Jonathan Eckstein, Simge Küçükyavuz, and Suvrajeet Sen; it is a great honor to have been selected as recipients of this prize. This research was supported in part by National Science Foundation award CMMI-1634768.

REFERENCES

[1] X. Bao, A. Khajavirad, N.V. Sahinidis, and M. Tawarmalani. Global optimization of nonconvex problems with multilinear intermediates. *Mathematical Programming Computation*, 7(1):1–37, 2014.

[2] G. Cornuéjols. *Combinatorial Optimization: Packing and Covering*, volume 74 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, 2001.

[3] Y. Crama. Concave extensions for non-linear 0−1 maximization problems. *Mathematical Programming*, 61(1–3):53–60, 1993.

[4] A. Del Pia and A. Khajavirad. On decomposability of multilinear sets. *Mathematical Programming*, 2017. DOI: 10.1007/s10107-017-1158-z.

[5] A. Del Pia and A. Khajavirad. A polyhedral study of binary polynomial programs. *Mathematics of Operations Research*, 42(2):389–410, 2017.

[6] A. Del Pia and A. Khajavirad. The multilinear polytope for acyclic hypergraphs. *SIAM Journal on Optimization*, 28(2):1049–1076, 2018.

[7] A. Del Pia and A. Khajavirad. The running intersection relaxation of the multilinear polytope. *Optimization Online manuscript 2018/05/6618*, 2018.

[8] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, 1983.

[9] G.P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I—Convex underestimating problems. *Mathematical Programming*, 10(1):147–175, 1976.

[10] M. Padberg. The boolean quadric polytope: Some characteristics, facets and relatives. *Mathematical Programming*, 45(1–3):139–172, 1989.

Sparsity Matters

Robert J. Vanderbei
Dept. of Operations Research and Financial Engineering
Princeton University
rvdb@Princeton.EDU

I am deeply honored and grateful to have been awarded the 2017 Khachiyan Prize from the INFORMS Optimization Society. I really love optimization — one might say I'm an optimistic person. Over the past several decades I have had the great pleasure to collaborate with a number of similarly optimistic researchers and we've had a lot of fun working on a broad range of topics in optimization. In this article, I will focus on one particular issue that permeates much of the world of optimization, namely, the importance of understanding that "modeling matters" and, in particular, that exploiting sparsity in a problem's representation can be extremely beneficial.

1 Introduction

As is well-known in the optimization world, the worst case complexity of interior-point methods for linear programming is roughly $O(n^{3.5}L)$ where n denotes the number of variables and L denotes the

Gerald Brown, Robert Vanderbei, and David Morton

number of bits of data needed to encode the problem. It is also well-known that the standard variants of the simplex method take an exponential number of pivots in the worst case but that, in practice, the simplex method and interior-point methods have similar performance on average although one algorithm might beat the other by a significant amount on a particular problem.

This last "well-known fact" is perhaps the most important one — things vary greatly from one instance of a problem to another. Those of us in the field of optimization understand this quite well but users from outside the field often don't appreciate this subtlety. It is very common to hear someone say that their problem has tens of thousands of variables and therefore even an n^3 algorithm will be too slow to solve the problem. That is certainly true in some cases. But, it is not universally true and that's what this article is about. Before getting into details, I'd like to mention that this topic brings to mind a comment made by John Forrest at a conference back in the late 1980's. He said

> The simplex method is 200 times faster than ...
> ... the simplex method.

In this modern era infused with many different optimization algorithms, I would generalize John's statement to this:

> Any optimization algorithm is 200 times faster than ...
> ... that same algorithm.

2 What Can Go Wrong?

The practical performance issues that I'd like to discuss can be broken down into three broad categories:

- Sometimes the straightforward/default numerical *linear algebra* isn't the best way to solve a particular type of problem.
- Sometimes the obvious natural formulation of an optimization problem isn't very easy to solve but there is a *mathematically equivalent variant* that one can solve quickly.
- Sometimes the real-world problem to be solved isn't precisely defined and an *alternate formulation* might be vastly easier to solve.

In the following sections, I will discuss some examples of each of these scenarios. Some of these examples are well known in the optimization community but a few of them are a little more specific to particular interests of my own. I hope even these esoteric examples stimulate some interest among a broad reader base.

3 Numerical Linear Algebra

Problems with Dense Columns

Some linear programming problems have a constraint matrix that is mostly sparse but might have one or a few dense columns. For example, consider this LP:

$$\begin{array}{lrcrcl} \text{maximize} & c^T x & + & c_0\, x_0 & & \\ \text{subject to} & Ax & + & a\, x_0 & = & b \\ & & & x, x_0 & \geq & 0, \end{array}$$

where x is a high-dimensional vector, x_0 is just a single scalar variable, A is a *very sparse matrix*, but a is a *dense vector*.

There are practical real-world problems with this structure. And, for example, the "Big-M" method for getting an initial starting point for interior-point methods involves such a structure.

In the early days of interior-point methods (the mid 1980's), the computationally intensive part of these algorithms involved finding the "step directions" by solving the so-called "normal equations" for the dual step direction Δy:

$$\begin{bmatrix} A & a \end{bmatrix} \begin{bmatrix} D^2 & 0 \\ 0 & d^2 \end{bmatrix} \begin{bmatrix} A^T \\ a^T \end{bmatrix} \Delta y = ...$$

where D is a diagonal matrix and d is a scalar. Multiplying out, we get:

$$(AD^2A^T + d^2\, aa^T)\, \Delta y = ...$$

The matrix AD^2A^T is *sparse* but aa^T is *dense*. Hence, the sum is dense and solving the system of equations as formulated is highly inefficient. There are three ways to address this problem:

- Use the *Sherman-Morrison-Woodbury* formula to express solutions to the dense system in terms of solutions to the system without the dense (rank one) term. [4, 1, 7]

- Solve the *reduced KKT system* instead of the normal equations. [11]
- Re-express the problem using several variables in place of the single variable x_0. [9]

Splitting Dense Columns

A problem with these constraints...

$$\begin{bmatrix} A & a \end{bmatrix} \begin{bmatrix} x \\ x_0 \end{bmatrix} = b$$

can be reformulated in this equivalent but larger but also sparser fashion...

$$\left[\begin{array}{c|ccccc} & a_1 & & & & \\ & & a_2 & & & \\ A & & & a_3 & & \\ & & & & \ddots & \\ & & & & & a_m \\ \hline & 1 & -1 & & & \\ & & 1 & -1 & & \\ & & & \ddots & \ddots & \\ & & & & 1 & -1 \end{array}\right] \left[\begin{array}{c} x \\ \hline x_{0,1} \\ x_{0,2} \\ x_{0,3} \\ \vdots \\ x_{0,m} \end{array}\right] = \left[\begin{array}{c} b \\ \\ \\ \hline 0 \\ 0 \\ \vdots \\ 0 \end{array}\right]$$

For an optimizer based on solving the normal equations, this second form of the problem will solve *much faster* than the original form. Of course, most/all modern interior-point method solvers solve the reduced KKT system to avoid this issue. This *dense columns* example is a bit dated but it illustrates an important point:

> The number of constraints, m, and the number of variables, n, often have little to do with how long it takes to solve a problem.

The *sparsity* of the constraint matrix plays a huge role.

Exploiting Sparsity

Modern solvers are designed to exploit sparsity in the constraint matrix. I recently solved a generalized network flow problem using AMPL [3] with LOQO [10]. The problem had 195,503 variables and 123,571 constraints. Of course, being a (generalized) network flow problem the constraint matrix had only about two nonzeros per column. I was able to solve it in just 3.9 seconds on my laptop computer. For comparison, I solved a dense problem with 1/100-th as many variables and 1/100-th as many constraints. It took 59 seconds to solve. If we were to scale this up to the same size as the network flow problem, it would take about $59 \times 100^{3.5} = 590,000,000$ seconds to solve — in other words, about 18.7 years (and, of course, it wouldn't fit in the memory on my laptop computer).

4 Mathematically Equivalent Problems

Now let's consider an example illustrating how problems often have equivalent formulations that differ dramatically in how long it takes to solve them. The problem we will consider is the *Markowitz model* for portfolio selection.

In the Markowitz model, our vector x of decision variables is a stochastic vector (i.e., the elements are nonnegative and sum to one) representing the fraction of our portfolio to invest in various asset choices and the problem is to optimize some combination of risk and reward. The reward term is linear and can be denoted $r^T x$ whereas the risk term is quadratic and can be expressed in terms of a covariance matrix: $x^T \Sigma x$. If we let μ denote the weighting factor that controls the trade-off between these two objectives, we can express the Markowitz model like this:

$$\begin{array}{lrcl} \text{minimize} & \mu\, x^T \Sigma x - r^T x & & \\ \text{subject to} & e^T x & = & 1 \\ & x & \geq & 0. \end{array}$$

Here's an equivalent formulation of the problem. The matrix Σ is positive semidefinite and therefore can be expressed (and probably was defined) as a product $\Sigma = U^T U$. So, we can reexpress the problem like this:

$$\begin{array}{lrcrcl} \text{minimize} & \mu\, y^T y & - & r^T x & & \\ \text{subject to} & y & - & Ux & = & 0 \\ & & & e^T x & = & 1 \\ & & & x & \geq & 0. \end{array}$$

This formulation has more variables and more constraints. But, the quadratic term in the objective function is now a simple (sparse) $y^T y$ whereas the quadratic term in the original formulation was dense

$(x^T \Sigma x)$. This can make a big difference. Also, the matrix U does not have to be a square matrix — it might have many fewer rows than columns.

Here's a specific example. For a problem with 10,000 market assets and a covariance matrix based on data from 100 time periods in the past, the first problem has 10,000 variables and just one constraint. It solves in 1480 seconds. The second formulation has 10,100 variables and 101 constraints. It solves in 10.3 seconds — a speedup by a factor of 144.

5 Alternate Formulations

In the previous section, we discussed the fact that there are often equivalent formulations of a problem one of which might solve much faster than another. Sometimes there is some freedom in the definition of the problem and minor changes to the definition can lead to dramatic speedups. To illustrate such benefits of *alternate formulations* let us consider various expressions of the *compressed sensing* problem.

Compressed sensing

The goal of compressed sensing is to recover a sparse signal from a small number of measurements. Let $x^0 = (x_1^0, \ldots, x_n^0)^T \in \mathbb{R}^n$ denote a signal to be recovered. Here, we assume that n is large and that the signal vector x^0 is sparse.

Let A be a given (or chosen) $m \times n$ matrix with $m \ll n$. The *compressed sensing problem* is to recover x^0 assuming only that we know $y = Ax^0$ and that x^0 is sparse.

Since x^0 is a sparse vector, one can hope that it is the sparsest solution to the underdetermined linear system and therefore can be recovered from y by solving

$$(\mathrm{P}_0) \qquad \min_x \|x\|_0 \quad \text{subject to } Ax = y,$$

where $\|x\|_0$ denotes the 0-pseudo-norm of x:

$$\|x\|_0 = \#\{i : x_i \neq 0\}.$$

As is well-known, this problem is NP-hard due to the nonconvexity of the 0-pseudo-norm.

To make the problem more tractible, it is common to replace the 0-pseudo-norm with the 1-norm: $\|x\|_1 = \sum_j |x_j|$. This new problem is called the *basis pursuit* problem:

$$(\mathrm{P}_1) \qquad \min_x \|x\|_1 \quad \text{subject to } Ax = y.$$

This problem can be converted to a linear programming problem using the standard tricks for handling absolute values in minimization problems. There is an extensive literature that studies the probability as a function of m, n, and the choice of A that a solution to the basis pursuit problem is actually a solution to the compressed sensing problem.

Kronecker Compressed Sensing

As formulated, the compressed sensing problem involves a signal that is a vector. In some applications, it is more natural to assume that the signal is a matrix or even a higher dimensional tensor. Let's consider a matrix signal.

Given a sparse matrix signal $X^0 \in \mathbb{R}^{n_1 \times n_2}$, we can use two sensing matrices $A \in \mathbb{R}^{m_1 \times n_1}$ and $B \in \mathbb{R}^{m_2 \times n_2}$ and try to recover X^0 from knowledge of $Y = AX^0B^T$ by solving the *Kronecker compressed sensing* problem:

$$(\mathrm{P}_2) \qquad \hat{X} = \operatorname{argmin} \|X\|_1 \quad \text{subject to } AXB^T = Y.$$

Here, of course, $\|X\|_1$ is the sum of the absolute values of all the entries of the matrix X. As with the basis pursuit problem, this problem can also be formulated as a linear programming problem.

When the signal is multidimensional, Kronecker compressed sensing is more natural than classical vector compressed sensing. Also, as we will explain shortly, the linear programming problem for Kronecker compressed sensing benefits from sparsity that is not present in the vector-based formulation. Hence, for a vector problem and a matrix problem of the same size (i.e., the same number of elements in the signal), one would expect the matrix problem to solve more quickly.

Kroneckerizing Vector Problems

Sometimes, even when facing vector signals, it is beneficial to use Kronecker compressed sensing due to its added computational efficiency.

More specifically, even though the target signal is a vector $x^0 \in \mathbb{R}^n$, if we assume that n can be

factored into $n_1 \times n_2$, then we can first reshape x^0 into a matrix $X^0 \in \mathbb{R}^{n_1 \times n_2}$ by putting successive length n_1 sub-vectors of x^0 into columns of X^0. We then multiply the matrix signal X^0 on both the left and the right by sensing matrices A and B to get a compressed matrix signal Y^0. We will show that we are able to solve this Kronecker compressed sensing problem much more efficiently than the corresponding vector compressed sensing problem.

Vectorizing the Kroneckerization

In the Kronecker problem the variables are naturally represented as a matrix. But, an LP solver expects to see a vector of variables, not a matrix. So, we need to rewrite the Kronecker problem in vector form.

The simple/natural/naive vectorization can be described as follows. Let $x = \text{vec}(X)$ and $y = \text{vec}(Y)$, where, as usual, the vec$(\cdot)$ operator takes a matrix and concatenates its elements column-by-column to build one large column-vector containing all the elements of the matrix.

In terms of x and y, problem (P_2) can be rewritten as an equivalent *vector compressed sensing* problem:

$$\text{vec}(\hat{X}) = \text{argmin}\|x\|_1 \;\text{ subject to }\; Ux = y,$$

where U is given by the $(m_1 m_2) \times (n_1 n_2)$ Kronecker product of A and B:

$$U = B \otimes A = \begin{bmatrix} Ab_{11} & \cdots & Ab_{1n_2} \\ \vdots & \ddots & \vdots \\ Ab_{m_2 1} & \cdots & Ab_{m_2 n_2} \end{bmatrix}.$$

The matrix U is fully dense. *This is bad.*

Sparsifying the Constraint Matrix

The key to an *efficient* algorithm for solving the linear programming problem associated with the Kronecker sensing problem lies in noting that the dense matrix U can be factored into a product of two sparse matrices:

$$\begin{aligned} U &= \begin{bmatrix} Ab_{11} & \cdots & Ab_{1n_2} \\ \vdots & \ddots & \vdots \\ Ab_{m_2 1} & \cdots & Ab_{m_2 n_2} \end{bmatrix} \\ &= \begin{bmatrix} A & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & A \end{bmatrix} \begin{bmatrix} b_{11}I & \cdots & b_{1n_2}I \\ \vdots & \ddots & \vdots \\ b_{m_2 1}I & \cdots & b_{m_2 n_2}I \end{bmatrix} \\ &=: VW. \end{aligned}$$

Here, I denotes an $n_1 \times n_1$ identity matrix, 0 denotes an $m_1 \times n_1$ zero matrix, and V is a block-diagonal matrix with A on the diagonal blocks.

Exploiting the Sparsification

Assuming that A and B are dense matrices (as they generally are in Kronecker compressed sensing), the matrix U is usually completely dense. But, it is a product of two sparse matrices: V and W. We can exploit this sparse factorization.

Let's introduce some new variables, call them z, and rewrite the constraints like this:

$$\begin{array}{rcll} z & - & Wx & = 0 \\ Vz & & & = y. \end{array}$$

Using the common trick of expressing a free variable as the difference of two nonnegative variables and the absolute value of that free variable as the sum of the two nonnegative variables, we can convert the problem to a linear program:

$$\begin{array}{lrcll} \min\limits_{x^+,x^-} & & & \mathbf{1}^T(x^+ + x^-) & \\ \text{subject to} & z & - & W(x^+ - x^-) & = 0 \\ & Vz & & & = y \\ & & & x^+, x^- & \geq 0. \end{array}$$

This formulation has more variables and more constraints. But, the constraint matrix is *sparse*. And as we've discussed already, for linear programming, sparsity of the constraint matrix is the key to algorithm efficiency.

Numerical Results

The numerical results I'll present here were first published in [12].

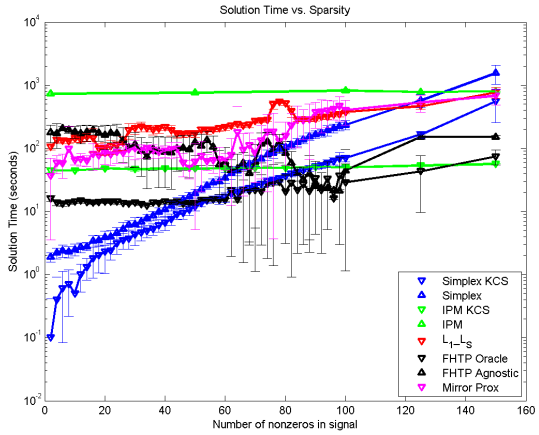
For the *vector* sensor, we generated random problems using $m = 1{,}122 = 33 \times 34$ and $n = 20{,}022 = 141 \times 142$. We varied the number of nonzeros k in signal x^0 from 2 to 150. We solved the straightforward linear programming formulations of these instances

using my interior-point solver LOQO [10]. We also solved a large number of instances of the problem using the parametric simplex method as described in [8].

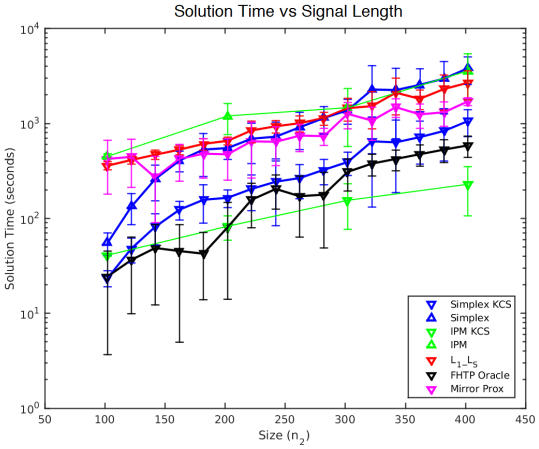
We followed a similar plan for the *Kronecker* sensor. For these problems, we used $m_1 = 33$, $m_2 = 34$, $n_1 = 141$, $n_2 = 142$, and various values of k. Again, the straightforward linear programming problems were solved by LOQO and the parametrically formulated versions were solved by a custom developed parametric simplex method.

For the Kronecker sensing problems, the matrices A and B were generated so that their elements are independent standard Gaussian random variables. For the vector sensing problems, the corresponding matrix $\mathbf{U}$ was used.

We also ran some publicly-available, state-of-the-art codes: *ℓ_1_ℓ_s* [6], *FHTP* [2], and *Mirror Prox* [5].



$m = 1{,}222$, $n = 20{,}022$. Error bars represent one standard deviation.



$m = 1{,}122$, $k = 100$, $n = 141 \times n_2$.

Conclusions

The interior-point solver (LOQO) applied to the Kronecker sensing problem is uniformly faster than both ℓ_1_ℓ_s and the interior-point solver applied to the vector problem (the three horizontal lines in the plot).

For very sparse problems, the parametric simplex method is best. In particular, for $k \leq 70$, the parametric simplex method applied to the Kronecker sensing problem is the fastest method. It can be two or three orders of magnitude faster than ℓ_1_ℓ_s.

But, as explained earlier, the Kronecker sensing problem involves changing the underlying problem being solved. If one is required to stick with the vector problem, then it too is the best method for $k \leq 80$ after which the ℓ_1_ℓ_s method wins.

Instructions for downloading and running the various codes/algorithms described herein can be found at:

`http://www.orfe.princeton.edu/~rvdb/tex/CTS/kronecker_sim.html`

REFERENCES

[1] I.C. Choi, C.L. Monma, and D.F. Shanno. Further development of a primal-dual interior point method. *ORSA J. on Computing*, 2:304–311, 1990.

[2] S. Foucart. Hard thresholding pursuit: An algorithm for compressive sensing. *SIAM Journal on Numerical Analysis*, 49(6):2543–2563, December 2011.

[3] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, Belmont CA, 1993.

[4] P.E. Gill, W. Murray, and M.A. Saunders. A single-phase dual barrier method for linear programming. Technical Report SOL 88-10, Systems Optimization Laboratory, Stanford University, Stanford, CA, 1988.

[5] A. Juditsky, F. Kılınç Karzan, and A. Nemirovski. Randomized first order algorithms with applications to ℓ_1-minimization. *Mathematical Programming*, 142(1):269–310, Dec 2013.

[6] S.J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. An interior-point method for large-scale ℓ_1-regularized least squares. *IEEE Transactions on Selected Topics in Signal Processing*, 1(4):606–617, December 2007.

[7] I.J. Lustig, R.E. Marsten, and D.F. Shanno. Computational experience with a primal-dual interior point method for linear programming. Technical Report SOR 89-17, Department of Civil Engineering and Operations Research, Princeton University, October 1989.

[8] B. Rudloff, F. Ulus, and R.J. Vanderbei. A parametric simplex algorithm for linear vector optimization problems. *Mathematical Programming, Series A*, 163(1):213–242, 2017.

[9] R.J. Vanderbei. Splitting dense columns in sparse linear systems. *Lin. Alg. and Appl.*, 152:107–117, 1991.

[10] R.J. Vanderbei. LOQO user's manual — version 3.10. *Optimization Methods and Software*, 12:485–514, 1999.

[11] R.J. Vanderbei and T.J. Carpenter. Symmetric indefinite systems for interior-point methods. *Mathematical Programming*, 58:1–32, 1993.

[12] R.J. Vanderbei, K. Lin, H. Liu, and L. Wang. Revisiting Compressed Sensing: Exploiting the Efficiency of Simplex and Sparsification Methods. *Mathematical Programming, Series C*, 8:253–269, 2016

Mountains of Optimization indeed! (Photo by Thiago Serra)

2018 IOS Conference Report

Alexandra M. Newman
Department of Mechanical Engineering
Colorado School of Mines
anewman@mines.edu

The 2018 edition of the INFORMS Optimization Society conference was held in Denver, Colorado on March 23–25, 2018. Over 200 participants descended on the mile-high city, contributing to nearly 70 sessions, three tutorials, and seven plenaries.

The theme of the conference was "Mountains of Optimization," and mountains there were. Sessions on optimization under uncertainty and nonlinear optimization were among the most copious, but attendees were also treated to topics on discrete optimization, optimization in machine learning and statistics, network optimization, and software and implementation, inter alia.

The seven plenaries featured:

- Shabbir Ahmed: "Value of Multi-Stage Stochastic Optimization in Power Systems Operations"
- Marcos Goycoolea: "Large-scale Open Pit Mine Production-Scheduling"
- Moritz Hardt: "Non-convex non-optimization"
- Illya Hicks: "Discrete Optimization and Network Analysis"
- Karla Hoffman: "Hybrid optimization algorithms to solve real-world problems" (based on material from this year's Franz Edelman Award winning team)
- John Hooker: "What Decision Diagrams Can Do for You"
- Sven Leyffer: "A Globally Convergent Cutting-Plane Method for Simulation-Based Optimization with Integer Constraints"

The three tutorials consisted of:

- Bob Fourer: "A Guide to Identifying Good Near-Optimal Formulations for Hard Mixed-Integer Programs"
- Ed Klotz: "Performance Tuning For CPLEX's Spatial Branch-and-Bound Solver For Global Nonconvex Mixed Integer Quadratic Programs"
- Warren Powell: "A Unified Modeling and Algorithmic Framework for Optimization under Uncertainty"

All plenary and tutorial speakers were kind enough to supply their slides, which are posted on the conference website at: `http://orwe-conference.mines.edu/info.html`.

A special "thank you" must be extended to the co-organizers of the conference, Stephen Billups and