For any graph G, a matching J in G means a subset J of the edges such that each node is hit by at most one edge of J. Let $P(G)$ be the polytope whose vertices are the 0,1 vectors of the matchings in G. The Konig-Egervary Theorem [1931] about the optimum assignment problem, (the prototype of matroid-polytope intersection), implies: When G is bipartite, the convex hull of the 0,1 vectors of the matchings in G is:

$$P(G) = x \geq 0 : \text{for each node } v \text{ of } G, \sum(x_e : \text{edges } e \text{ hitting node } v) \leq 1. \tag{1}$$

Indeed this system of inequalities is "totally dual integral (TDI)". I think that the K-E theory is the first instance of linear programming duality. The way they say the K-E Theorem is: The maximum total of given weights c_e of the edges e of a matching in a bipartite G = the min sum Σy_v of integers $y_v \geq 0$ assigned to the nodes v of G so that for each edge e of G, $y_{v_1} + y_{v2} \geq c_e$ where v_1 and v_2 are the nodes hit by edge e.

This follows from the above inequality system being TDI, and conversely this minmax equality implies that the inequality system is TDI. However in 1931 the LP theory connecting the two were not known. Much of combinatorial optimization in the 1950s by Alan Hoffman, David Gale, Harold Kuhn, Ray Fulkerson, George Dantzig, and others, is closely related to K-E. I will indicate the story of how my good luck under Alan Goldman was to generalize K-E to non-bipartite graphs, and also to "matroid intersections".

NP is the set of predicates which for the instances that are true there is an easy proof. coNP is the set of predicates which for the instances that are not true there is an easy proof that they are not true. P is the set of predicates for which there is a "good algorithm" to decide whether the predicate is true or false. In order to give the definitions some manageable mathematical meaning "good" and "easy" means polynomial time relative to the size of writing down the instance.

NP and P have origins in "the marriage theorem", a corollary of the bipartite K-E matching theory: A matchmaker has as clients the parents of some boys and some girls where some boy-girl pairs love each other. The matchmaker must find a marriage of all the girls to distinct boys they love or else prove to the parents that it is not possible. The input to this marriage problem is a bipartite graph G with boy nodes, girl nodes, and edges between them representing love. A possible legal marriage of some of the girls to some of the boys is represent ed by a subset M of the edges of G, which is a matching. The matchmaker's problem is to find a matching which hits all the girl nodes Or else prove to the parents that there is none. The Marriage Theorem: Either:

$$A(G) = [\text{There is a way for the girls to marry distinct boys they love}] \text{ or else} \tag{2}$$
$$B(G) = [\text{There is a subset } S \text{ of the girls which is bigger than the set of boys who someone in } S \text{ loves}], \tag{3}$$

Not both.

In other words the Marriage Theorem is: $A(G) = \text{not}B(G)$. And so$A(G) \in \text{NP} \bigcap \text{coNP}$, since both $A(G)$ and $B(G)$ are clearly in NP. From Matt Baker: https://mattbaker.blog/2014/06/25/the-mathematics-of-marriage: A gorgeous application of the marriage theorem is the following magic trick: Deal a deck of cards into 13 piles of 4 cards each. Select one card from each pile so that no value (Ace through King) is repeated. This can always been done, no matter how the cards were originally dealt! Matt deduces this from the Marriage Theorem said in a way which pure mathies love: The girls can marry distinct boys they love if and only for every subset S of the girls, $|S| \geq$ the number of boys which S to get her loves.

While I was trying to assign radio frequencies to airports, my reading this way of stating, and an algorithmically inefficient proof of the Marriage Theorem, blew my mind with puzzlement. Is looking at every subset of girls easier than looking at every matching? Pure mathies love proofs which are bad as algorithms and convey the impression that there is no easy way to actually get what the theorem says exists. The proof in the book called "Proofs From The Book" is like that. There is a simpler proof of the Marriage Thm and K-E which is a polynomial time algorithm for finding what the theorems say exist. Thinking about all of this in 1960, under Alan Goldman's supervision, it occurred to me that maybe a general predicate is in P whenever it is in NP$\bigcap$coNP.

He liked that, and let me not worry so much about airports. The convex hull of any finite set Q of vectors in the space of vectors, coordinatized by the edge-set E of graph G, is a polytope, the bounded solution-set of a finite set L of linear inequalities. Optimizing cx over members of Q is the same as optimizing cx subject to L. Normally for an easily described finite set Q of points in R^E , the number of inequalities needed is exponential relative to $|E|$. An arrangement of the girls and boys at a round banquet table so that each is, on each side, next to someone loved, is the same as a TSP tour in the graph G.

Suppose we want a TSP tour which optimizes the sum of love-weights on edges in C. The TSP, in a yes-no form, is given $t =$ (a graph G, a love-weight c_e for each edge e, a tour H in G), then where $A(t)$ is the predicate "H is not the largest total love-weight tour in G", is $A(t)$ true or false? This predicate $A(t)$ is clearly in NP.

Is it in coNP? TSP is represented by the set Q of the 0-1 vectors x of the TSP tours in G and a linear function cx to be optimized over Q. Is there an NP set L of linear inequalities such that optimizing cx over solutions to L is the same as optimizing cx over Q? In 1954, George Dantzig, with Ray Fulkerson and Selmer Johnson, solved a particular 48-city traveling salesman problem by using linear programming. Their linear inequality system L' , which is satisfied by a 0,1 vector x if and only if x is the vector of a TSP tour, is NP, though it is exponentially large.

Unfortunately the solution-set of L' has some extreme points which are fractional rather than TSP tour vectors, and so optimizing subject to L' might not be by a TSP tour. Their work was a motivation for Ralph Gomory in 1958 to introduce cutting plane methods for integer linear programming. There was no mention of 'easy', meaning polynomially bounded time. (Until my student Vasek Chatal, there was no stated theorem other than that the algorithm is finite.) The fact that cuts and lp optimizations do not commute created skepticism.

What is the "cut-depth" of a cutting plane method? The 1954 TSP paper of Dantzig et al prompted me in 1961 to notice the simple result that Theorem. If there is an NP description of the set L of inequalities describing the convex hull of finite point-set Q, as well as an NP description of the members of Q, then it does not matter that L is exponentially large in order to have the predicate A(x) = [point x of Q is optimum over Q] in NP $\bigcap$ coNP.

And maybe that puts "optimize over Q" algorithmically into P. It seemed plausible that if there is an easy description of Q then there might be an easy description of a linear system, L, describing the convex hull of Q. That is an important step toward an easy general algorithm for optimizing over Q. I was never able to find an NP description of a linear system L whose solution-set is the convex hull of the vectors of the TSP tours in a complete graph, G, and so I conjectured in 1966 that TSP is not in P, and hence that $NP \neq P$.

The hypothesis $NP \neq P$ is now usefully presumed. It is used all the time and its failure would be cataclysmic to all the great work of math programming. Finding a mathematical proof has defied enormous attention for over 50 years.

Having such a proof does not really matter since it is true as long as we do not know a polytime algorithm for an NP complete problem. Possibly there is no mathematical proof of $NP \neq P$. It follows from work of Godel and Turing that most of true math does not have any mathematical proof. If I had been asked in 1966 I might have said that "$NP \neq P$ is probably simple to prove by sorting out the formalities, but that would not settle whether TSP is in P".

However Dick Karp showed in 1972 that TSP is NP complete, and so $NP \neq P$ would indeed determine that TSP is not in P. Throughout the 1960s , while looking for easy algorithms of course I found some polytime reductions between problems, but I never dreamed that there exist NP-complete problems. It is now a cosmic tragedy that there being so many NP-complete problems has even given NP a bad name, when NP itself is a wonderfully positive thing.

During the 1960s, as part of a search for TSP in P, I did find some NP sets Q of points to have of convex-hull-determining NP linear systems, L, and from that some P algorithms – like matchings and matroid intersections. They were all hoped to be ingredients for solving TSP. In 1961, my wonderful mentor Alan Goldman arranged through his wonderful Princeton mentor, Professor Tucker, thesis supervisor of Nash's equilibria and much else, for me to be a junior participant in a Summer long workshop at the RAND Corporation, across from Muscle Beach, down the street from Hollywood.

Seemingly every known combinatorist participated. The National Bureau of Standards was being careful, possibly an after effect of some McCarthyism at NBS, and so it refused Alan's request that I attend the workshop of this private government contractor, RAND. Alan did not disapprove of my idea that I resign from NBS. At the end of the workshop, NBS hired me back with a very good raise. At the workshop I needed a successful example of the NP idea, such as the non-bipartite matching problem.

The day before my scheduled lecture to the eminences I still did not have it. Then Eureka, I did! You shrink blossoms! So the lecture was a success. It attracted discussion and lots of good attention. Professor Tucker offered me a visiting research position at Princeton, chairing his Combinatorics and Games Seminar. And so my first glimpse of mathematical heaven was at the notorious RAND Corporation – as exciting as the RAND Corporation's Daniel Ellsberg revealing the Pentagon Papers. Alan's wife Cynthia usually drove him to and from work, but occasionally I would.

During one of these inspiring drives Alan reminded me that there is an easy algorithm for finding a min cost spanning tree in a graph. He asked me "What does that have to do with linear programming?" This led to my favorite theorem, the matroid-polytope intersection theorem. A matroid M is an abstraction of the linearly independent subsets of the set E of columns of a matrix, M: For any subset S of E, every (inclusion) maximal M-independent subset J of S is the same size, called $rM(S)$, the M-rank of set S.

Since the forest edge-sets of a graph are the independent sets of a matroid which is given by linear independence in a matrix over the field where $1+1=0$, and such that each column has two 1's, we see that matroids do have an interesting variety of structure.

The popular 'greedy algorithm', for finding an optimum total weight spanning tree of a graph, generalizes in two ways: The Greedy Theorem. For any matroid M on the set E and any weighting ce of its elements e, "the greedy algorithm" finds the 0,1 vector x of an independent set of M which maximizes cx in the polytope $P(M) = x \geq 0 : foreverysubsetSofE, \sum xe : einS \leq rM(S)$.

Corollary. The vertices of $P(M)$ are the 0,1 vectors of the independent sets of M. (You should be ashamed if, instead of this, you teach only the Kruskal spanning tree version, presented by Boruvka in 1931.) Suppose we have any two matroids M_1 and M_2 on set E.

In general the 'independence system', $M_1 \bigcap M_2$, consisting of sets independent in both M_1 and M_2 is not a matroid. We want to describe the convex hull, say $P(M_1 \bigcap M_2)$, of the vectors of sets in $M_1 \bigcap M_2$. Every common vertex of any two polyhedra is a vertex of their intersection. In general however, there will be other vertices as well. It is rare for two polyhedra to fit together so neatly that the only vertices of the intersection are the common vertices. The Amazing Matroid Polytope Intersection Theorem: $P(M_1 \bigcap M_2) = P(M_1) \bigcap P(M_2)$.

The dual optimum is integers when the c of max cx over$P(M_1) \bigcap P(M_2)$ is integers. When the c is all ones we have the 'Cardinality Matroid Intersection Theorem': max $\{\ \left|J\right| : J \varepsilon M_1 \bigcap M_2\ \}$ = min $\{\ rM_1(S) + rM_2(E - S) : SE\ \}$. $\subseteq$ In other words, the following linear system is Totally Dual Integral (TDI): [x≥0; for every subset S of E, $\sum xe : einS \leq rM_1(S)$ and $\sum xe : einS \leq rM_22(S)$].

Proved by proving that a polynomial time algorithm more complicated than the greedy algorithm maximizes any cx over $P(M_1) \bigcap P(M_2)$ by a common independen t set of M_1 and M_2 , and the dual optimum is integers if the c of the primal objective cx is integers. The Konig-Egervary (K-E) Theorem [1931] about the optimum assignment problem is a special case where each matroid, M_1 and M_2 , is a rainbow matroid, which means a subset J is independent in matroid Mk when the members of J have different Mk -colors, where for each $k = 1or2$, the set E of elements is Mk -colored, that is partitioned, in any way.

A novel instance of matroid intersection was the optimum (single) branching problem: given a direct graph G with a specified root node r and with weights ce on the edges e, find a directed spanning tree T rooted at r such that Σce over edges e in T is max (or min).

A set J of edges is in M_1 when the edges are directed toward distinct nodes different from r. A set J of edges is in M_2 when it is the edge-set of a forest in G. The common bases of these matroids are the directed spanning trees rooted at node r.

If the intersection of three matroids were nice then we could solve the TSP problem of finding an optimum spanning directed path in G from node r to node s by taking matroid M_3 to be where set J is independent when the edges of J are directed away from distinct nodes different from node s.

I found the optimum branching solution by applying this obvious application of matroid intersections. A direct description of the algorithm is quite elegant. I had a hunch that it would be more popular than general matroid intersections, and it has been. Karp and Fulkerson each presented different proofs of the algorithm, Bob Tarjan refined the running time with data structures, and applications have been presented.

I felt as did the algebraists of olden times who liked to hide their methods. As part of 1960s counterculture, clearly the need have 'results' to get worldly math position was a social illness. (Indeed, most normal people leave math for this reason.) The Optimum Branching Systems problem (OBS) is Given a digraph G, specified root nodes $r(i)$, and a cost for each edge of G, find a least cost collection of edge-disjoint directed spanning trees in G, rooted respectively at the nodes $r(i)$, i.e., $r(i)$-branchings in G.

OBS easily reduces to where all the root nodes,$r(i)$, are the same. And so OBS can be more conveniently stated as Given a digraph G, a specified root node r, an integer $k > 0$, and a cost for each edge, find k edge-disjoint directed spanning trees all rooted at node r which minimize the total cost of the edges used.

The well known, widely treated, min cost network flow problem is a special case of OBS. However OBS does not reduce to it. The min cost network flow problem is Given a digraph G, a source node r in G, a sink node t in G, an integer k, and edge costs, find k mutually edge-disjoint directed paths from r to t which minimizes the total cost of the edges used. (An integer "capacity be on edge e" is simulated by be edges in parallel.) Reduce the k flow problem to a k branching problem by adjoining cost-free edges from t to each node of G except r. (This reduction takes the shortest path problem to a single branching problem.) There is a polynomial time algorithm for OBS, also as an intersection of two matroids.

However, it is considerably more complicated than for a single optimum branching. It requires additional theorems and algorithms. Matroid intersection is the only approach known for solving OBS. Unsuccessful attempts have been made to describe nicely a direct algorithm for OBS. Curiously, unlike the optimum single branching problem, the simplest way known to describe an algorithm for OBS is by matroid intersection for general matroids, but more than that is needed.

Matroid algorithms use subroutine sources of the matroids, M, which say when a set is independent in M. For OBS, matroid M_1 is simple: a set J of edges is independent in M_1 when none of them are directed toward root node r and at most k of them are directed toward any other node. Matroid M_2 is where a set J of edges is independent when it can be partitioned into the edge sets of at most k edge-disjoint forests in graph G.

It is not obvious that M_2 a matroid. And a polytime algorithm is not obvious for recognizing whether a set J can be partitioned into at most k edge-disjoint forests in graph G. Besides the theorem that the edge-sets of forests in G are the independent sets of a matroid, we use the deeper Matroid Partition Theorem that For any matroid M, the sets partionable into k independent sets of M are the independent sets of a matroid M_2 , and Either a set J of elements of M is partionable this way or else: there is a subset S of J such that S is bigger than $krM(S)$.

(Of course both can not be true.) We use a non-obvious polytime matroid partitioning algorithm which proves this Matroid Partitioning Theorem. That feeds the optimum matroid intersection algorithm for these matroids, M_1 and M_2 .

All of that gives us a good algorithm for finding an optimum set J of edges which (1) can be partitioned into k sets where each has no edges into r , and one edge into each other node; (2) can be partitioned into k spanning trees of graph G. That does not mean that there is a partition of J into k sets which satisfies both (1) and (2).

Well, there is. We will see that as another big theorem. Rainbow Spanning Trees. Given a graph G and any coloring of the edge-set E of G, a rainbow forest in G is a forest of G which uses any color at most once. (*) Find if possible a set J of edges of G which can be partitioned into k edge-disjoint rainbow spanning trees of G. This is a known topic in graph theory. (**) An r-branching in a directed G is the special case of rainbow spanning tree where each color is the set of edges into a node of G different from root r. For $k = 1$, (*), finding an optimum edge-set J which is a rainbow spanning tree of G, whenever there one, is polynomial time, using matroid intersection. For $k > 1$, (*) might be generally hard.

It is probably NP complete. However, God is sometimes kind. In the case of (**) we can do (*). This is a nifty theorem about edge-disjoint branchings in directed graphs. We do not know a generalization to matroids, or to different ways of coloring the edges. It is interesting to compare the post-optimization of OBS with the post-optimiz ation of mincost flow.

In well known algorithms for the mincost network flow problem (which in fact can be easily reduced to the K-E optimum assignment problem) one finds an optimum edge-set, say J, which has k edges directed into the sink node t, and k edges directed out of the source node r, and, for every other node v, J has the same number direct out of v as directed into v, and J has no directed cycles. If you are dispatching k drivers you do not simply tell them to use only the roads in J. You partition J into k directed paths from r to t. That is so easy to do that we sometimes forget to do it.

That is analogous to solving OBS by finding an optimum set J of edges which satisfies (1) can be partitioned into k sets such that each has none into r and one into each other node; (2) can be partitioned into k spanning trees of graph G.

It is not quite as simple as for network flows to get a partition which satisfies both (1) and (2). But we can do it. This can be seen as a corollary of an admirable generalization of the maxflow-m incut theorem: It is not quite as simple as for network flows to get a partition which satisfies both (1) and (2). But we can do it. This can be seen as a corollary of an admirable generalization of the maxflow-m incut theorem with a polynomial time algorithmic proof:

The Disjoint Branching Theorem. The max number of edge-disjoint spanning directed trees in G, rooted at node r, equals the min size of a set C of edges in G which separates r from some non-empty set S of other nodes of G. (This means C is the set of edges directed into S from the node-complement of S which contains r. It is called a cut.) Compare with the maxflow-mincut theorem: The max number of edge-disjoint directed paths from node r to node t in network G equals the min size of a set C of edges in G which separate node r from node t.

Corollary of The Disjoint Branching Theorem. If (and of course only if) a set J of edges in directed graph G (2) can be partitioned into k spanning trees of graph G, and (1) can be partitioned into k sets such that each set has none directed into node r and one into each other node, then J can be partitioned into k spanning directed trees rooted at node r.

The various ingredients for solving OBS are in books on combinatorial optimizat ion, but I'm still trying get people to do a good computer implementation. It is clearly possible, but a bit complicated. There is an enormous amount on refinement and implementation of "the blossom algorithm" for optimum matchings but none for optimum matroid intersections. I suppose that the subroutine oracle nature of matroid intersections is scary.