tensor data. Furthermore, it is unknown how the initial guess to a solution affects convergence, since SparTen methods may converge slowly—or worse, stagnate—on real data if the initial state is far from a solution. And, lastly, the average impact of input parameters on algorithm convergence is unclear.

To address these unknowns, we present the results of numerical experiments to assess the sensitivity of software parameters on algorithm convergence for a range of values with benchmark tensor problems. Every experiment was replicated with 30 randomly chosen initial guesses on three diverse computer architectures to aid statistical interpretation. With our results, we (1) provide new results that offer a realistic picture of algorithm convergence under reasonable resource constraints, (2) establish practical bounds on parameters such that, if set at or beyond these values, convergence is unlikely, and (3) identify areas of performance degradation and convergence toward qualitatively different results owing to parameter sensitivities.

We limited our study to multicore CPU architectures only, using OpenMP [9] to manage the parallel computations across threads/cores. Although SparTen, through Kokkos, can leverage other execution backends—e.g., NVIDIA’s CUDA framework for GPU computation—we focus solely on diversity in CPU architectures in this work.

This paper is structured as follows. Section II summarizes basic tensor notation and details. Section III describes the hardware environment, test data, and experimental design of the sensitivity analysis. Section IV provides detailed results of the sensitivity analyses. Section V offers concluding remarks and lays out future work.

II. BACKGROUND

We briefly describe below the problem we are addressing in this report; for a detailed description of CP decomposition
algorithms implemented in SparTen, refer to the descriptions in Hansen et al. [6].

The \textit{R}-component Canonical Polyadic (CP) decomposition is given as follows:

\[ \mathbf{X} \approx \mathbf{M} = [\lambda; \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}] = \sum_{r=1}^{R} \lambda_r \mathbf{a}_r^{(1)} \circ \cdots \circ \mathbf{a}_r^{(N)}, \quad (1) \]

where \( \lambda = [\lambda_1, \ldots, \lambda_R] \) is a scaling vector, \( \mathbf{a}_r^{(n)} \) represents the \( r \)-th column of the factor matrix \( \mathbf{A}^{(n)} \) of size \( I_n \times R \), and \( \circ \) is the vector outer product. We refer to the operator \( [\lambda; \cdot] \) as a Kruskal operator, and the tensor \( \mathbf{M} \), with its specific multilinear model form, as a Kruskal tensor in (1). See [1] for more details regarding these definitions.

SparTen addresses the special case when the elements of \( \mathbf{X} \) are nonnegative counts. Assuming the entries in \( \mathbf{X} \) follow a Poisson distribution with multilinear parameters, the low-rank CP decomposition in (1) can be computed using the CP-APR methods, PDNR and PQNR, introduced by Hansen et al. [6].

III. METHODS

In this section, we describe the hardware platforms, data, and SparTen algorithm parameters used in our experiments.

A. Hardware Platforms

We used diverse computer architectures to perform our experiments, with hardware and compiler specifications detailed in Table I. Intel 1–4 are production clusters with hundreds to thousands of nodes, whereas ARM and IBM clusters are advanced architecture research testbeds with tens of nodes each. We employed the maximum number of OpenMP threads available per node from each platform to maximize throughput and configured the maximum wall-clock limit as 12:00 hours for all experiments. All parallelism was solely across threads on a single node. The GNU compiler, \texttt{gcc}, was used, with \texttt{-O3} optimization and Kokkos architecture-specific flags enabled.

<table>
<thead>
<tr>
<th>Arch</th>
<th>Processor</th>
<th>Threads</th>
<th>RAM (GB)</th>
<th>GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>ThunderX2</td>
<td>256</td>
<td>255</td>
<td>7.2.0</td>
</tr>
<tr>
<td>IBM</td>
<td>Newell Power9</td>
<td>80</td>
<td>319</td>
<td>7.2.0</td>
</tr>
<tr>
<td>Intel 1</td>
<td>Sandy Bridge</td>
<td>16</td>
<td>64</td>
<td>8.2.1</td>
</tr>
<tr>
<td>Intel 2</td>
<td>Broadwell</td>
<td>72</td>
<td>128</td>
<td>8.2.1</td>
</tr>
<tr>
<td>Intel 3</td>
<td>Sandy Bridge</td>
<td>16</td>
<td>64</td>
<td>8.2.1</td>
</tr>
<tr>
<td>Intel 4</td>
<td>Sandy Bridge</td>
<td>32</td>
<td>64</td>
<td>8.2.1</td>
</tr>
</tbody>
</table>

B. Data

We conducted experiments using sparse tensors of count data from the FROSTT collection [10]. Specifically, we chose the three datasets listed in Table II to account for size, dimensionality, and density (i.e., the ratio of nonzero entries to the total number of elements in the tensor). Throughout the discussion below, we refer to the data using the short names listed in the table.

C. Software Parameter Definitions & Experimental Ranges

PDNR and PQNR are composed of standard techniques in the numerical optimization literature. Specifically, for each tensor mode, the Newton optimization computes the gradient and Hessian matrix. Then, the inverse Hessian is approximated to compute a search direction and an Armijo backtracking line search is used to compute the Newton step. The inverse Hessian is approximated for each parameter in the table denote the Intel platform used, with \texttt{-O3} optimization and Kokkos architecture-specific flags enabled.

D. Experiments

An individual experiment is a job \( j \) on platform \( m \) solving a PDNR/PQNR row subproblem for dataset \( d \) with SparTen solver \( s \), parameter \( p \), parameter value \( v \), and random initialization \( r \); all remaining software parameters are fixed at the default values listed in Table III. Certain experiments denoted with an asterisk \(^*\) were run only on Intel hardware due to limited resources associated with the other architectures; this accounts for the larger number of experiments reported for these platforms. We conducted tests on these values to provide better resolution of the impact of the parameter where nearby values—i.e., on the bounds of the test range—contained uncertainty in the results. Furthermore, we split up the experiments across the Intel platforms by parameter, running the full set of experiments across all parameter values and all random initializations on a single platform. The superscripts denoted for each parameter in the table denote the Intel platform number specified in Table I. Since we report only the number of function evaluations and outer iterations in our results, we expect that running our experiments in this way has produced valid results.
presenting statistical trends in the results. means and 95% confidence intervals (as defined in [11]) when
experiments consists of 30 replicates (i.e., 30 random initial-
ewline
f

Convergence (i.e., the number of evaluations of the negative log
likelihood objective function). As each of our
experiments were replicated using 30 random initializations.

In all experiments, we fit a 5-component CP decomposition
using a tolerance of $10^{-4}$ (i.e., the value of $\tau$ in Equation
(20) in [6], the violation of the Karush-Kuhn-Tucker (KKT)
conditions, used as the stopping criterion for the methods
we explore here). Computation of a CP decomposition using
PDNR or PQNR in SparTen requires an initial guess of the
model parameters—i.e., initial values for $M$ in (1)—drawn
from a uniform distribution in the range $[0, 1]$. As such, all
experiments were replicated using 30 random initializations.

We report results on the amount of computation required for
convergence (i.e., the number of evaluations of the negative log
likelihood objective function). $f(M)$, defined in Equation (4)
of [6]) and the quality of the solution (i.e., the value of the
negative log likelihood objective function). As each of our
experiments consists of 30 replicates (i.e., 30 random initial-
izations) across three CPU architectures, we report sample
means and 95% confidence intervals (as defined in [11]) when
presenting statistical trends in the results.

IV. RESULTS

In this section we analyze the results of the parameter sensi-
tivity experiments and describe the statistical relationships
between the convergence properties of the PDNR and PQNR
methods and their input parameters.

In total, 21,960 unique experiments were planned, account-

ing for running PDNR and PQNR with random initializations

across all parameter value ranges on the various hardware
architectures described in Sections III-A and III-C. An
experiment converged if the final KKT violation is less than the
value of $\tau = 10^{-4}$; an experiment reached maximum iterations
if the number of outer iterations exceeded the maximum
limit (i.e., max_outer_iterations) and did not converge; an
experiment was canceled if it exceeded the wall-clock limit
(i.e., SparTen neither converged to a solution nor reached
maximum number of outer iterations within 12 hours); and
an experiment was missing if it did not run due to a failure
of the system to launch the experiment or other system issue.

Of the planned experiments, we collected data from 16,139
experiments.

Table IV presents the number of experiments planned
(plan.) as defined above and the number of planned experi-
ments where data was collected (i.e., planned minus missing).
For those collected (coll.), the table shows the percentage of
experiments that were canceled (canc.), converged (conv.), or
exceeded the maximum iterations (max. iter.). We note that
the most complete set of experiment results were obtained
on the Intel platforms. Although there are many missing
experiment results (miss.) for the IBM and ARM platforms,
we attempt to identify patterns in the data we collected if

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td>PDNR</td>
<td>1110</td>
<td>1110</td>
<td>4.8%</td>
<td>82.2%</td>
<td>13.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td></td>
<td>ARM</td>
<td>1110</td>
<td>1110</td>
<td>10.5%</td>
<td>76.5%</td>
<td>13.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td></td>
<td>PQNR</td>
<td>990</td>
<td>237</td>
<td>0.0%</td>
<td>0.0%</td>
<td>100.0%</td>
<td>76.1%</td>
</tr>
<tr>
<td>IBM</td>
<td>PDNR</td>
<td>1110</td>
<td>692</td>
<td>11.3%</td>
<td>73.3%</td>
<td>15.4%</td>
<td>37.7%</td>
</tr>
<tr>
<td></td>
<td>ARM</td>
<td>1110</td>
<td>424</td>
<td>51.2%</td>
<td>12.0%</td>
<td>36.8%</td>
<td>61.8%</td>
</tr>
<tr>
<td></td>
<td>PQNR</td>
<td>990</td>
<td>293</td>
<td>61.8%</td>
<td>0.0%</td>
<td>38.2%</td>
<td>70.4%</td>
</tr>
<tr>
<td>Inte</td>
<td>PDNR</td>
<td>1680</td>
<td>1673</td>
<td>5.0%</td>
<td>86.4%</td>
<td>8.6%</td>
<td>0.4%</td>
</tr>
<tr>
<td></td>
<td>ARM</td>
<td>1680</td>
<td>1663</td>
<td>11.0%</td>
<td>80.6%</td>
<td>8.4%</td>
<td>1.0%</td>
</tr>
<tr>
<td></td>
<td>PQNR</td>
<td>1440</td>
<td>1434</td>
<td>12.1%</td>
<td>78.6%</td>
<td>9.3%</td>
<td>0.4%</td>
</tr>
</tbody>
</table>

### TABLE III

SparTen software parameter descriptions and values used in our experiments.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Values Used in Experiments</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_outer_iterations</td>
<td>Maximum number of outer iterations</td>
<td>10000</td>
</tr>
<tr>
<td>max_inner_iterations</td>
<td>Maximum number of inner iterations</td>
<td>20</td>
</tr>
<tr>
<td>max_backtrack_steps</td>
<td>Maximum number of backtracking steps in line search</td>
<td>10, 2, 4, 8, 16, 32, 64, 128, 256, 512</td>
</tr>
<tr>
<td>min_step_size</td>
<td>Tolerance for nonzero line search step length</td>
<td>$10^{-7}$, $10^{-8}$, $10^{-9}$, $10^{-10}$, $10^{-11}$</td>
</tr>
<tr>
<td>step_reduction_factor</td>
<td>Factor to reduce line search step size</td>
<td>0.5, 0.1, 0.3, 0.5, 0.7, 0.9</td>
</tr>
<tr>
<td>mu_initial</td>
<td>Initial value of damping parameter</td>
<td>$3^{3}$</td>
</tr>
<tr>
<td>damping_increase_factor</td>
<td>Scalar value to increase damping parameter in next iterate</td>
<td>$10^{-3}$, $10^{-4}$, $10^{-5}$, $10^{-6}$, $10^{-7}$, $10^{-8}$</td>
</tr>
<tr>
<td>damping_decrease_tol</td>
<td>Tolerance to decrease the damping parameter</td>
<td>0.75, 0.505, 0.75, 0.9</td>
</tr>
<tr>
<td>eps_div_zero_grad</td>
<td>Tolerance defining active (nonzero) variables</td>
<td>$10^{-4}$, $10^{-5}$, $10^{-6}$, $10^{-7}$, $10^{-8}$</td>
</tr>
<tr>
<td>size_LBFGS</td>
<td>Number of limited-BFGS update pairs</td>
<td>3, 1, 2, 3, 4, 5, 10, 15, 20</td>
</tr>
<tr>
<td>log_zero_safeguard</td>
<td>Tolerance defining active (nonzero) variables</td>
<td>$10^{-3}$, $10^{-4}$, $10^{-5}$, $10^{-6}$, $10^{-7}$, $10^{-8}$</td>
</tr>
<tr>
<td>eps_zero_safeguard</td>
<td>Tolerance to avoid computing log(0)</td>
<td>$10^{-3}$, $10^{-4}$, $10^{-5}$, $10^{-6}$, $10^{-7}$, $10^{-8}$</td>
</tr>
<tr>
<td>eps_active_set</td>
<td>Tolerance defining active (nonzero) variables</td>
<td>$10^{-3}$, $10^{-4}$, $10^{-5}$, $10^{-6}$, $10^{-7}$, $10^{-8}$</td>
</tr>
</tbody>
</table>

1PDNR-specific; 2PQNR-specific; 3Intel platform used for experiments; *values evaluated on Intel platform only.

### TABLE IV

Experiments run on the different datasets and hardware platforms.
there is strong evidence to support our claims. We note that a few parameters (eps_active_set, min_step_size, suff_decrease_tol, damping_increase_tol, damping_decrease_tol) showed no statistically significant differences across the range of input values used in the experiments. We conjecture that we did not find values where the parameters display sensitivities in the chosen tensor problems, thus it remains unclear if this behavior holds in general.

A. General Convergence Results on Real-World Data

As discussed in Section I, applying PDNR and PQNR to real-world data has been explored previously in the literature only for a single problem. From Table IV, we observed that PQNR is canceled more than PDNR in the allotted time across datasets and CPU platforms. This confirms our intuition, since it is a classical result in iterative methods that damped Newton methods converge quadratically, in comparison to quasi-Newton methods, which converge superlinearly. Specifically, PQNR calls the objective function 2.7 times more than PDNR on average on the chico data and fails to converge for any experiment on lbnl data across all hardware platforms. By contrast, PDNR converges in 86% of lbnl experiments across platforms when only 32 outer iterations are allowed.

B. Sensitivity of Convergence and Solution Behavior

There are certain ranges of parameter values that lead to good or bad convergence behaviors in general. This is illustrated in Figures 1–3, where parameter values and random initializations are depicted across the horizontal and vertical axes, respectively. These heatmaps display total objective function evaluations, where solid columns of a single shade indicate the same convergence behavior across all 30 random initializations. Green shades are consistent with converged experiments. Vertical bands not shaded green identify values that may impact algorithm performance, due either to iteration constraints (blue hues) or excessive computations corresponding to slow convergence or stagnation (red hues). Hatches denote non-converging exit status. Grey represents missing data, i.e., experiments that were planned but never conducted due to resource limitations—e.g., dequeued by the cluster administrator—or a system failure. Nearly solid column lines of the same shade indicate similar behavior, but also that there is some sensitivity of those parameter values to the initial starting point of the iterative methods.

In several cases, there is a tendency to time-out at one or both bounds of the test ranges for both solvers. The behavior of numerical stability parameters eps_div_zero_grad and log_zero_safeguard was consistent across combinations of solver, data, and CPU hardware. When eps_div_zero_grad is large, gradient directions that do not lead to objective function improvements may be scaled the same as gradient directions that do lead to such improvements. Furthermore, the corresponding eigenvalues of the Hessian matrix are amplified and Hessian information may be lost when determining the next iterate. For example, PDNR loses Hessian information as eps_div_zero_grad increases on chico data; PDNR rarely converges and PQNR never converges when this parameter is relatively large—i.e., $10^{-5}$. Moreover, both algorithms are sensitive to the parameter’s lower bound, as small values may be insufficient to avoid an ill-conditioned Hessian matrix. In either case, additional iterations follow to correct errors incurred by eps_div_zero_grad values, large and small.

PDNR typically does not converge for large log_zero_safeguard values on large tensor problems. This parameter sets a nonzero offset in logarithm calculations to avoid explicitly computing log(0). High precision in logarithm computations tends to ensure the objective function is minimized accurately. When the value is too large, the calculated logarithm may be too small, and more backtracking steps are required to sufficiently decrease the objective function in the line search routine, making time-outs more likely. On the other hand, the effect of the parameter on convergence is indistinguishable for values smaller than $10^{-8}$ across all experiments.

This effect—when convergence behavior is similar for values set within sensitivity constraints—is common among several algorithm parameters corresponding to the different numerical optimization subroutines that comprise PDNR and PQNR. Two examples are damping_increase_factor and damping_decrease_factor, which control updates to the PDNR Hessian matrix damping parameter $\mu$. SparTen rarely converges when the former is set too low (1.5); the likely effect is that the updated damping factor is insufficient to guarantee a well-conditioned Hessian and too many unimportant directions are considered when computing the search direction. Above the 1.5 bound, the cost in objective function calls does not change significantly. The PQNR-specific parameter, size_LBFGS, behaves similarly; the only observable difference occurs when the update size is 1, using only the current iterate in the BFGS update.

Other parameters show meaningful differences in cost, defined in terms of the number of function evaluations required before convergence is achieved, when varied. Hansen et al. predict in [6] that when the damping parameter $\mu$ is set too large, a loss of Hessian information follows, which impacts convergence. For example, when $\mu_{\text{initial}}$ is large, the computational cost grows dramatically and time-outs become more likely, since the initial step length will at first be very small in every outer iteration and useful Hessian information is discarded in early stages of the inner loop solves. Convergence is most likely for a large, but not too-large, value, i.e., $\mu_{\text{initial}} = 10^{-3}$. Cost grows 177.2% on lbnl and nearly doubles (+92.2%) on chico as $\mu_{\text{initial}}$ grows from $10^{-3}$ to $10^{-2}$. It is important to note in the former case that this cost is skewed by one experiment that converged after nearly 42,000 outer iterations, in comparison to 1,300 for the other parameter values on average, illustrated in Figure 4, where the x-axis is truncated to highlight the differences in total cost. Smaller values (i.e, $10^{-8}$) seem to perform better for chico, the smallest, densest problem and larger values (i.e., $10^{-5}$) tend to perform better for large, sparse problems.

Allowing many backtracking steps during the line search may cause PDNR to waste effort; however, PQNR appears to
perform better, in general, with more steps. PDNR is sensitive to the number of backtracking steps on *chicago*: average work performed is less when the maximum number of allowed steps is large and more work is performed when the number of steps is small. On *lbml*—the sparsest tensor problem considered—PDNR performs better with fewer backtracking steps (see Figure 5). The average cost incurred by PQNR decreases as \texttt{max\_backtrack\_steps} increases.

The line search parameter \texttt{step\_reduction\_factor} is used to reduce the line search step size between iterations. On a large, sparse tensor problem, increasing this parameter may accelerate convergence. On the other hand, a small value makes convergence less certain. Figure 6 illustrates this behavior on the *lbml* data: the average total cost decreased by 77\% as \texttt{step\_reduction\_factor} increased from 0.1 to 0.5 (SparTen default) and decreased another 28\% from 0.5 to 0.9. On *nell* data, PQNR only converged for large values (0.7, 0.9).

Parameter sensitivities affect not only convergence behavior, but may also produce qualitatively different results. Figure 7 illustrates the effect where large \texttt{eps\_div\_zero\_grad}—and consequently, small step length—minimizes calls to the objective function and results in minimal objective function.
value (higher values on the y-axis correspond to a smaller negative LogLikelihood). Most striking is that larger eps_div_zero_grad decreases the objective function more than an order of magnitude. This result was collected from 79 of 90 planned PDNR experiments on lbnl, and thus we consider this interesting effect worthy of further investigation.

V. CONCLUSIONS

Using results from more than 16,000 numerical experiments on several hardware platforms, we presented experimental results that expand our understanding of average PDNR and PQNR convergence on real-world tensor problems. We have shown that when using PQNR to compute large tensor decompositions convergence is less-likely under reasonable resource constraints. We have shown that some software parameters are sensitive to bounds on values. Further, we showed that varying several parameters can dramatically impact algorithm performance, and in some cases, may produce qualitatively different results.

Future work may address the issue of stagnation in Newton optimization methods for CP decompositions. We showed examples where the solver converged to a solution slowly but within the allotted time of 12 hours. For those experiments that timed out, it is unknown whether SparTen would eventually converge to a solution or stagnate without making progress. We anticipate that stagnation could be determined if the objective function values converge to a statistical steady state without satisfying the convergence criterion. Future development of SparTen may include dynamic updates to algorithm parameters based on local convergence information. Lastly, future experiments could explore coupled sensitivities among algorithm parameters, as this work was limited to single parameter, univariate analyses.

ACKNOWLEDGEMENT

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA0003525.
REFERENCES