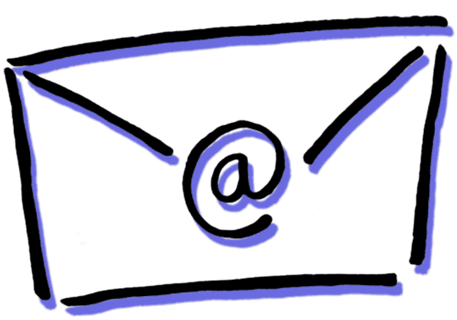
SIAG/OPT Views and News

A Forum for the SIAM Activity Group on Optimization

Volume 29 Number 1 December 2021

Contents

Are you receiving this by postal mail?
Do you prefer electronic delivery?



Email siagoptnews@lists.mcs.anl.gov

Articles

Targeting Exascale with Julia on GPUs for multiperiod optimization with scenario constraints

Mihai Anitescu, Kibaek Kim, Youngdae Kim, Adrian Maldonado, François Pacaud, Vishwas Rao, Michel Schanen, Sungho Shin, Anirudh Subramanyam[1]

1 Introduction

We present a case study of how, confronted with the problem of rewriting an optimization-centered exascale application from scratch because of technological constraints, we were able to do so in less than 18 months relying on features provided by the Julia language and to run it at the largest scale available on exascale-type architectures. The technological constraints stemmed from the fact that all exascale architectures will be GPU based whereas most existing high-performance large-scale optimization tended to rely on specialized sparse saddle point solvers for performance. As is well known, sparse linear algebra on GPUs is difficult; but, importantly, what we would have needed to address this challenge on the time scale of the project was unlikely to become available. The fact that Julia provides just-in-time compilation and has exceptional compactness and that its philosophy embeds transformable programming allowed us to consider rewriting the application from scratch, under some assumptions of support of the AMD and Intel GPU architectures by the Julia community that have so far panned out. Employing our experience in decomposition for large-scale optimization and automatic differentiation and targeting multiple exascale architectures, we were able to execute problems at scale on both very large-scale AMD- and NVIDIA-based architectures, and we fully expect that before a year we will succeed in doing so with the Intel architecture. Moreover, through the KernelAbstractions.jl package in Julia we see a way to do so—and have partially implemented it—without a line of architecture-dependent code in the application. We believe that this is an indication that advances in compiler technologies may be at a stage where supporting radically different architectures without compromising performance and with manageable (if not even comfortable) development effort may

[1] All authors' affiliation and photos are at the end of the article.

be within sight. We share with the optimization community our experience of this journey so far.

2 Context

We present the circumstances that led us to decide to rewrite an entire application in Julia.

2.1 Scenario-Driven Optimization Problems on CPU

In the past decade the Argonne optimization group, along with many other researchers, has been interested in solving scenario-driven optimization problems (such as two-stage stochastic optimization problems [21, 27, 16, 15, 14]) on very large parallel computers. A big demand in this area has stemmed from an increased focus on including uncertainty in decisions about energy systems, particularly for stochastic dispatch to accommodate an increased penetration of renewable resources on the grid. Even without uncertainty, a flagship problem in this class is security-constrained alternating current optimal power flow (OPF), which in many market areas in the United States is the standard for computing the locational marginal prices at which the market is cleared. In that nonlinear, nonconvex optimization problem the scenarios are brought about by a contingency: losing one of the network assets (e.g., [3, 30]). This allows the market formulation to explicitly account for reliability. It is of interest to solve such problems and versions of them that consider multiple contingencies, incorporation of uncertainty, and multiperiod formulations for responsive markets or control.

In our endeavors we leveraged the fact that decomposition algorithms have been successful for solving large-scale optimization problems. The key idea of the algorithms is to exploit the special structures that are embedded in a large optimization problem, which can lead to the decomposition of the problem into a number of smaller problems. While this can be accomplished in many ways, we have pursued mainly two approaches: interior-point methods with decomposition at the linear algebra level and Lagrangian approaches where the optimization problem itself was decomposed.

In the first category, the parallel optimization solver PIPS[2] exploits a block-angular structure of the Karush–Kuhn–Tucker (KKT) system when solving nonlinear optimization problems by using interior-point methods. The block-angular KKT system can then be solved in parallel by using the Schur complement, which has been successfully applied for solving optimal power flow problems on high-performance computing systems with many CPUs (e.g., [21, 27]). In the design of the approach, for best performance to productivity ratio it is essential to be able to solve efficiently one scenario on one compute node and thus fit one scenario in the node memory. Moreover, classical interior-point approaches for nonconvex optimization require access to linear algebra capable of determining the inertia of the KKT matrix efficiently. On CPU-based architectures, however, both these features were easily available, and this was for several years our preferred approach when targeting the largest problems we could solve.

In the second category, Lagrangian-based decomposition algorithms have been developed and applied (e.g., dual decomposition [16, 15, 14, 13], alternating direction method of multipliers or ADMM [5, 33], progressive hedging [35, 29, 10]) for solving large-scale optimization problems on high-performance computing (HPC) systems. Such algorithms have far more flexibility compared with linear-algebra-based decomposition since they can allow the computational unit to be as small as one desires. On the other hand, they are first-order methods, and although deriving excellent computational performance per iteration they cannot compete overall with the first category of algorithms. This is also partially behind the success of interior-point algorithms in particular when targeting a very high level of accuracy.

Another thread in our work has been our increasing reliance on Julia, which we have worked with since 2014. For instance, the Argonne team extended JuMP—an optimization algebraic modeling package in Julia—for stochastic programming with the package StochJuMP, which can create a stochastic programming problem instance in parallel [12]. Later, the package was further generalized for any structured optimization and interfaced with the parallel optimization solvers PIPS and DSP.[3] In addition, we were able to quickly prototype new optimization algorithms by leveraging the modeling convenience for several application-specific models such as the attacker-defender problem [4] and Lagrangian-based network decomposition [36] in power systems. We also made publicly available more experimental prototypes of generic optimization solvers such as the simplex method, sequential quadratic programming method, reduced-optimization method, and alternating direction method of multipliers.[4]

Another consideration is the choice or development of the modeling environment. It provides multiple features, but an important one in our context consists of derivative evaluations. In optimization, modeling environments for nonlinear programming include AMPL [8], JuMP [6], and Pyomo [11]. These environments, however, are generally not built with extreme scalability in mind, which was one consideration behind the development of StochJuMP [12]. None of these environments supports automatic differentiation (AD) on GPUs.

2.2 The exascale project and the GPU pressure

In the late 2000s to early 2010s the United States embarked on an initiative to bring about exascale computing. While the effort had many drivers, an important one was (and still is) the concern about a possible slowdown in Moore's law, which had driven much of the increase in computational capability. This concern is tightly related to the rapid increase of power consumption for the top supercomputers. In 2006 the Blue Gene/L led the Top500 list with 1,433 kW power consumption for 280.6 TFlop/s. Comparing this with today's fastest computer with a power consumption of 29,899

[2] https://github.com/Argonne-National-Laboratory/PIPS

[3] https://github.com/Argonne-National-Laboratory/DSP

[4] https://exanauts.github.io/

kW for 442,010 TFlop/s illustrates that a factor of 30 of the 2000 speedup observed in LINPACK can be attributed directly to power. Moreover, the absolute power consumption estimates for top supercomputers were getting outside the reasonable ranges for *any* facility.

In anticipation of this power wall, the U.S. Department of Energy created the Exascale Computing Project (ECP),[5] which we joined in 2016 as the part of the ExaSGD project.[6] ExaSGD aims to solve scenario-driven optimization problems and some of their extensions, such as multiperiod optimization, at exascale. While the nature of the target exascale architectures was unclear as it was evolving, in about early 2019 it became apparent that the exascale architectures would all be GPU based. Moreover, our project timelines required a reasonable expectation of demonstrating a solution at scale by September 2021.

GPUs are accelerators with special parallel threading cores akin to vector architectures of the past and follow the single instruction, multiple data (SIMD) paradigm according to the Flynn taxonomy [7]. For example, NVIDIA classifies its architecture as streamed multiprocessing single instruction, multiple threads.[7] The streaming hints at a memory hierarchy that enables high bandwidth through a wide bus instead of a deep pipeline. The threads are then executed in lockstep with one instruction and fetching their own data. This approach adds flexibility to this elaborate system that lets anything achieve tremendous acceleration provided that it fits into this blueprint of massively parallel but lockstepped threads.

With their origin in mass-appeal graphics processing for games, GPUs have by now increased their numerical precision and added operations for computational science, such as the tensor cores to cater to the data center market with a focus on machine learning. Contending with them as mathematicians and computer scientists also allows us to have a glimpse of and prepare ourselves for the hardware of the future. If the ECP pathway is an indication, what will drive future hardware will be the appeal to the mass consumer market, which is still currently unknown. This requires a flexible software infrastructure that can leverage a variety of accelerators and vendors as they become available.

2.3 Evaluating the way forward

The distinguishing feature of streaming calculations on GPUs is that computations are very fast but data movement is (relatively) slow. Consequently, certain operations that are essential for efficiently factorizing a sparse matrix, such as reordering, are exceedingly inefficient to carry out on GPUs. To investigate the linear algebra situation, our ECP project team did a thorough profiling of sparse linear solvers for KKT matrices on GPUs led by Kasia Świrydowicz [34]. The best solver outcomes on both architectures are summarized in Table 1. Here the GPU used for profiling is an NVIDIA V100, the current backbone of the second-ranking TOP500 supercomputer Summit.

	MA57 (s)	STRUMPACK (s)
ACTIVSg200	0.2	1.6
ACTIVSg10k	0.6	3.7
ACTIVSg70k	6	27

Table 1: Summary of results presented in [34]. Fastest CPU solver MA57 compared to fastest GPU solver STRUMPACK.

At that time, MA57 on CPUs seemed to run faster than what we had available on GPUs, and the situation looked at least as dire in the fall of 2019 when we were doing technical planning. This seemed to indicate that a linear algebra decomposition approach, which was our predominant go-to method for scalability (Section 2.1), could not succeed in utilizing the GPU well. Moreover, anything relying on direct sparse linear algebra (at least for solving the problem on one compute node) for performance would likely be untenable. Beyond this we had to be concerned with obtaining derivatives efficiently for our models, as well as possibly supporting two different architectures (AMD and Intel GPUs), which led to concerns about portability as well.

The first important decision was whether to keep relying on the interior-point method as the exterior structure, and in particular for the problem at the node (where memory requirements during factorization would be more limiting compared with the first stage, which could in principle rely on distributed storage). Although significant progress had been made in iterative sparse linear algebra for interior-point algorithms when used for convex optimization [1, 9], their performance still exhibited significant variance, even when the stopping tolerance was not very aggressive (about 10^{-4}). The state-of-the art reference [1] stated, "We hope that this work provides a first step toward the construction of generalizable preconditioners for linear and quadratic programming problems." Although we used older references when making our decisions in 2019–2020, we interpreted the situation that while some hope existed for reliably preconditioning interior point methods in the future, we did not seem to be there in 2021, even for convex problems. In addition, we had to contend with the nonconvexity of the ACOPF constraints. We thus decided to try approaches other than interior-point methods, although other parts of the project remained committed to them, and we believe that in some cases they will prove the dominant solver as they did in the past.

Given our experience with Lagrangian-based decomposition, we recognized that it would have to play a big role. It has been particularly successful when a decomposable structure is neither available at the linear algebra level nor exploitable for parallel computation on HPC, which is exactly the situation we are facing with next-generation HPC architecture with many GPUs (vs. CPUs). Nevertheless, which algorithmic combination would prove successful was difficult to foresee since the data on Lagrangian algorithms for nonlinear programming was scarce at the scale and type we were considering. Several requirements were identified.

[5] https://www.exascaleproject.org
[6] https://www.exascaleproject.org/research-project/exasgd/
[7] https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

R1 We can rapidly code many different algorithms and test them on the relevant scales; therefore the environment needed *expressivity and compactness*.

R2 Given our compressed timeline, we could not afford to prototype in one language (e.g., MATLAB) and implement at performance in another (say, C or C++). This problem is encountered particularly by the machine learning (ML) community and their frameworks. The Julia creators specifically identified and set out to solve this *two-language problem* [2].

R3 We get efficient *derivative information* in terms of both speed of execution and speed of development. A subtle issue is that no optimization modeling environment that evaluated derivatives on GPUs existed (and exists in 2021) so we had to consider rewriting a big portion of the modeling frontend in any case.

R4 We needed to cover multiple architectures simultaneously (NVIDIA for initial prototyping and AMD and Intel at production), so the approach needed to be amenable to *portability*.

R5 The nature of the exascale project requires *performance* comparable to C and C++ or any other conceivable environment.

After exploring our other assignments in the project and analyzing and minor prototyping other ways, in spring 2020 it became clear we had to rewrite our pipeline from scratch with a demonstration due date of September 2021. We decided to leave as one of our paths forward the ability to have the entire application from the modeling to the lowest-level computation done in Julia in a way that addressed R1–R5. The ExaSGD project itself had other pathways, not dependent on Julia, that we could and would leverage. It also seemed useful, however, given various other technological uncertainties, to maintain a path of full control over the entire application to allow ourselves full flexibility to deal with unforeseen events. In our judgment, Julia was the only solution at the time that could satisfy R1–R5 in our time horizon, and we decided to take that path.

One significant uncertainty remained, namely, the support for vendor libraries (CUDA, ROCm, oneAPI) that needed to be developed by the Julia community with whom we did have contact. Working with Julia has been a bit of adventure in the past seven years in terms of stability of interfaces between new releases; and, in that light, requirement R4 was in particular a bit more concerning than the others. Nevertheless, we also had experienced the energy and enthusiasm behind the Julia community, which, combined with that of our team and our experience, seemed to make the effort worthwhile and certainly intriguing and exciting.

We thus decided to take that leap. We share with the readers our understanding of the features of Julia that make it an excellent match for our requirements R1–R5, and we describe the capabilities that resulted by September 2021.

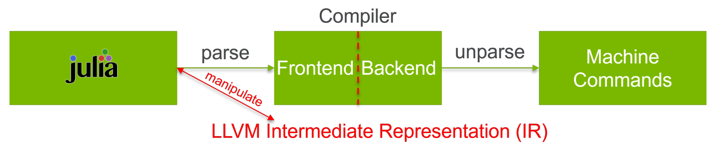


Figure 1: Julia's just-in-time compilation enables LLVM IR manipulation through expression transformations.

3 Some Features of Julia and Consequences for Its Capabilities

The fact that Julia offers expressivity and language compactness, combined with performance comparable to C and C++, has been already broadly discussed and employed in the scientific and, particularly, the optimization community, driven mainly by the excellent modeling language JuMP [6]. These features address requirements R1 and R5, and subsequently R2. Such features are also achievable to some extent by other environments such as the current effort to allow compilation of Python through efforts such as Numba [19], although it is hard to argue that no limits are imposed by a language that was designed for interpretation. However, Julia's design has some special features that make it versatile for achieving R4, and it is expected to achieve R3 by relying on the LLVM backend.

3.1 LLVM's Intermediate Representation and Julia's Access to It

Formal language is the scientific field of abstract analysis of both artificial and natural languages. It led to huge successes in both human linguistics when Chomsky tried to uncover a universal human language grammar, and it built the foundation of virtually any modern compiler. In essence, human language is arguably the most complete abstract model to describe the universe. Languages have to be fast-to-interpret, yet compact, information carriers. The interpretation recreates the information from a stream of data into structure about semantics and meaning. This parsing process from stream of data to structure is decomposed into well-defined stages: lexical analysis, semantical analysis, attribute annotation, and so on.

Compilers encode the semantics into an annotated abstract syntax tree (AST). LLVM started out in 2000 as the first compiler to strictly standardize that intermediate stage called LLVM intermediate representation (IR). The LLVM language represented by a syntax tree is then unparsed into machine instructions. Effectively, a compiler is nothing more than a language translator, usually from more complex human to more primitive machine hardware targeted structure.

Julia has direct access to the LLVM IR at runtime, see Figure 1. There is no need to parse code to apply automatic differentiation or define new paradigms such as operator overloading at the language level. Julia can access the code representation at the compiler level, and it can leverage the entire compiler infrastructure to manipulate the code semantics. This is distinct from any other language used in scientific computing (C/C++, Fortran, Python, MATLAB). Having a language that easily allows bidirectional communication between program and compiler is crucial for applied math and numerical science. That is exactly the mechanism by which algorithms and mathematics are combined to create new models and methods for solving problems and representing physical processes. Currently, portability happens to be the main driver of this evolution. Whether Julia has nailed this is up for debate. However, the direction is indisputably right.

3.2 Code Transformation in Julia and Differentiable Programming

Differentiable programming is a term coined by the machine learning community that requires any model used in the loss function to provide gradients. Any code written in domain-specific ML languages such as TensorFlow require its intrinsic operations (e.g., linear solvers) to provide their adjoint or gradient implementation. Thus, through AD machinery these implementations can then be integrated into machine learning models. Such features are of major interest in nonlinear optimization applications since effective derivative information access is a major plus.

The obvious downside of these frameworks is that all intrinsic operations and functions must be written in a domain-specific language that then goes through the entire architecture-specific compilation pipeline. As opposed to common scientific programming languages such as Fortran, C/C++, or Python, Julia is inherently differentiable because of the reasons related to its language design and closeness to the compiler (see Section 3.1). Additionally, differential programming requires the resolution of the two-language problem in R2 in order to achieve any seamless integration of numerical simulations into ML.

We illustrate how differentiating code can be defined as a graph transformation algorithm on the syntax tree of a program and thus the LLVM IR. Given an implementation of $y = f(x_1, x_2) = e^{(x_1 \cdot x_2)}$, we have the following single-assignment code.

```
z = x*y
w = exp(z)
```

An AD tool could generate the following gradient statement.

```
z = x*y
w = exp(z)
dz = exp(z)
dx = y*dz
dy = x*dz
```

Such differentiation can be implemented in various ways; but remember that Julia parses the original program into LLVM IR, which we represent with a simplified abstract syntax tree (AST) in Figure 2a.

Just like any code transformation, AD can be implemented as a manipulation of the AST. First, the AST is annotated with the derivative partials on the edges. Then, we get the derivative of an output with respect to an input by multiplying the edges on the path of the AST, giving us the expressions `dx=y*exp(z)` and `dy=x*exp(z)`, spanning themselves as an AST, as shown in Figure 2b. One advantage of this approach is that the compiler now has full access to the complete AST, being able to apply further optimizations (e.g., loop-invariant code motion [24] by leveraging the compiler pipeline while differentiating).

Moreover, access to the IR by the differentiation tools allows reusing the compiler optimizations, sometimes vastly reducing the compute time of the gradients by exploiting the structure identified by the compiler. Moses and Churavy [24] show that for a broad class of computational patterns, IR-based AD tools such as Enzyme result in factors of 2–100 improvement for a benchmark of classical computational patterns. Since the IR is language independent, one can, in theory, use it for differentiation for all programming languages that have an LLVM frontend. However, targeting the LLVM IR directly is a challenging endeavor. Julia alleviates it by offering indirect access to the LLVM IR through expressions that can be used to implement similar techniques (e.g., Diffractor, Zygote), albeit then restricted to the Julia language. Using the Julia AD tool Zygote, we illustrate in §3.4 how portability and AD are combined to give us portable and *highly optimized differentiation*.

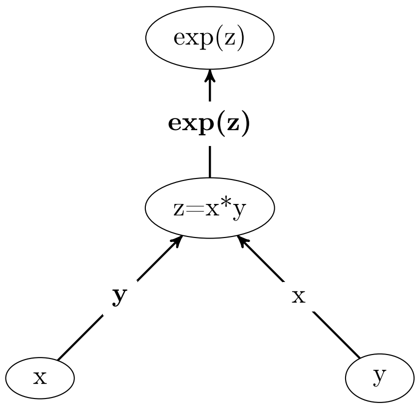
3.3 Portable Programming and Julia

In our codes we rely mainly on `KernelAbstractions.jl` for implementing our architecture-abstracted kernel functions. It unifies all the different vendor APIs (wrapped in Julia through `CUDA.jl`, `AMDGPU.jl`, `oneAPI.jl`) under one single API. Through a flip of a switch we are able to run the same code on CPU, AMD GPU, and NVIDIA GPU (see Figure 3).[8] This Julia native way of providing automatic code transformations for each architecture keeps all concerns neatly separated; AD, optimization, and hardware developers can develop in parallel with only minor interactions.

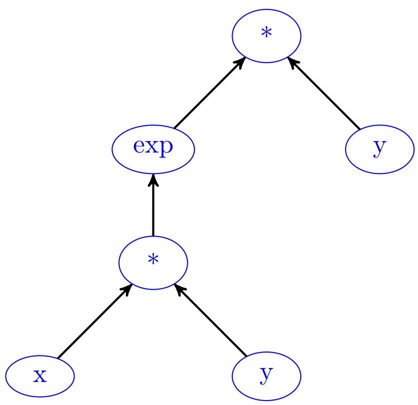
`KernelAbstractions.jl` defines various ways of partitioning work items and work groups that a kernel is applied to. In Figure 4 we show a simple work item partitioning that defines a global work item id (e.g., 1–100) and a local work item id that corresponds to the sequence number that a thread is working on (e.g., for five threads this would range from 1 to 20).

In the application code the kernels are instantiated by passing a `device` argument `f = tron(device, T, N)` and defining a number of workers `T` plus the number of work items `N`. Only at that moment is the actual kernel implementation generated based on the `device` argument, which could be `CUDADevice()` (CUDA), `ROCDevice()` (AMD), or `CPU()` for the host CPU architecture. After that the kernel can be launched asynchronously, and we can wait for its completion via `wait(event)`. These macros in Julia have built-in support for metaprogramming, which enables the algorith-

[8] Currently, we are working together with Julia Lab on oneAPI support for `KernelAbstractions.jl`.



(a) Parsed syntax tree $e^{x \cdot y}$, annotated with partial derivatives (in bold). The partial $\frac{\partial g}{\partial x}$ is equal to the multiplied path of the edges from x to g yielding the statement $e^{x \cdot y} \cdot y$.



(b) The generation of the expression $\frac{\partial g}{\partial x}$ differentiation should be implemented directly by a compiler. Julia provides access to the LLVM syntax tree as defined by the IR. In this way transformations can be applied recursively or combined with other transformations, since the AST of the gradient respects the same LLVM IR.

Figure 2: Differentiation on ASTs

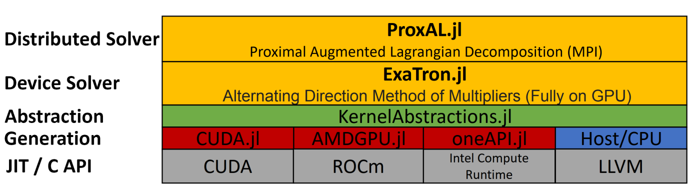


Figure 3: KernelAbstractions.jl serves as a portability layer for the big three GPU vendors NVIDIA (CUDA), AMD (ROCm), and Intel (oneAPI).

Listing (2.1) Kernel Function

```
@kernel function tron_kernel(args...)
# Global item ID
I = @index(Global, Linear)
# Local item ID
J = @index(Local, Linear)
...
end
```

Listing (2.2) Algorithm Function

```
function application(device)
    workers = T
    workitems = N
    # Generate
    tron = tron_kernel(device, T, N)
    # Execute kernel
    event = tron(args...)
    # Wait until event terminates
    wait(event)
end
application(CUDADevice())
# application(ROCDevice())
# application(CPU())
```

Figure 4: Example application implementation with kernel function implemented using KernelAbstractions.jl.

	Kokkos	KernelAbstractions.jl
Core source code (lines of code)	108,000	2,600
CUDA interface (lines of code)	13,190	442
Contributors	100+	20

Table 2: Overview of effort required for adding a new architecture that has LLVM backend support. Comparing C++ source code files in `kokkos/core/src` and `kokkos/core/src/Cuda` in version 3.5 to `KernelAbstractions.jl`

mic generation of code. A Julia program can algorithmically change itself at runtime based on input from another program. This is exactly what portability layers in other languages (e.g., Kokkos, Raja, HIP, OpenMP) try to achieve for various parallel programming paradigms, albeit with *much larger developer effort*, as illustrated in Table 2.

`KernelAbstractions.jl` relies on an interface layer for each GPU architecture that defines its `device` object (e.g, `CUDADevice()`). This layer itself relies on a host of generation packages such as `GPUCompiler.jl`. Adding another GPU architecture, however, requires only a thin interface layer. We illustrate this in Table 2 by comparing with the mature C++ portability library Kokkos that leverages C++ templates. This assumes vendor support for Julia that provides an implementation of CUDA in Julia as is currently done by CUDA.jl.

Architecture portability is challenging in most languages. Often, we are interested in developing solvers that work both in CPU and in GPU architectures, which can lead to complicated code in most traditional languages. To solve this problem, Julia offers the **multiple dispatch** paradigm, where different implementations of a method can be defined. This allows programmers to develop **device-independent code** by leveraging abstraction layers. In this way, higher levels of the code offer an abstraction that closely matches mathematical expressions, whereas lower layers can be highly tuned to ensure the highest levels of performance. An example is shown in snippet 2.3.

Listing 2.3: Thanks to Julia's multiple dispatch paradigm, routines such as this `solve` algorithm can be written in an abstract fashion allowing input arrays from different architectures.

```
using CUDA

function solve(A, b)
    x = similar(b)
    ...
    res = norm(A*x - b)
    ...
    return x
end

#A = Array([2 1; 1 2])
A = CuArray([2 1; 1 2])

#b = Array([1 1])
b = CuArray([1 1])

solve(A, b)
```

In snippet 2.3 we can see that switching between different architectures is a matter of selecting the type of the arrays. Rather than creating `solveCpu` and `solveGpu` methods, one can simply write one method that takes different types as input. Here we can see a `solve` method with an operation that computes the residual of the solution. The algorithm begins with a call to `similar`, which creates a solution vector of the given type. The residual norm involves a subtraction, a matrix-vector multiplication, and a norm computation. All of these operations are abstract and can accept arrays of different types. In runtime, Julia's powerful multiple dispatch will compile the appropriate method. This programming paradigm allows one to simplify code, promote abstract routines that closely match mathematical description, and enable extensibility of the code base.

3.4 Composable Programming: Portable Differentiation

We illustrate in snippet how all this machinery enables *composable programming* by combining two expression transformations (Listing (2.4)). Applying automatic differentiation to portability now is just a mere superposition of both codes; the actual application method implementation `f` is unaware of both notions.

Listing 2.4: The function f has no notion of AD or of CUDA. However, we can instantiate the input variable x as a CUDA, ROCm, or oneAPI array, and it will do differentiation through that function on the GPU. The parallelization happens through the broadcast operator ".".

```
using Zygote
using CUDA
# using AMDGPU
# using OneAPI

function f(x)
  return sum(x.*x)
end
x = [i/(1.0+i) for i in 1:10] |> CuArray
# x = [i/(1.0+i) for i in 1:10] |> ROCArray
# x = [i/(1.0+i) for i in 1:10] |> oneArray
dx = gradient(f, x)
```

We initialize an array `x` and move it to the GPU using the pipe operation `|> CuArray`. The function `f` computes the sum of an element-wise square operation. The operator `"."` is called the broadcast operator in Julia and behaves similarly to the Matlab dot operator. When passing `x` on the GPU to the function `f`, this function is instantiated in CUDA through the LLVM backend. Using the AD tool Zygote, we then generate the gradient of that CUDA implementation of `f`. No domain-specific language for the function `f` has been used.

4 Optimization on GPU Using Julia

As expressed in §2.3, R1 required prototyping and moving to production without recoding the most promising algorithms we encountered.

4.1 Algorithms Prototyped April 2020 – Sept 2021

Following is a summary of the algorithms we were able to prototype and our experience with them.

1. Simplex-type algorithms underlying sequential linear and quadratic programming for solving generic nonlinear optimization problems. Such algorithms are particularly powerful when we need to solve a number of problems with minor changes in problem data, for example, in a branch-and-bound method for solving mixed-integer programming. The Julia prototype of the SQP algorithm is available in https://github.com/exanauts/SQP.jl. Profiling has shown that other options are faster for nonlinear programming (NLP) proper, but we are still pursuing it for integer programming on GPUs.

2. ExaTron [18] is a GPU-based batch solver that can solve a large number (e.g., tens of thousands) of independent nonlinear nonconvex problems in parallel entirely on GPUs, without data transfers between the CPU and GPU. ExaTron implements a Newton-based trust-region method [20] as its underlying algorithm and has been implemented and tested on AMD and NVIDIA GPUs. Its implementation is available at https://github.com/exanauts/ExaTron.jl. Our original attempt was to apply ExaTron for an augmented Lagrangian formulation ACOPF, which did not have good performance; but as the unit solver of ExaADMM it proved excellent.

3. ExaADMM [17] implements an ADMM algorithm with convergence guarantees, providing a highly scalable solution method for solving alternating current optimal power flow problems on GPUs. All operations per ADMM iteration, such as primal and dual variable updates, are performed on GPUs. In ExaADMM, subproblem solves are significantly accelerated by ExaTron. Its implementation is available at `https://github.com/exanauts/ExaAdmm.jl`. This algorithm was tested on CPU, AMD GPU, and NVIDIA GPU platforms, and it is the production second-stage solver that we will describe in more detail in §4.2.

4. ExaADMM also supports online tracking of solutions as parameter changes over time. In contrast to interior-point algorithms, ADMM can exploit warm starting to significantly accelerate its solution time. In [17], we demonstrate the online tracking ability of ExaADMM, outperforming Ipopt in solving ACOPFs as load changes over time.

5. ProxAL: Our goal with `ProxAL.jl` `https://github.com/exanauts/ProxAL.jl` was to prototype a distributed parallel decomposition algorithm for solving linearly coupled multiblock-structured NLP problems. This is the production first-stage solver and decomposition manager; we describe it in more detail in §4.2.

6. MadNLP: MadNLP.jl `https://github.com/MadNLP/MadNLP.jl` is a pure-Julia NLP solver based on the widely used filter line-search interior-point algorithm. The solver was originally designed for CPUs [31], but we were able to quickly adapt it to run also on GPUs with dense linear algebra. Currently, MadNLP.jl can solve dense NLPs on NVIDIA GPUs; AMD and Intel GPU supports are on the way; and it is the core solver for Argos.jl.

7. Argos.jl `https://github.com/exanauts/Argos.jl` is a Julia package to solve OPF problems entirely on the GPU, leveraging the GPU capabilities of the nonlinear optimization solver MadNLP. The efficient resolution of OPF requires the evaluation of second-order derivatives. Argos.jl densifies the OPF problems by using a reduced space approach. By combining these two elements together with the interior-point algorithm implemented MadNLP, we get a tractable method to solve OPF problems on the GPU [26], which we currently tested on NVIDIA.

4.2 Our September 2021 Production Configuration

From the seven pathways described in §4.1, based on the prospects we saw in April 2021, we settled on ProxAL.jl for the first-stage decomposition manager and ExaADMM as the second-stage resolution of the problem at the node, as depicted in Figure 5. To explain some of the scalability and

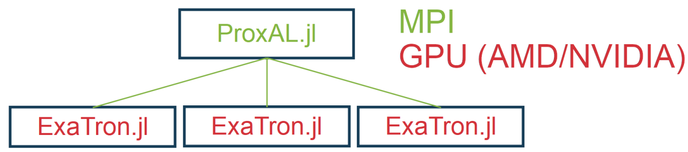


Figure 5: Two-stage parallelization strategy. ProxAL decomposes along the periods, and each ExaTron instance solves a single ACOPF.

performance features, we will briefly describe their algorithmic features. Detailed description of these algorithms and their performance can be found in [32], [17].

Both algorithms can be described as Lagrangian-based decomposition algorithms for nonlinear optimization problems of the following form:

$$\underset{x_1,\ldots,x_N}{\text{minimize}} \quad \sum_{j=1}^{N} f_j(x_j) \tag{1a}$$

$$\text{subject to} \quad \sum_{j=1}^{N} A_j x_j = b, \tag{1b}$$

$$x_j \in X_j \quad \forall j = 1,\ldots,N, \tag{1c}$$

where X_t are compact (possibly nonconvex) sets, $A_t \in \mathbb{R}^{m\times n_t}$ are matrices, and $f_t : \mathbb{R}^{n_t} \mapsto \mathbb{R}$ are continuously differentiable (possibly nonconvex) functions. In the above formulation, the decision variables are grouped into T blocks $(x_1, x_2, \ldots, x_T)$ coupled only via the linear constraints $\sum_{t=1}^{T} A_t x_t = b$.

The coupling constraint (1b) is one of three types.

C1 Coupling between successive periods in a multiperiod approach, typically the difference between two successive (in time) generator settings connected by the ramping variable. N can be very large in this case, between a few dozen and a few thousand.

C2 Coupling between stages in two-stage scenario optimization-type problems (the scenarios may be of either the uncertainty or the contingency type). Then $N = 2$.

C3 Coupling between branches, buses, and generators in a network. In this case, N is proportional to the number of *types* of grid components in a network (typically low).

To decompose C1 and C2, we use ProxAL; for C3 we use ExaADMM, fundamentally to solve one ACOPF problem on a compute node.

At its heart, `ProxAL.jl` performs ADMM-style updates while requiring only local solutions of the linearly coupled individual-block NLP problems. Unlike standard ADMM schemes for nonconvex problems, however, `ProxAL.jl` introduces certain proximal terms to the augmented Lagrangian function in a controlled manner and performs only Jacobi-style updates [32]. This approach is essential where N may be very large for which Gauss–Seidel sweeps would

Table 3: Performance of solving ACOPF from cold start

Data	ADMM	Time (secs)		Solution Quality	
	Iterations	ADMM	Ipopt	$\|c(x)\|_\infty$	$\frac{\|f-f^*\|}{f^*}$
1354pegase	823	1.99	2.44	1.23e-03	0.05%
2869pegase	1,230	4.19	6.09	3.64e-04	0.03%
9241pegase	1,372	7.95	50.80	1.12e-03	0.08%
13659pegase	1,529	8.70	131.12	1.25e-03	0.05%
ACTIVSg25k	3,307	36.05	118.64	1.21e-02	0.09%
ACTIVSg70k	2,897	69.81	469.03	1.52e-02	2.20%

severely hamper performance. Users of `ProxAL.jl` can easily switch between a JuMP backend (where they can specify a local CPU- or GPU-based NLP solver) or the aforementioned GPU-based ExaTron backend for solving individual NLP problems. ProxAl.jl can decompose the problem by time periods and scenarios and distribute the computations over multiple compute nodes with multiple GPUs by using the MPI protocol [32]. This enables one to solve multiperiod contingency-constrained ACOPF—extremely large-scale nonlinear nonconvex optimization on GPU clusters, some of which we demonstrate below.

ExaADMM.jl For decomposing the network problem for the scenario C3, our basic idea is to exploit the massive parallel computing capability of GPUs to decompose the original problem into a number of *small subproblems* that can be efficiently solved in parallel on GPUs through batch nonlinear programming by ExaTron [18]. In this case we may have a large number of such subproblems; however, the massive parallel computing capability of GPUs enables us to promptly process them using batch nonlinear programming, as described in [18].

An example of our approach in the context of ACOPF computation has been demonstrated in [18, 17]. To scale ACOPF solutions with multiple GPUs on supercomputers, we implement an ADMM algorithm (ExaADMM) that decomposes the original problem into a number of small component subproblems, similar to [23], each of which represents an electric grid component as in [22, 23]. The key advantages of this approach are that most subproblems (particularly for buses and generators) have closed-form solutions, and the other subproblems are nonconvex nonlinear optimization each with 6 variables, 8 bound constraints, and 2 convex inequality constraints only. As a result, each iteration of the ADMM algorithm can stream the computation of the tiny nonlinear problems to GPUs. Those nonlinear problems are solved in batch on GPUs by using ExaTron. Such a batch can be solved far faster on GPUs than on CPUs, as can be seen in Figure 6 (where 6 GPUs solve a large batch 40 times faster compared with 40 CPUs.)

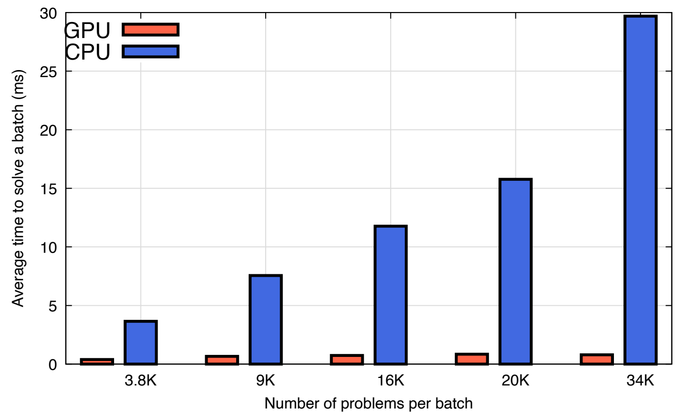


Figure 6: Performance comparison: 40 CPUs vs 6 GPUs on a Summit node. The GPUs solve batches of 34K problems 40 times faster than the CPUs do.

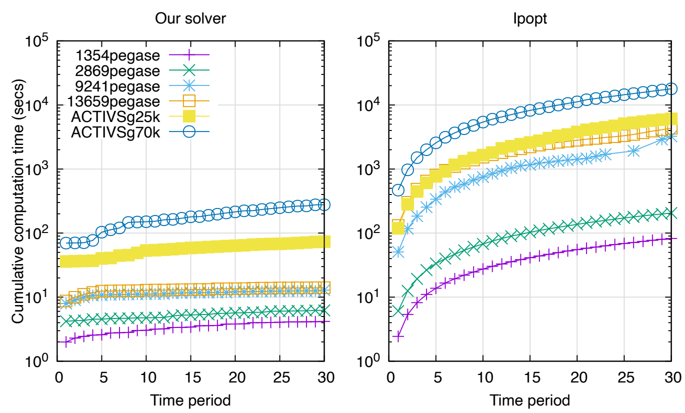


Figure 7: Cumulative computation time of warm start.

4.3 Performance

Derivative computation performance As stated in Section 4.1, our `Argos.jl` algorithm involves the accumulation of a reduced Hessian $W_{uu} = \nabla^2_{uu} L$ in the reduced control space of the optimal power flow with variable u. The reduced Hessian is evaluated efficiently using the Implicit Function theorem, and by computing in parallel batches of Hessian-vector products $W_{uu} \cdot v$ using Forward-over-reverse AD. The full method is described in [25]. From first-order adjoint codes (reverse pass), we generate second-order derivatives automatically via `ForwardDiff.jl` [28][9]. The benefit is that this automatically generates code that allows a SIMD propagation of multiple forward directions (equal to the number of colors of the sparsity pattern) on the GPU, achieving excellent performance. In addition, we use a batched linear solver to compute multiple projections in the reduced space at once by running N batches. The performance over various case sizes and batch sizes can be seen in Figure 10. Note that as a reference we compare the GPU implementation against a sequential CPU implementation. The CPU implementation would likely exhibit further speedup through threaded parallelism. Note also that the code currently uses handwritten adjoints because of limitations of current AD tools that either miss a feature or would impact performance. However, we plan on integrating `Enzyme.jl`[10] at some point.

Cold-Start Performance of ExaADMM Table 3 demonstrates the computational performance of ExaADMM when we solve ACOPF from cold start. The results show up to 6 times faster computation time of ExaADMM than that of Ipopt. Since ADMM is a first-order method, solu-

[9] `https://github.com/JuliaDiff/ForwardDiff.jl`
[10] `https://github.com/wsmoses/Enzyme.jl`

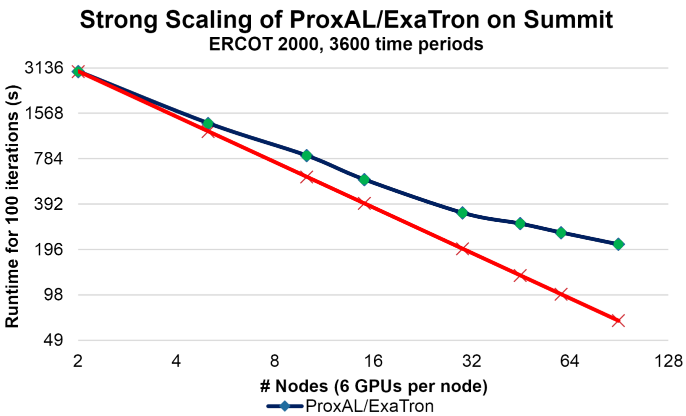


Figure 8: Scaling of ProxAL + ExaTron on Summit at OLCF

Figure 9: Speed of light for ExaTron on a single GPU solving the ACTIVSg70k case.

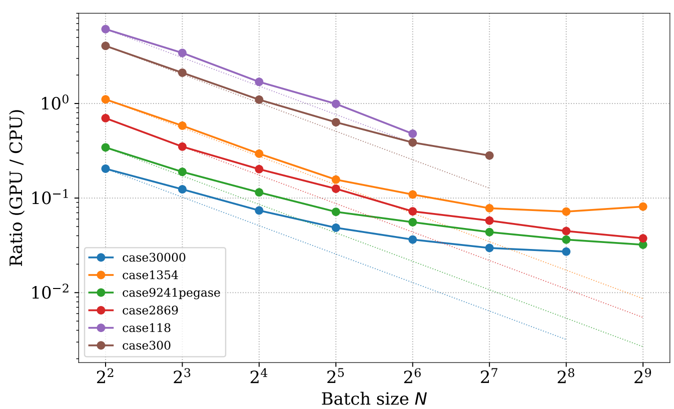


Figure 10: Parallel scaling of the total reduced Hessian accumulation $\nabla^2 F$ with batch size N. A ratio value < 1 indicates a faster runtime compared with that of UMFPACK and AutoDiff on the CPU in absolute time. The dotted lines indicate the linear scaling reference. Lower values imply a higher computational intensity.

tion accuracy may not be comparable to that of the second-order method; however, it still shows a reasonable accuracy in $\|c(x)\|_\infty$, particularly given the standards for the application in practice (where a few percentage error is acceptable). The relative objective gap shows little difference in most cases between two objective values computed by ExaADMM and Ipopt, respectively, except for the 70K case, although we used the same stopping criterion as in [33]. We note that the problem is the largest of its type that would make sense to solve, since it is of the size of the U.S. Eastern Interconnect.

Warm-Start Performance of ExaADMM Figure 7 shows ExaADMM's online tracking capability by warm start. We solve 30 time periods one time period at a time in a rolling horizon fashion in this case. In most networks, ExaADMM finds solutions for all the 30 time periods by exploiting warm start, even before Ipopt finds a solution for the first period.

Peak Performance of ExaADMM An important issue in the Exascale project is the fraction of peak performance achieved, since the metrics of success revolve around the intensity of the floating-point operations. Because the ProxAL decomposition layer is light, much of this performance will be achieved by the nodal solver, in this case ExaADMM. In Figure 9 we see that ExaADMM achieves about 10% of the "speed of light" intensity on NVIDIA V100, which is much in line with classical numerical libraries on CPUs. While we believe we have ideas how to improve it by factors of 2–3 (since 10% was essentially our first result), the performance is definitely in the range acceptable for exascale-class supercomputers.

Scalability of ProxAL Running on an entire exascale machine will be carried out by scenario decomposition that is managed by ProxAL. As expected from the thinness of the ProxAL layer, good scalability of the approach is fairly straightforward for the large run in Figure 8.

Portability As indicated in requirement R4 in §2.3, we aimed to see whether we can support multiple architectures. At the current stage, our kernel abstraction has machine-independent code and, as can be seen in Figure 11, creates identical results on two different architectures: AMD GPUs and NVIDIA GPUs.

4.4 Summary Status

The advanced features of Julia, some of which we described in §3, and its (in our opinion) extraordinary proficiency allowed us to overcome several suboptimal outcomes as described in §4.1 and to successfully rewrite an application from scratch in about 18 months and put us within striking distance of being able to run simulations compatible with exascale supercomputing requirements. Julia allowed us to fulfill all our requirements R1–R5 that we laid out in §2.3. Given its excellent design and compactness, the language Julia has another salient feature that is familiar to MATLAB users: people with moderate computing experience can pick it up incredibly fast, and it does not require major sacrifices in performance. These features allowed us to proceed in multiple directions simultaneously and thus to effectively mitigate the fairly large space of technological uncertainty that we faced as described in §2.2 and §2.3. Despite the large number of authors of this communication, the average funding was for about 3 full-time employees during this time, so we believe the development throughput efficiency to have been excellent.

```
┌mschanen@login2 /autofs/nccs-svm1_home1/mschanen/git/ProxAL.jl ‹ecp/milestone5-ka*›
└$ hostname -d
spock.olcf.ornl.gov
┌mschanen@login2 /autofs/nccs-svm1_home1/mschanen/git/ProxAL.jl ‹ecp/milestone5-ka*›
└$ git rev-parse --short HEAD
e3a085e
┌mschanen@login2 /autofs/nccs-svm1_home1/mschanen/git/ProxAL.jl ‹ecp/milestone5-ka*›
└$ cat out
ProxAL/ExaTron 4 ranks, 2 periods, 19 contingencies
Creating problem: 52.182024954
Benchmark Start
---------------------------------------------------------------------------------------------------------------
iter ramp_err   ramp_err   ctgs_err   ctgs_err   dual_error lyapunov_f    rho_t    rho_c  theta_t  theta_c      tau
     (penalty)  (actual)   (penalty)  (actual)
---------------------------------------------------------------------------------------------------------------
   0 2.7086e+00 2.7086e+00 2.8642e+00 2.8642e+00 0.0000e+00 2.4459e+03 0.00e+00 0.00e+00 1.00e+06 1.00e+06 0.00e+00
[ Info: tau-update not finalized when decompCtgs = true
   1 3.1696e-01 3.1696e-01 1.5841e+00 1.5841e+00 4.5424e-02 2.4270e+03 1.00e-03 1.00e-03 1.00e+06 1.00e+06 2.00e-03
   2 1.9158e-01 1.9158e-01 1.8108e+00 1.8108e+00 1.0228e-02 2.4260e+03 1.00e-03 2.00e-03 1.00e+06 1.00e+06 4.00e-03
   3 1.4575e-01 1.4575e-01 1.8118e+00 1.8118e+00 2.1105e-02 2.4356e+03 2.00e-03 4.00e-03 1.00e+06 1.00e+06 8.00e-03
   4 5.2708e-02 5.2708e-02 1.8066e+00 1.8066e+00 1.3437e-02 2.4456e+03 2.00e-03 8.00e-03 1.00e+06 1.00e+06 1.60e-02
```

(a) ProxAL.jl/ExaTron.jl on Spock (x86)/AMD (ROCm)

```
┌mschanen@login3 /gpfs/alpine/scratch/mschanen/csc359/ProxAL.jl ‹ecp/milestone5-ka*›
└$ hostname -d
summit.olcf.ornl.gov
┌mschanen@login3 /gpfs/alpine/scratch/mschanen/csc359/ProxAL.jl ‹ecp/milestone5-ka*›
└$ git rev-parse --short HEAD
e3a085e
┌mschanen@login3 /gpfs/alpine/scratch/mschanen/csc359/ProxAL.jl ‹ecp/milestone5-ka*›
└$ cat proxal-tests.1511211
Wed Oct 13 16:58:57 EDT 2021
┌ Warning: HSA runtime has not been built, runtime functionality will be unavailable.
│ Please run Pkg.build("AMDGPU") and reload AMDGPU.
└ @ AMDGPU /ccs/proj/csc359/michel/julia_depot_spock/dev/AMDGPU/src/AMDGPU.jl:166
...
┌ Warning: ROCm-Device-Libs were not found, device intrinsics will be unavailable.
│ Please run Pkg.build("AMDGPU") and reload AMDGPU.
└ @ AMDGPU /ccs/proj/csc359/michel/julia_depot_spock/dev/AMDGPU/src/AMDGPU.jl:195
ProxAL/ExaTron 4 ranks, 2 periods, 19 contingencies
Creating problem: 115.342806677
Benchmark Start
---------------------------------------------------------------------------------------------------------------
iter ramp_err   ramp_err   ctgs_err   ctgs_err   dual_error lyapunov_f    rho_t    rho_c  theta_t  theta_c      tau
     (penalty)  (actual)   (penalty)  (actual)
---------------------------------------------------------------------------------------------------------------
   0 2.7086e+00 2.7086e+00 2.8642e+00 2.8642e+00 0.0000e+00 2.4459e+03 0.00e+00 0.00e+00 1.00e+06 1.00e+06 0.00e+00
[ Info: tau-update not finalized when decompCtgs = true
   1 3.1696e-01 3.1696e-01 1.5841e+00 1.5841e+00 4.5424e-02 2.4270e+03 1.00e-03 1.00e-03 1.00e+06 1.00e+06 2.00e-03
   2 1.9158e-01 1.9158e-01 1.8108e+00 1.8108e+00 1.0228e-02 2.4260e+03 1.00e-03 2.00e-03 1.00e+06 1.00e+06 4.00e-03
   3 1.4575e-01 1.4575e-01 1.8118e+00 1.8118e+00 2.1105e-02 2.4356e+03 2.00e-03 4.00e-03 1.00e+06 1.00e+06 8.00e-03
   4 5.2708e-02 5.2708e-02 1.8066e+00 1.8066e+00 1.3437e-02 2.4456e+03 2.00e-03 8.00e-03 1.00e+06 1.00e+06 1.60e-02
```

(b) ProxAL.jl/ExaTron.jl on Summit (PowerPC)/NVIDIA (CUDA): AMDGPU.jl is reporting that no AMD GPUs are available

Figure 11: Exact match of results with ProxAL/ExaTron and two dramatically different architectures on the ERCOT 2000 bus case with 2 periods and 19 contingencies

5 Outlook

In the near term, we are not completely satisfied with the convergence rates of some of our algorithms, which we believe can be improved. Moreover, we intend to pursue a viable second-order method, and not just first-order decompositions, to achieve more robustness in convergence behavior. We also aim to obtain a more formalized modeling environment suitable for streaming processors as well as to pursue forays in integer programming on GPUs with the goal of solving other problems of interest. Nevertheless, the fact that we are within the performance range of our exascale target after 18 months has more than justified our leap of faith in using Julia, and it is a testimony for the conviction of the Julia community, with whom we needed to interact at various times.

Our decision to go with Julia came also from another driver, the one concerning hardware risk, which we believe (speaking strictly in the name of the authors of the paper and not the ECP project or any other entity) affects not just the exascale project but potentially all of technical computing and perhaps particularly numerical optimization. Given the concerns about power consumption we illustrated above, we find that a real possibility exists that most average hardware of the future will require using an accelerator such as a GPU. Many of these accelerators will be designed with data-driven applications in mind, and many of the libraries we currently use on CPUs will be unable to produce the kind of performance we have been accustomed to (see Table 1). Such a situation would mean having to rewrite from scratch many of our approaches, which is to large extent what we have described in §4.1—and, moreover, on architectures that will keep changing. Julia proved to have excellent programmer productivity on GPUs in addition to its portability and performance features that we outlined. All these features made our endeavor bearable in this case and even exciting, given the novelty of the situation and the learning opportunities. We hope our experience will be of use to our readers if they find themselves in similar circumstances.

Acknowledgment

We thank our JuliaLab collaborators Alan Edelman, Valentin Churavy, and Julian Samaroo. We also acknowledge our colleagues from the ExaSGD project for input and help at various stages of this projects. In particular we thank Kasia Świrydowicz and Slaven Peleš for feedback and discussions around their profiling presented in Table 1. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR) under Contract DE-AC02-06CH11357.

References

[1] Luca Bergamaschi et al. "A new preconditioning approach for an interior point-proximal method of multipliers for linear and convex quadratic programming". In: *Numerical Linear Algebra with Applications* 28.4 (2021), e2361.

[2] Jeff Bezanson et al. "Julia: A Fresh Approach to Numerical Computing". In: *SIAM Review* 59.1 (2017), pp. 65–98. DOI: 10.1137/141000671. URL: https://doi.org/10.1137/141000671.

[3] Naiyuan Chiang, Cosmin G Petra, and Victor M Zavala. "Structured nonconvex optimization of large-scale energy systems using PIPS-NLP". In: *2014 Power Systems Computation Conference*. IEEE. 2014, pp. 1–7.

[4] Brian C Dandurand, Kibaek Kim, and Sven Leyffer. "A Bilevel Approach for Identifying the Worst Contingencies for Nonconvex Alternating Current Power Systems". In: *SIAM Journal on Optimization* 31.1 (2021), pp. 702–726.

[5] Wei Deng et al. "Parallel multi-block ADMM with $o(1/k)$ convergence". In: *Journal of Scientific Computing* 71.2 (2017), pp. 712–736.

[6] Iain Dunning, Joey Huchette, and Miles Lubin. "JuMP: A Modeling Language for Mathematical Optimization". In: *SIAM Review* 59.2 (2017), pp. 295–320. DOI: 10.1137/15M1020575.

[7] M.J. Flynn. "Very high-speed computing systems". In: *Proceedings of the IEEE* 54.12 (1966), pp. 1901–1909. DOI: 10.1109/PROC.1966.5273.

[8] Robert Fourer, David M Gay, and Brian W Kernighan. *AMPL*. Boyd & Fraser, 1995.

[9] Jacek Gondzio. "Interior point methods 25 years later". In: *European Journal of Operational Research* 218.3 (2012), pp. 587–601.

[10] Ge Guo et al. "Integration of progressive hedging and dual decomposition in stochastic integer programs". In: *Operations Research Letters* 43.3 (2015), pp. 311–316.

[11] William E Hart et al. *Pyomo–optimization modeling in Python*. Vol. 67. Springer, 2017.

[12] Joey Huchette, Miles Lubin, and Cosmin Petra. "Parallel algebraic modeling for stochastic optimization". In: *2014 First Workshop for High Performance Technical Computing in Dynamic Languages*. IEEE. 2014, pp. 29–35.

[13] Kibaek Kim, Audun Botterud, and Feng Qiu. "Temporal decomposition for improved unit commitment in power system production cost modeling". In: *IEEE Transactions on Power Systems* 33.5 (2018), pp. 5276–5287.

[14] Kibaek Kim and Brian Dandurand. "Scalable branching on dual decomposition of stochastic mixed-integer programming problems". In: *Mathematical Programming Computation (to appear)* (2021).

[15] Kibaek Kim, Cosmin G Petra, and Victor M Zavala. "An asynchronous bundle-trust-region method for dual decomposition of stochastic mixed-integer programming". In: *SIAM Journal on Optimization* 29.1 (2019), pp. 318–342.

[16] Kibaek Kim and Victor M Zavala. "Algorithmic innovations and software for the dual decomposition method applied to stochastic mixed-integer programs". In: *Mathematical Programming Computation* 10.2 (2018), pp. 225–266.

[17] Youngdae Kim and Kibaek Kim. "Accelerated Computation and Tracking of AC Optimal Power Flow Solutions using GPUs". In: *arXiv preprint arXiv:2110.06879* (2021).

[18] Youngdae Kim et al. "Leveraging GPU batching for scalable nonlinear programming through massive Lagrangian decomposition". In: *arXiv preprint arXiv:2106.14995* (2021).

[19] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. "Numba: A LLVM-based Python JIT Compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. LLVM '15. Austin, Texas: Association for Computing Machinery, 2015. ISBN: 9781450340052. DOI: 10.1145/2833157.2833162. URL: https://doi.org/10.1145/2833157.2833162.

[20] Chih-Jen Lin and Jorge J. Moré. "Newton's method for large bound-constrained optimization problems". In: *SIAM Journal on Optimization* 9.4 (1999), pp. 1100–1127.

[21] Miles Lubin et al. "Scalable stochastic optimization of complex energy systems". In: *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2011, pp. 1–10.

[22] Sleiman Mhanna, Archie C. Chapman, and Gregor Verbič. "Component-Based Dual Decomposition Methods for the OPF Problem". In: *Sustainable Energy, Grids and Networks* 16 (Dec. 2018), pp. 91–110. DOI: 10.1016/j.segan.2018.04.003.

[23] Sleiman Mhanna, Gregor Verbic, and Archie C. Chapman. "Adaptive ADMM for Distributed AC Optimal Power Flow". In: *IEEE Transactions on Power Systems* 34.3 (May 2019), pp. 2025–2035. DOI: 10.1109/TPWRS.2018.2886344.

[24] William S Moses and Valentin Churavy. "Instead of Rewriting Foreign Code for Machine Learning, Automatically Synthesize Fast Gradients". In: *arXiv preprint arXiv:2010.01709* (2020).

[25] Francois Pacaud et al. "General Memory-Independent Lower Bound for MTTKRP". In: *Proceedings of the 2022 SIAM Conference on Parallel Processing for Scientific Computing (PP)*. to appear. URL: https://web.cels.anl.gov/~mschanen/SIAM-PP.pdf.

[26] François Pacaud et al. "A Feasible Reduced Space Method for Real-Time Optimal Power Flow". In: *arXiv preprint arXiv:2110.02590* (2021).

[27] Cosmin G Petra and Mihai Anitescu. "A preconditioning technique for Schur complement systems arising in stochastic optimization". In: *Computational Optimization and Applications* 52.2 (2012), pp. 315–344.

[28] Jarrett Revels, Miles Lubin, and Theodore Papamarkou. "Forward-mode automatic differentiation in Julia". In: *arXiv preprint arXiv:1607.07892* (2016).

[29] Sarah M Ryan et al. "Toward scalable, parallel progressive hedging for stochastic unit commitment". In: *2013 IEEE Power & Energy Society General Meeting*. IEEE. 2013, pp. 1–5.

[30] Michel Schanen et al. "Toward multiperiod AC-based contingency constrained optimal power flow at large scale". In: *2018 Power Systems Computation Conference (PSCC)*. IEEE. 2018, pp. 1–7.

[31] Sungho Shin et al. "Graph-Based Modeling and Decomposition of Energy Infrastructures". In: *IFAC-PapersOnLine* 54.3 (2021), pp. 693–698.

[32] Anirudh Subramanyam et al. "A Globally Convergent Distributed Jacobi Scheme for Block-Structured Nonconvex Constrained Optimization Problems". In: *arXiv preprint arXiv:2112.09027* (2021).

[33] Kaizhao Sun and Xu Andy Sun. "A Two-Level ADMM algorithm for AC OPF With Global Convergence Guarantees". In: *IEEE Transactions on Power Systems* 36.6 (2021), pp. 5271–5281. DOI: 10.1109/TPWRS.2021.3073116.

[34] Kasia Świrydowicz et al. "Linear solvers for power grid optimization problems: a review of GPU-accelerated linear solvers". In: *Parallel Computing* (2021), p. 102870.

[35] Jean-Paul Watson and David L Woodruff. "Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems". In: *Computational Management Science* 8.4 (2011), pp. 355–370.

[36] Weiqi Zhang, Kibaek Kim, and Victor M Zavala. "On the Tightness and Scalability of the Lagrangian Dual Bound for the Alternating Current Optimal Power Flow Problem". In: *arXiv preprint arXiv:2104.03788* (2021).

Mihai Anitescu
Mathematics and Computer Science Division
Argonne National Laboratory
Department of Statistics, University of Chicago
anitescu@mcl.anl.gov
https://www.mcs.anl.gov/~anitescu/

Kibaek Kim
Mathematics and Computer Science Division
Argonne National Laboratory
kimk@anl.gov
https://kibaekkim.github.io/

Youngdae Kim
Mathematics and Computer Science Division
Argonne National Laboratory
youngdae@anl.gov
https://www.anl.gov/profile/youngdae-kim-0

Adrian Maldonado
Mathematics and Computer Science Division
Argonne National Laboratory
maldonadod@anl.gov
https://www.anl.gov/profile/daniel-adrian-maldonado

François Pacaud
Mathematics and Computer Science Division
Argonne National Laboratory
fpacaud@anl.gov
https://frapac.github.io/pages/about/

Vishwas Rao
Mathematics and Computer Science Division
Argonne National Laboratory
vhebbur@anl.gov
https://www.anl.gov/profile/vishwas-rao

Michel Schanen
Mathematics and Computer Science Division
Argonne National Laboratory
mschanen@anl.gov
https://www.anl.gov/profile/michel-schanen

Sungho Shin
Mathematics and Computer Science Division
Argonne National Laboratory
sshin@anl.gov
https://sunghoshin.com

Anirudh Subramanian
Mathematics and Computer Science Division
Argonne National Laboratory
asubramanyam@anl.gov
https://scholar.google.com/citations?user=yDf5WlEAAAAJ

Conic optimization for solving convex quadratic optimization with indicators

Andrés Gómez
Daniel J. Epstein Department of Industrial and Systems Engineering
University of Southern California
gomezand@usc.edu
https://sites.google.com/usc.edu/gomez

1 Introduction

Consider a mixed-integer optimization problem with indicator variables

$$\min_{x,y} \; a^\top x + c^\top y + y^\top Q y \tag{1a}$$
$$\text{s.t. } Gx + Hy \leq b \tag{1b}$$
$$y_i(1-x_i) = 0 \qquad \forall i \in \{1,\dots,n\} \tag{1c}$$
$$x \in \{0,1\}^n,\; y \in \mathbb{R}^n, \tag{1d}$$

where $Q \in \mathbb{R}^{n\times n}$ is a given positive-semidefinite matrix, $G, H \in \mathbb{R}^{m\times n}$, $a, c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. Problem (1) arises pervasively in practice, e.g., in portfolio optimization [8] (where the quadratic term $y^\top Q y$ corresponds to the variance of the portfolio) and in sparse regression [11] (where the quadratic function corresponds to the widely used quadratic loss function). The complementary constraints (1c) and binary constraints are used to force the logical condition that $x_i = 0 \implies y_i = 0$, thus indicator variables x correspond to the support of the continuous variables y.

Solution approaches for non-convex optimization problems such as (1) are often based on the solution of suitable convex relaxations of the problem. In this paper we consider relaxations of the mixed-integer epigraph of the specific quadratic function appearing in (1), that is, the set

$$X = \left\{(x,y,t) \in \{0,1\}^n \times \mathbb{R}^{n+1} : t \geq y^\top Q y,\; (1b)-(1d)\right\}.$$

Note that since problem (1) is NP-hard [10] even without constraints (1b), obtaining a complete description of the convex hull of X may not be possible. Thus, most approaches in the literature are based on characterizing convex hulls of simpler version of X – by restricting matrix Q to be of a simple class and ignoring additional constraints – and using the resulting insights to design practical convex relaxations for the general problem (1). In this paper, we review recent results concerning relaxations of set X, and then discuss open questions.

2 Conic quadratic relaxations

Off-the-shelf codes for mixed-integer optimization have made great strides over the past two decades towards solving mixed-integer conic quadratic optimization problems. Therefore, an initial push towards solving (1) focused on constructing conic-quadratic relaxations of X.

We first point out that using generalized disjunctive programming [9], it is possible to construct conic quadratic extended formulations of X, even with arbitrary constraints (1b). The resulting formulations, however, may have up to $\mathcal{O}(n2^n)$ additional variables. If matrix Q is of rank $r < n$, then one can obtain conic quadratic extended formulations with $\mathcal{O}(n^{r+1})$ additional variables [18]. A direct application of these convexifications for (1) results in problems with substantially more variables, that cannot be handled effectively by off-the-shelf solvers. Nonetheless, disjunctive programming remains a powerful tool for deriving sharper convexifications by projecting out the additional variables [4, 19, 2]. We now turn to classes of set X whose convex hull is conic quadratic-representable in the original space of variables.

Perspective reformulation If $n = 1$, and set X reduces to $X_1 = \left\{(x,y,t) \in \{0,1\} \times \mathbb{R}^2 : t \geq qy^2,\; y(1-x) = 0\right\}$ with $q > 0$, then the closure of the convex hull of X_1 is $\text{cl conv}(X_1) = \left\{(x,y,t) \in [0,1] \times \mathbb{R}^2 : xt \geq qy^2\right\}$. This result, which is a consequence of the convexifications in [9], was first observed in [15], and shown to be conic quadratic representable in [1]. Figure 1 depicts the perspective function $f(x,y) = y^2/x$ describing $\text{cl conv}(X_1)$.

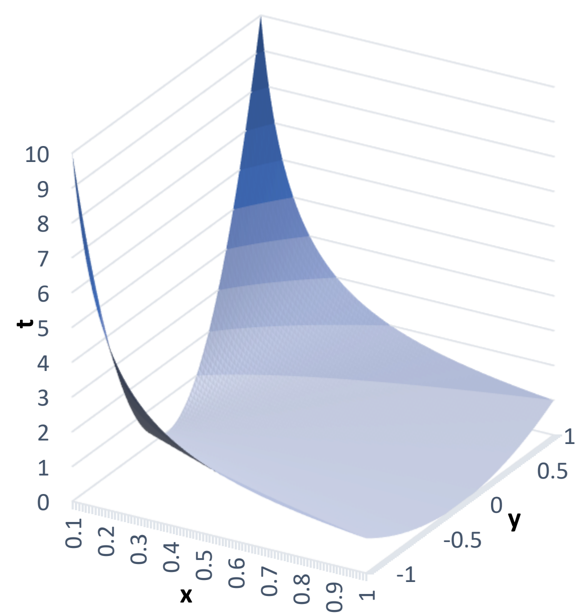


Figure 1: Graph of the perspective function $t = y^2/x$.

The perspective reformulation can be used to strengthen relaxations of problem (1) as follows: if $Q = D + R$ where $D \in \mathbb{R}^{n\times n}_+$ is a diagonal matrix and $R \succeq 0$, then we can reformulate (1) as

$$\min_{x,y,t} \; a^\top x + c^\top y + \sum_{i=1}^n D_{ii} t_i + y^\top R y \tag{2a}$$
$$\text{s.t. } t_i x_i \geq y_i^2 \quad \forall i \in \{1,\dots,n\} \tag{2b}$$
$$(1b) - (1d),\; t \in \mathbb{R}^n_+. \tag{2c}$$

A matrix D is often directly available from the context. For example, in portfolio optimization, the covariance matrix Q has to be estimated from data; a common method to use to do so is to use a factor model, in which case $Q = F\Sigma F^\top + D$ where $F \in \mathbb{R}^{n\times k}$ is the exposure matrix, $\Sigma \in \mathbb{R}^{k\times k}$ is the factor covariance matrix and D is a diagonal matrix with the residual variances. In regression problems, given a model matrix $A \in \mathbb{R}^{k\times n}$ and regularization term $\lambda > 0$, the goal is to minimize $\|b - Ay\|_2^2 + \lambda\|y\|_2^2$; in this case, $Q = A^\top A + \lambda I$, and one can set $D = \lambda I$. If a matrix D is not immediately available, a simple heuristic is to set $D = \lambda_{\min}(Q)I$, where $\lambda_{\min}(Q)$ denotes the minimum eigenvalue of Q. More sophisticated heuristics that involve solving semidefinite optimization problems (SDPs) have also been proposed [16].

Due to the relative simplicity of the convex relaxation induced by the perspective reformulation, and difficulties arising from solving large nonlinear mixed-integer optimization problems (arising from alternatives to the perspective reformulation) via branch-and-bound, approaches based on formulation (2) are still the preferred method to solve (1) to optimality. There has been a recent push to devise specialized methods for solving (2), which avoid using standard off-the-shelf technology for handling conic quadratic constraints [13, 14, 20] and scale to larger instances.

The rank-one reformulation Consider a rank-one matrix Q, that is, $Q = qq^\top$ for some $q \in \mathbb{R}^n$ and set X reduces to $X_{R1} = \left\{(x, y, t) \in \{0, 1\}^n \times \mathbb{R}^{n+1} : t \geq (q^\top y)^2,\ (1c)\right\}$, where we assume that $q_i \neq 0$ for all $i \in \{1, \ldots, n\}$ (otherwise variables x_i and y_i can be ignored when studying X_{R1}). The closure of the convex hull of X_{R1} is

$$\text{cl conv}(X_{R1}) = \Big\{(x, y, t) \in [0, 1]^n \times \mathbb{R}^{n+1} : \\ t \geq (q^\top y)^2,\ (\boldsymbol{e}^\top x)t \geq (q^\top y)^2\Big\},$$

where $\boldsymbol{e} \in \mathbb{R}^n$ is a vector of 1s [3]. The perspective reformulation is a special case of rank-one reformulation with $n = 1$.

To use the rank-one reformulation for problem (1), a similar decomposition approach as used by the perspective reformulation can be used. In particular, if $Q = \sum_{j\in J} q_j q_j^\top + R$ where $R \succeq 0$ and J is some index set, then problem (1) can be reformulated as

$$\min_{x,y,t}\ a^\top x + c^\top y + \sum_{j\in J} t_j + y^\top R y \tag{3a}$$

$$\text{s.t. } t_j \geq (q_j^\top y)^2,\ (\boldsymbol{e}_j^\top x)t_j \geq (q_j^\top y)^2 \quad \forall j \in J \tag{3b}$$

$$(1b) - (1d),\ t \in \mathbb{R}_+^J, \tag{3c}$$

where $\boldsymbol{e}_j \in \mathbb{R}^n$ such that $(\boldsymbol{e}_j)_i = 1$ if $(q_j)_i \neq 0$ and $(\boldsymbol{e}_j)_i = 0$ otherwise.

Note that each element $j \in J$ can be associated with a subset of $\{1, \ldots, n\}$, corresponding to the non-zero entries of q_j. Thus J can be associated with a subset of the power set of $\{1, \ldots, n\}$, that is, $J \subseteq 2^n$, and we can rewrite constraints (3b) as

$$\min\left\{1, \sum_{i\in S} x_i\right\} t_S \geq \left(\sum_{i\in S} (q_S)_i y_i\right)^2 \quad \forall S \in J. \tag{4}$$

The natural convex relaxation of (3) is particularly strong when vectors q_j are sparse, e.g., $J = \{S \subseteq \{1, \ldots, n\} : |S| \leq \kappa\}$ for a small $\kappa \in \mathbb{Z}_+$. Suitable vectors q_j may be naturally available in some applications, e.g., in sparse inference problems with Markov Random Fields [6], but in general an additional optimization problem may need to be solved in order to find them (discussed in the next section).

There is strong evidence suggesting that if matrix Q is not rank-one, then cl conv(X) is not conic quadratic representable in the original space of variables. Indeed, current conic quadratic formulations for the full-rank case with $n = 2$ already require several additional variables [17], while a recent description in the original space of variables [7] uses a infinite number of conic quadratic constraints. Thus, strong relaxations of X may require more sophisticated nonlinear constraints, discussed next.

3 Semidefinite relaxations

Reformulations (2) and (3) can serve as the basis for stronger conic relaxations of (1). Given $J \subseteq 2^n$ and vectors $\{q_S\}_{S\in J}$, where the support of $q_S \in \mathbb{R}^n$ is S, define

$$\phi\left(\{q_S\}_{S\in J}\right) \overset{\text{def}}{=} \min\ a^\top x + c^\top y + \sum_{S\in J} t_S + y^\top R y \tag{5a}$$

$$\text{s.t. } (4),\ (1b) \tag{5b}$$

$$x \in [0, 1]^n, y \in \mathbb{R}^n, t \in \mathbb{R}_+^J, \tag{5c}$$

where $R = Q - \sum_{S\in J} q_S q_S^\top$ is completely determined by vectors q_S. Observe that (5) is the natural convex relaxation of (3), obtained by dropping the complementary and integrality constraints. Moreover, if $J = \{\{i\} : i \in \{1, \ldots, n\}\}$, then (5) is the natural convex relaxation of (2). The vectors q_S that yield the strongest relaxation correspond to optimal solutions of the SDP

$$\max_{\{q_S\}_{S\in J}} \phi\left(\{q_S\}_{S\in J}\right) \text{ s.t. } Q - \sum_{S\in J} q_S q_S^\top \succeq 0. \tag{6}$$

Moreover, it can be shown that min and max can be interchanged in (6) without any duality gap, obtaining an SDP relaxation of (1). Such formulations were first derived in the context of the perspective reformulation [12, 25], and then generalized to rank-one reformulations [3].

We now present an alternative derivation of such formulations based on more direct arguments, following the discussions in [18, 22]. The nonlinear objective of (1) can be linearized with the introduction of additional variables $Y \overset{\text{def}}{=} yy^\top$ as $a^\top x + c^\top y + \langle Q, Y\rangle$, where $\langle Q, Y\rangle = \sum_{i,j} Q_{ij} Y_{ij}$ is the Frobenius inner product. Moreover, since $Q \succeq 0$, a relaxation of (1) in this extended space is

$$\min_{x,y,Y}\ a^\top x + c^\top y + \langle Q, Y\rangle \tag{7a}$$

$$\text{s.t. } Y \succeq yy^\top \tag{7b}$$

$$(1b),\ x \in [0, 1]^n,\ y \in \mathbb{R}^n,\ Y \in \mathbb{R}^{n\times n}. \tag{7c}$$

Relaxation (7) is equivalent in terms of strength to the natural convex relaxation of (1). Relaxation (7) can be strengthened by noting that for any vector $q \in \mathbb{R}^n$, if $Y = yy^\top$, then $\langle qq^\top, Y\rangle = (q^\top y)^2$. Moreover, if $x \in \{0, 1\}^n$, then inequality

(4) holds at equality. Thus, for any $S \in J$, letting Y_S be the submatrix of Y indexed by S and letting y_S be the vector of y indexed by S, we find that

$$\begin{aligned}
&\langle q_S q_S^\top, Y_S\rangle \geq \frac{\left(\sum_{i\in S}(q_S)_i y_i\right)^2}{\min\left\{1, \sum_{i\in S} x_i\right\}},\ \forall q_S \in \mathbb{R}^n \\
\Leftrightarrow\ &\langle q_S q_S^\top, Y_S - \frac{y_S y_S^\top}{\min\left\{1, \sum_{i\in S} x_i\right\}}\rangle \geq 0,\ \forall q_S \in \mathbb{R}^n \\
\Leftrightarrow\ &Y_S - \frac{y_S y_S^\top}{\min\left\{1, \sum_{i\in S} x_i\right\}} \succeq 0 \\
\Leftrightarrow\ &\begin{pmatrix} Y_S & y_S \\ y_S^\top & \min\left\{1, \sum_{i\in S} x_i\right\} \end{pmatrix} \succeq 0, \qquad (8)
\end{aligned}$$

where the last equivalence follows from the Schur complement. Thus constraints (8) can be added to (7) for every $S \in J$; the resulting conic formulation is stronger than (5) (for any choice of q vectors), and they are equivalent if vectors are chosen according to (6), see [12, 18]. Similar relaxations have been used with convexifications other than rank-one [17, 19], although the resulting semidefinite relaxations are more complicated.

Connections with other conic relaxations An alternative idea to derive strong convex relaxations for (1) is based directly on the linearization of the quadratic term as $\langle Q, Y\rangle = \sum_{i,j} Q_{ij}Y_{ij}$, and thus revolves around relaxations of the mixed-integer set

$$Z = \left\{(x, y, Y) \in \{0,1\}^n \times \mathbb{R}^{n+n^2} : Y \succeq yy^\top,\ (1b) - (1d)\right\}.$$

Set Z is substantially more general than X: while a description cl conv(X) would result in ideal relaxations of (1) for a specific matrix Q, a description of cl conv(Z) would result in ideal relaxations of (1) for *any* matrix Q. As a consequence, cl conv(Z) is also substantially more complex [2]. Whereas relaxations based on X may be more suitable to solve problem (1), set Z plays a critical role in more general problems, e.g., problems with multiple quadratic constraints and/or non-convex quadratic terms.

Nonetheless, the analysis in this section shows how relaxations for the simpler set X can be lifted into relaxations for Z. Indeed, given any fixed matrix $Q \succeq 0$, the system of inequalities

$$\langle Q, Y\rangle \geq t,\ (x, y, t) \in \text{cl conv}(X) \qquad (9)$$

is valid for cl conv(Z). Moreover, cl conv(Z) is fully described by the (infinite) system of inequalities (9), one for each $Q \succeq 0$ –where set X implicitly depends on Q. In particular, inequalities (8) can be interpreted as inequalities describing a portion of cl conv(Z), corresponding to the matrices of rank one.

Connections with regularization Consider the "ℓ_0"-regularized problem

$$\min_{y\in\mathbb{R}^n} c^\top y + y^\top Qy + \|y\|_0, \qquad (10)$$

where $\|y\|_0 = \sum_{i=1}^n \mathbb{1}_{\{y_i\neq 0\}}$ is the support of y. Problem (10) is a special case of (1) with $a = \boldsymbol{e}$ and no constraints (1b). A standard approach in the statistical literature is to approximate (10) with a continuous optimization problem of the form

$$\min_{y\in\mathbb{R}^n} c^\top y + y^\top Qy + r(y), \qquad (11)$$

where $r : \mathbb{R}^n \to \mathbb{R}_+$ is a suitable regularization function used as a proxy of the support of y. The most widely used regularization is the ℓ_1 approximation [21] where $r(y) = \sum_{i=1}^n |y_i|$, but several alternatives have been proposed in the literature. The regularization function r is often non-differentiable at the origin and separable, and possibly non-convex.

As first pointed out in [12], relaxations of the form (5) can be interpreted as non-convex regularizations, where

$$r(y) = y^\top(R - Q)y + \min_{x,t}\sum_{S\in J} t_S \text{ s.t. } (4), (5c).$$

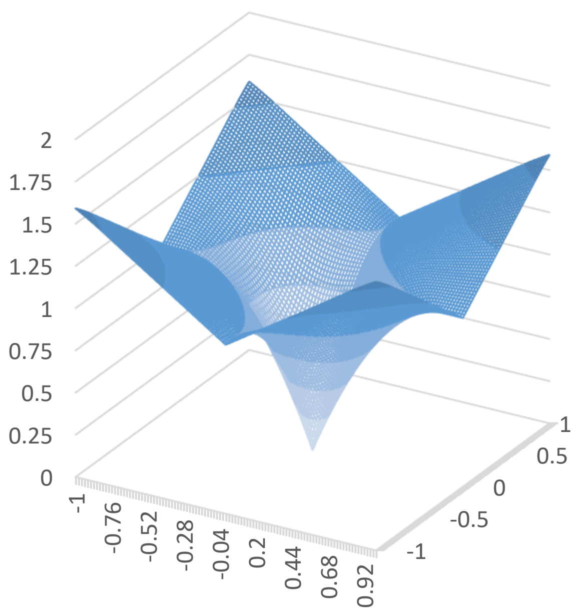
While the regularization functions are non-convex, the regularized problem (11) is still convex. Intuitively, solving problem (6) amounts to finding a maximal non-convex regularization (among a class of penalty functions) that ensures the convexity of the ensuing regularized problem. In fact, as shown in [12], convexifications based on the perspective reformulation is equivalent to the minimax concave penalty (MC+) [24], proposed independently in the statistical literature. On the other hand, the convexifications based on rank-one reformulations result in a new class of non-separable regularizations that better approximate the support of y. Figure 2 depicts the regularization functions $r(y)$ induced by perspective and rank-one reformulations in an example with $n = 2$.

4 Additional results and open questions

While the conic relaxations presented in this paper substantially improve upon the natural convex relaxation of (1), it has been unclear whether such relaxations match the structure of cl conv(X), whether they are "necessary" to describe it (since cl conv(X) is not polyhedral, there is not a clear characterization of necessary inequalities), or whether cl conv(X) is SDP-representable in the original space of variables. Nonetheless, recent results [7] shed some light into these questions when there are no constraints (1b). In particular, separation over cl conv(X) can be accomplished by solving an optimization problem with a single positive-definite constraint and (a possibly exponential number of) linear constraints. Moreover, it is possible to construct an extended SDP-representable formulation of (1) with a quadratic number of additional variables, a single conic constraint, and exponentially many linear inequalities. These result suggest than the class of semidefinite optimization problems is sufficiently rich to represent cl conv(X).

Most convexification approaches in the literature study sets with no constraints (1b), thus a natural question is how much additional constraints change the structure of the convex hulls. Existing works [7, 22, 23] point out that if the

(a) Perspective reformulation



(b) Rank-one reformulation

Figure 2: Interpretation of (5) as non-convex regularizations $r(y)$ with $n = 2$.

constraints involve only the indicator variables x, then the overall structure of the convex remains unchanged, that is, the class of inequalities required to describe $\mathrm{cl\ conv}(X)$ is the same. However, if the constraints include even simple nonnegativity constraints $y \geq 0$, then the structure seems to be substantially different from the unconstrained case [4, 5, 19]. Studies involving additional and more sophisticated constraints on the continuous variables are largely missing in the literature.

A critical challenge towards devising practical methods to solve (1) to optimality is designing an effective branch-and-bound method. On the one hand, practical off-the-shelf mixed-integer optimization solvers are able to handle conic quadratic constraints, but as pointed out in this note, those are insufficient to describe the underlying convex hull of interest. On the other hand, the relevant convex hulls seem to be SDP-representable, but designing an efficient general-purpose branch-and-bound algorithm for mixed-integer semidefinite optimization problems is very difficult. Nonetheless, the results to date suggest that a compromise may be possible: indeed, describing $\mathrm{cl\ conv}(X)$ does not require arbitrary semidefinite constraints, but rather constraints with a particular structure (and, in an extended formulation, a single conic constraint). Thus, devising methods that can handle this specific class of relaxations may be much simpler than designing a general purpose MISDP solver.

References

[1] M Selim Aktürk, Alper Atamtürk, and Sinan Gürel. “A strong conic quadratic reformulation for machine-job assignment with controllable processing times”. In: *Operations Research Letters* 37 (2009), pp. 187–191.

[2] Kurt M Anstreicher and Samuel Burer. “Quadratic optimization with switching variables: The convex hull for $n = 2$”. In: *Mathematical Programming* 188 (2021), pp. 421–441.

[3] Alper Atamtürk and Andrés Gómez. “Rank-one Convexification for Sparse Regression”. In: *arXiv preprint arXiv:1901.10334* (2019).

[4] Alper Atamtürk and Andrés Gómez. “Strong formulations for quadratic optimization with M-matrices and indicator variables”. In: *Mathematical Programming* 170 (2018), pp. 141–176.

[5] Alper Atamtürk and Andrés Gómez. “Supermodularity and valid inequalities for quadratic optimization with indicators”. In: *arXiv preprint arXiv:2012.14633* (2020).

[6] Alper Atamtürk, Andrés Gómez, and Shaoning Han. “Sparse and Smooth Signal Estimation: Convexification of L0-Formulations”. In: *Journal of Machine Learning Research* 22.52 (2021), pp. 1–43.

[7] Alper Atamtürk et al. “On the convex hull of convex quadratic optimization problems with indicator variables”. In: *Technical report* (2021).

[8] Daniel Bienstock. “Computational study of a family of mixed-integer quadratic programming problems”. In: *Mathematical Programming* 74.2 (1996), pp. 121–140.

[9] Sebastián Ceria and João Soares. “Convex programming for disjunctive convex optimization”. In: *Mathematical Programming* 86 (1999), pp. 595–614.

[10] Yichen Chen et al. “Strong NP-hardness for sparse optimization with concave penalty functions”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 740–747.

[11] Alison Cozad, Nikolaos V Sahinidis, and David C Miller. "Learning surrogate models for simulation-based optimization". In: *AIChE Journal* 60.6 (2014), pp. 2211–2227.

[12] Hongbo Dong, Kun Chen, and Jeff Linderoth. "Regularization vs. Relaxation: A conic optimization perspective of statistical variable selection". In: *arXiv preprint arXiv:1510.06083* (2015).

[13] Antonio Frangioni, Fabio Furini, and Claudio Gentile. "Approximated perspective relaxations: A project and lift approach". In: *Computational Optimization and Applications* 63 (2016), pp. 705–735.

[14] Antonio Frangioni, Fabio Furini, and Claudio Gentile. "Improving the approximated projected perspective reformulation by dual information". In: *Operations Research Letters* 45 (2017), pp. 519–524.

[15] Antonio Frangioni and Claudio Gentile. "Perspective cuts for a class of convex 0–1 mixed integer programs". In: *Mathematical Programming* 106 (2006), pp. 225–236.

[16] Antonio Frangioni and Claudio Gentile. "SDP diagonalizations and perspective cuts for a class of nonseparable MIQP". In: *Operations Research Letters* 35 (2007), pp. 181–185.

[17] Antonio Frangioni, Claudio Gentile, and James Hungerford. "Decompositions of Semidefinite Matrices and the Perspective Reformulation of Nonseparable Quadratic Programs". In: *Mathematics of Operations Research* 45.1 (2020), pp. 15–33.

[18] Shaoning Han and Andrés Gómez. "Compact extended formulations for low-rank functions with indicator variables". In: *arXiv preprint arXiv:2110.14884* (2021).

[19] Shaoning Han, Andrés Gómez, and Alper Atamtürk. "2x2 convexifications for convex quadratic optimization with indicator variables". In: *arXiv preprint arXiv:2004.07448* (2020).

[20] Hussein Hazimeh, Rahul Mazumder, and Ali Saab. "Sparse regression at scale: Branch-and-bound rooted in first-order optimization". In: *arXiv preprint arXiv:2004.06152* (2020).

[21] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288.

[22] Linchuan Wei, Andrés Gómez, and Simge Küçükyavuz. "Ideal formulations for constrained convex optimization problems with indicator variables". In: *Forthcoming in Mathematical Programming* (2021).

[23] Weijun Xie and Xinwei Deng. "Scalable algorithms for the sparse ridge regression". In: *SIAM Journal on Optimization* 30 (2020), pp. 3359–3386.

[24] Cun-Hui Zhang. "Nearly unbiased variable selection under minimax concave penalty". In: *The Annals of statistics* 38.2 (2010), pp. 894–942.

[25] Xiaojin Zheng, Xiaoling Sun, and Duan Li. "Improving the performance of MIQP solvers for quadratic programs with cardinality and minimum threshold constraints: A semidefinite program approach". In: *INFORMS Journal on Computing* 26 (2014), pp. 690–703.

In Memoriam

Shabbir Ahmed (1969 – 2019)

Photo from the Georgia Tech ISYE institutional page

Shabbir Ahmed was the Anderson-Interface Chair and Professor in the H. Milton Stewart School of Industrial and Systems Engineering at Georgia Institute of Technology. Shabbir passed away on June 19th, 2019, at the age of 49, after a hard-fought battle with cancer. Shabbir was born in Bangladesh and received his B.Eng. in mechanical engineering from Bangladesh University of Engineering and Technology in 1993. He moved to the U.S. in 1995, where he received his M.S. and Ph.D. from the University of Illinois at Urbana-Champaign in 1997 and 2000, respectively.

Shabbir joined the H. Milton School of Industrial and Systems Engineering at Georgia Institute of Technology in the year 2000 and had been a faculty member there since then. His contributions to integer programming, stochastic programming and their applications were substantial. He is recognized as a leading (perhaps *the* leading) researcher in the field of stochastic integer programming. Shabbir's accomplishments were recognized with many awards throughout his career. Shabbir won first prize in the George B. Dantzig Dissertation Award in 2000. He received the NSF career award in 2002 and the IBM Research Award twice, in 2002 and 2005. He was recognized by Georgia Tech as the Coca-Cola Junior Professor in 2005 and Dean's Professor and Stewart Faculty Fellow in 2014. In 2017, Shabbir was named the Anderson-Interface Chaired Professor and was selected as an INFORMS fellow. In 2018, Shabbir received the prestigious Farkas Prize from the INFORMS Optimization Society.

Stochastic programming considers optimization models in which some of the parameters are uncertain and modeled as random variables. Integer programming considers optimization models in which some of the decision variables are constrained to be discrete (e.g., 0 or 1), enabling modeling of discrete decisions and logical relationships. Problems containing stochastic parameters or integer variables (or both) arise in a huge variety of important application areas. In addition to advancing the theory and methodology for solving stochastic and integer programming problems, Shabbir made significant contributions to applications of these problems in areas including in energy [13, 21, 30, 35], supply chain and logistics [5, 16, 25, 24], and finance [22].

Shabbir's work significantly advanced the field of stochastic integer programming. Already during his Ph.D. study, Shabbir published influential works on a finite branch and bound method for two-stage stochastic programs with integer variables in the second stage [8] and multi-stage stochastic capacity expansion problem with integer variables in all stages [6]. After joining Georgia Tech, Shabbir and his coauthors designed a method for deriving cuts for a multi-stage stochastic program based on cuts for a deterministic version of the problem [15]. In [1, 14] he derived novel scenario decomposition algorithms for binary stochastic programs. Most recently, in [36] Shabbir and his co-authors proposed the groundbreaking Stochastic Dual Dynamic integer Programming (SDDiP) method for solving multi-stage stochastic integer programs, a notoriously difficult problem class due to the triple challenge of uncertainty, dynamics, and discrete decisions. SDDiP has been shown to be effective in solving many multistage stochastic integer programs, for example, portfolio optimization [36], unit commitment [35], hydropower scheduling [18], facility location [34], among many others. To enhance SDDiP, in [4] Shabbir and co-authors introduced a related method for multi-stage stochastic integer programs that does not require discretization of the state variables.

Shabbir also made breakthrough contributions to a particular class of stochastic programming problem known as chance-constrained programs (CCPs). CCPs allow a decision-maker to limit risk by ensuring that the chosen decisions lie within a desired set with high probability. Shabbir's work addressed three fundamental challenges for solving CCPs: the high-dimensional integration required to evaluate feasibility of a CCP, the potential non-convexity of the feasible region, and the hardness of optimization. In [7, 19, 22], Shabbir and his collaborators provided rigorous analyses of the Monte Carlo sample average approximation method for solving chance constrained programs, resolving the issue of high-dimensional integration. To address the nonconvexity, Shabbir and his co-authors proposed novel relaxation or approximation schemes [10, 2, 9, 29], where the resulting problems are convex programs. To speed up solving a chance constrained program to optimality, Shabbir and his co-authors developed valid inequalities [20, 9, 23], decomposition approaches [26], and analyzed the effectiveness of a family of cuts known as quantile cuts [31].

Shabbir also had significant impact on the direction of research taken by the mixed-integer programming community beyond his work on stochastic integer programming. In [28], Shabbir and his co-authors developed the first-known branch and bound algorithms using polyhedral approximations to solve mixed-integer second order conic programs. Shabbir and his collaborators analyzed different mixed-integer models for non-separable piece-wise linear optimization [27]. In [17], they developed a new class of effective valid inequalities for solving the challenging mixed-integer bilinear programs, and in [12], they introduced and studied forbidden vertices with important application in improving the classical integer L-shaped method [11]. Finally, in the series of papers [3, 32, 33] Shabbir and his collaborators explored various connections between cutting-planes for structured linear and nonlinear integer programs and submodular optimization.

As a colleague, Shabbir was well-known for always finding time for everyone, for his kindness and his wisdom. As a collaborator, Shabbir had a great collaborative attitude and was a visionary – he was quick to see connections to related and emerging areas and forge productive partnerships both in academia and industries. He was striving to finish the works with his collaborators even one week before his death. As a mentor, he always prioritized students' needs over his own interests and was always willing to listen to students and to help them as much as he could. Over 19 years of his career, he successfully advised over twenty-five Ph.D. students.

Farewell, Shabbir! May you rest in peace! We will miss you.

Santanu S. Dey, H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology.
Email: santanu.dey@isye.gatech.edu

Jim Luedtke, Department of Industrial and Systems Engineering, University of Wisconsin, Madison.
Email: jim.luedtke@wisc.edu

Nick Sahinidis, H. Milton Stewart School of Industrial and Systems Engineering and School of Chemical and Biomolecular Engineering, Georgia Institute of Technology.
Email: nikos@gatech.edu

Weijun Xie, Department of Industrial and Systems Engineering, Virginia Tech.
Email: wxie@vt.edu

References

[1] Shabbir Ahmed. "A scenario decomposition algorithm for 0–1 stochastic programs". In: *Operations Research Letters* 41.6 (2013), pp. 565–569.

[2] Shabbir Ahmed. "Convex relaxations of chance constrained optimization problems". In: *Optimization Letters* 8.1 (2014), pp. 1–12.

[3] Shabbir Ahmed and Alper Atamtürk. "Maximizing a class of submodular utility functions". In: *Mathematical programming* 128.1 (2011), pp. 149–169.

[4] Shabbir Ahmed, Filipe Goulart Cabral, and Bernardo Freitas Paulo da Costa. "Stochastic Lipschitz dynamic programming". In: *Mathematical Programming* (2020), pp. 1–39.

[5] Shabbir Ahmed, Ulaş Çakmak, and Alexander Shapiro. "Coherent risk measures in inventory problems". In: *European Journal of Operational Research* 182.1 (2007), pp. 226–238.

[6] Shabbir Ahmed and Nikolaos V Sahinidis. "An approximation scheme for stochastic integer programs arising in capacity expansion". In: *Operations Research* 51.3 (2003), pp. 461–471.

[7] Shabbir Ahmed and Alexander Shapiro. "Solving chance-constrained stochastic programs via sampling and integer programming". In: *State-of-the-art decision-making tools in the information-intensive age*. Informs, 2008, pp. 261–269.

[8] Shabbir Ahmed, Mohit Tawarmalani, and Nikolaos V Sahinidis. "A finite branch-and-bound algorithm for two-stage stochastic integer programs". In: *Mathematical Programming* 100.2 (2004), pp. 355–377.

[9] Shabbir Ahmed and Weijun Xie. "Relaxations and approximations of chance constraints under finite distributions". In: *Mathematical Programming* 170.1 (2018), pp. 43–65.

[10] Shabbir Ahmed et al. "Nonanticipative duality, relaxations, and formulations for chance-constrained stochastic programs". In: *Mathematical Programming* 162.1-2 (2017), pp. 51–81.

[11] Gustavo Angulo, Shabbir Ahmed, and Santanu S Dey. "Improving the integer L-shaped method". In: *INFORMS Journal on Computing* 28.3 (2016), pp. 483–499.

[12] Gustavo Angulo et al. "Forbidden vertices". In: *Mathematics of Operations Research* 40.2 (2015), pp. 350–360.

[13] Beste Basciftci et al. "Stochastic Optimization of Maintenance and Operations Schedules Under Unexpected Failures". In: *IEEE Transactions on Power Systems* 33.6 (2018), pp. 6755–6765. DOI: 10.1109/TPWRS.2018.2829175.

[14] Yan Deng, Shabbir Ahmed, and Siqian Shen. "Parallel Scenario Decomposition of Risk-Averse 0-1 Stochastic Programs". In: *INFORMS Journal on Computing* 30.1 (2018), pp. 90–105.

[15] Yongpei Guan, Shabbir Ahmed, and George L Nemhauser. "Sequential pairing of mixed integer inequalities". In: *Discrete Optimization* 4.1 (2007), pp. 21–39.

[16] Yongpei Guan et al. "A branch-and-cut algorithm for the stochastic uncapacitated lot-sizing problem". In: *Mathematical Programming* 105.1 (2006), pp. 55–84.

[17] Akshay Gupte et al. "Solving mixed integer bilinear problems using MILP formulations". In: *SIAM Journal on Optimization* 23.2 (2013), pp. 721–744.

[18] Martin N Hjelmeland et al. "Nonconvex medium-term hydropower scheduling by stochastic dual dynamic integer programming". In: *IEEE Transactions on Sustainable Energy* 10.1 (2018), pp. 481–490.

[19] James Luedtke and Shabbir Ahmed. "A sample approximation approach for optimization with probabilistic constraints". In: *SIAM Journal on Optimization* 19.2 (2008), pp. 674–699.

[20] James Luedtke, Shabbir Ahmed, and George L Nemhauser. "An integer programming approach for linear programs with probabilistic constraints". In: *Mathematical programming* 122.2 (2010), pp. 247–272.

[21] Ali Irfan Mahmutoğulları et al. "The Value of Multi-Stage Stochastic Programming in Risk-Averse Unit Commitment Under Uncertainty". In: *IEEE Transactions on Power Systems* 34.5 (2019), pp. 3667–3676. doi: 10.1109/TPWRS.2019.2902511.

[22] Bernardo K Pagnoncelli, Shabbir Ahmed, and Alexander Shapiro. "Sample average approximation method for chance constrained programming: theory and applications". In: *Journal of optimization theory and applications* 142.2 (2009), pp. 399–416.

[23] Feng Qiu et al. "Covering linear programming with violations". In: *INFORMS Journal on Computing* 26.3 (2014), pp. 531–546.

[24] Tjendera Santoso et al. "A stochastic programming approach for supply chain network design under uncertainty". In: *European Journal of Operational Research* 167.1 (2005), pp. 96–115.

[25] Peter Schütz, Asgeir Tomasgard, and Shabbir Ahmed. "Supply chain design under uncertainty using sample average approximation and dual decomposition". In: *European Journal of Operational Research* 199.2 (2009), pp. 409–419.

[26] S. Shen, J.C. Smith, and S. Ahmed. "Expectation and chance-constrained models and algorithms for insuring critical paths". In: *Management Science* 56 (2010), pp. 1794–1814.

[27] Juan Pablo Vielma, Shabbir Ahmed, and George Nemhauser. "Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions". In: *Operations research* 58.2 (2010), pp. 303–315.

[28] Juan Pablo Vielma, Shabbir Ahmed, and George L Nemhauser. "A lifted linear programming branch-and-bound algorithm for mixed-integer conic quadratic programs". In: *INFORMS Journal on Computing* 20.3 (2008), pp. 438–450.

[29] Weijun Xie and Shabbir Ahmed. "Bicriteria approximation of chance-constrained covering problems". In: *Operations Research* 68.2 (2020), pp. 516–533.

[30] Weijun Xie and Shabbir Ahmed. "Distributionally robust chance constrained optimal power flow with renewables: A conic reformulation". In: *IEEE Transactions on Power Systems* 33.2 (2017), pp. 1860–1867.

[31] Weijun Xie and Shabbir Ahmed. "On quantile cuts and their closure for chance constrained optimization problems". In: *Mathematical Programming* 172.1 (2018), pp. 621–646.

[32] Jiajin Yu and Shabbir Ahmed. "Maximizing a class of submodular utility functions with constraints". In: *Mathematical Programming* 162.1-2 (2017), pp. 145–164.

[33] Jiajin Yu and Shabbir Ahmed. "Polyhedral results for a class of cardinality constrained submodular minimization problems". In: *Discrete Optimization* 24 (2017), pp. 87–102.

[34] Xian Yu and Siqian Shen. "Multistage distributionally robust mixed-integer programming with decision-dependent moment-based ambiguity sets". In: *Mathematical Programming* (2020), pp. 1–40.

[35] Jikai Zou, Shabbir Ahmed, and Xu Andy Sun. "Multistage stochastic unit commitment using stochastic dual dynamic integer programming". In: *IEEE Transactions on Power Systems* 34.3 (2018), pp. 1814–1823.

[36] Jikai Zou, Shabbir Ahmed, and Xu Andy Sun. "Stochastic dual dynamic integer programming". In: *Mathematical Programming* 175.1 (2019), pp. 461–502.

Bulletin

Email items to siagoptnews@lists.mcs.anl.gov *for consideration in the bulletin of forthcoming issues.*

1 Event Announcements

IPCO 2022

The 23rd Conference on Integer Programming and Combinatorial Optimization (IPCO XXIII) will be held at the Eindhoven University of Technology from June 27-29, 2022. Before the conference, a summer school will take place on June 25-26. More details are available at the conference website: https://www.ipco2022.com.

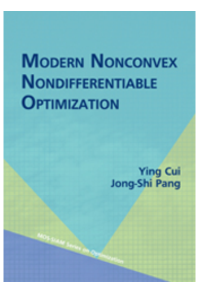
ISMP 2022

The 2022 edition of ISMP will be held in Beijing. Originally scheduled for 2021, due to the COVID-19 pandemic the conference has been postponed to August 14-19, 2022 and will be in hybrid format. For more details, see the conference website at http://ismp2022.csp.escience.cn.

ICCOPT 2022

The seventh International Conference on Continuous Optimization (ICCOPT) is scheduled for July 25-28, 2022 at the Lehigh University campus in Bethlehem, Pennsylvania, USA. It will be preceded by a Summer School on July 23-24. See https://iccopt2022.lehigh.edu for more information.

2 Books



Modern Nonconvex Nondifferentiable Optimization
By Ying Cui and Jong-Shi Pang
Publisher: SIAM
ISBN: 978-1-61197-673-1
Published: 2021
https://epubs.siam.org/doi/book/10.1137/1.9781611976748

ABOUT THE BOOK: After introducing the fundamentals of smooth optimization and convex optimization, the book describes the theory and practice of modern nonconvex, nondifferentiable optimization. It covers applications in Statistics, OR, and Machine Learning, and provides several examples and exercises for suitable use in advanced courses.

AUDIENCE: practitioners of Applied and Computational Mathematics, in particular Operations Research, Statistics, Computer Science (including Machine Learning), as well as engineers and economists. It is suited for advanced Optimization/OR courses.

3 Other Announcements

John von Neumann Theory Prize to A. Shapiro

Alexander Shapiro has been awarded the 2021 John von Neumann Theory Prize for his contributions to the theory and methods for Stochastic Programming. Congratulations Alexander!

Steele Prize to M. Goemans and D. Williamson

The 2022 Steele Prize is awarded to Michel Goemans and David Williamson for their Seminal Contribution to Research for their paper “Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming,” published in 1995 in the Journal of the ACM. Congratulations Michel and David!

Chair's Column

Katya Scheinberg, SIAG/OPT Chair
Cornell University, Ithaca, NY 18015-1582, USA
katyas@cornell.edu
https:/www.orie.cornell.edu/faculty-directory/katya-scheinberg

It is once again my privilege to write this column as a year has passed since our last issue. It has been another difficult year in terms of the pandemic and consequently in terms of professional events. SIAM Conference on Optimization (OP21), which was to be co-located with SIAM Annual Meeting (AN21), SIAM Conference on Discrete Mathematics (DM21), SIAM Conference on Applied Computational Discrete Algorithms (ACDA21) and the SIAM Conference on Control and Its Applications (CT21), was moved online, as were all the other co-located SIAM conferences. Although not ideal, the event overall was quite successful. The total number of attendees for all 5 conferences was more than 2000 and OP21 enjoyed a full program with 573 attendees, seven plenary talks, two minitutorials, and two SIAG prize talks.

Once again, our thanks go to Defeng Sun and Tamás Terlaky, the conference co-chairs, and the rest of the organizing committee, for the enormous work they put into the conference program. Current SIAG/OPT officers are now planning for the next installment of the SIAM Conference on Optimization – OP23. It is currently planned to be held in person in May of 2023, though the location will be finalized in early 2022. The conference co-chairs are Coralia Cartis (Oxford University) and Jeff Linderoth (University of Wisconsin).

The virtual business meeting of our SIAG was held during OP21 where several topics were discussed. Here is a brief summary:

- Initial plans for OP23 were announced.
- The new SIAG/OPT Fellows were welcomed, including Richard Byrd, David Gay and Defeng Sun (Class of 2020) and James Burke, Xiaojun Chen and Andreas Wächter (Class of 2021).
- New SIAM Engage community site was announced.
- Challenges and possible efforts in attracting more industry based members to SIAG/OPT and to SIAM in general were discussed.
- A proposal for the new SIAG/OPT Test of Time award was presented by the SIAG officers.

We will keep the community updated on any further developments regarding the last two items, via the email list and in the following issues of the Views and News. We hope these developments will generate further excitement among our members. SIAG/OPT keeps growing and is currently the third largest interest group within SIAM.

I wish you and your loved ones to stay healthy, optimistic, and productive in 2022 and beyond. Happy New Year.

Comments from the Editors

This edition of the "Views and News" comes with three great contributions. The first is by Mihai Anitescu and coauthors, who have built in less than 18 months, from scratch, a multi-period optimization solver, based on exascale architecture, aimed at Optimal Power Flow (OPF) applications with Julia, the rising language for computational applications.

The second article, by Andrés Gómez, describes the current state of the art in relaxations of quadratic optimization problems with *indicators*, i.e., binary variables that can "switch" on or off other variables.

Our third contribution is an obituary of the late Shabbir Ahmed, written by Santanu Dey, Jim Luedtke, Nick Sahinidis, and Weijun Xie.

Let us remind you that all issues of *Views and News* are available at the online archive: http://wiki.siam.org/siag-op/index.php/View_and_News.

The SIAG/OPT Views and News mailing list, where editors can be reached for feedback, is siagoptnews@lists.mcs.anl.gov. Suggestions for new issues, comments, and papers are always welcome.

Pietro Belotti
DEIB, Politecnico di Milano.
Email: pietro.belotti@polimi.it
Web: https://belotti.faculty.polimi.it

Somayeh Moazeni
School of Business, Stevens Institute of Technology.
Email: smoazeni@stevens.edu
Web: http://web.stevens.edu/facultyprofile/?id=2041