A Tour of Discrete Optimization

Ray Fulkerson's Impact: Past, Present, and Future

Karla Hoffman[1] and Ted Ralphs[2]

[1]Department of Systems Engineering and Operations Research, George Mason University
[2]COR@L Lab, Department of Industrial and Systems Engineering, Lehigh University

Fulkerson@100, Cornell University, 21 September 2024

Outline

Outline

Fulkerson and the "Computation" Revolution

- It is difficult to appreciate just how revolutionary certain ideas were *in their time*.
- There is no question that Fulkerson is viewed today as a pioneer.
- To fully appreciate his impact on the development of integer programming, it helps to put his early contributions in historical context.
- We focus on the ground-breaking paper of Dantzig, Fulkerson, and Johnson [1954].
- This paper was among a handful that ushered in a new era of computational optimization that we are essentially still living in.

Discrete/Combinatorial Optimization in the 40s and 50s

A few key quotes from Pulleyblank [2012]'s illuminating retrospective of the times paints a vivid picture.

> *"At this time, people were generally satisfied with algorithms whose running times could be proven to be finite..."*
>
> *"...many people in the operations research community considered a problem to be 'solved' if it could be formulated as an integer programming problem..."*
>
> *"the combinatorics community was not very interested in algorithms..."*
>
> *"Herb Ryser [noted] that there were two...types of problems...in combinatorics...: existence problems and enumeration problems."*

The Golden Age

DFJ's work took place at RAND, which was the undisputed center of research in optimization at the time. Work done there in in the '40s and '50s included:

- Dantzig [1948]: linear programming and the simplex algorithm.
- Robinson [1949]: "Hamiltonian games" (apparently coined the term "traveling salesman problem").
- Bellman [1954]: Markov decision processes and dynamic programming.
- Heller [1953] and Kuhn [1955]: foundations of polyhedral theory.
- **Dantzig, Fulkerson, and Johnson [1954]: foundations of computational MILP.**
- **Ford and Fulkerson [1956]: the maximum flow problem**.
- **Dantzig, Ford, and Fulkerson [1956]: primal-dual algorithm for LP**.
- Markowitz and Manne [1957]: early pre-cursor of branch-and-cut inspired by DFJ (coined the term "cutting plane").
- **Ford and Fulkerson [1958]: first column-generation algorithm.**
- Dantzig and Wolfe [1960]: column-generation for LP.

The Birth of "Computation"

- DFJ (implicitly) introduced the notion of "computation" and "practical algorithm" as we know it today.
- The essence of a "computational" method is that it is dynamically tailored to solve a specific problem/instance as efficiently as possible.
- This was essentially a new concept at the time.
- DFJ's method laid out core elements on which state-of-art algorithms still depend today.

1. Dynamically generating valid inequalities: *cutting-plane method*.
2. Sparsifying the graph by fixing variable values: *reduced-cost fixing*.
3. Use of the LP relaxation (strengthened by the above techniques) to derive a "certificate" of optimality: the *dual proof*.

Enter Duality

- All of the crucial elements of DFJ's method are "dual" in nature.
- Many of Fulkerson's contributions involved some duality-based theory and this is his legacy in a nutshell.
- Although their algorithm was "primal" at its core, they realized that dual arguments are necessary to construct a formal proof.
- Importantly, they proposed constructing a "dual proof" dynamically, with the implicit goal of making it as small as possible.

The Traveling Salesman Problem

- The traveling salesman/salesperson problem (TSP) is now well-known as perhaps the most well-studied combinatorial optimization problem.
- This may be in part due to DFJ's choice to study it in this paper.
- This problem has always tickled the imagination of the general public and the paper actually made headlines in the press at the time.
- Consider the following tour on a five-node example from DFJ.

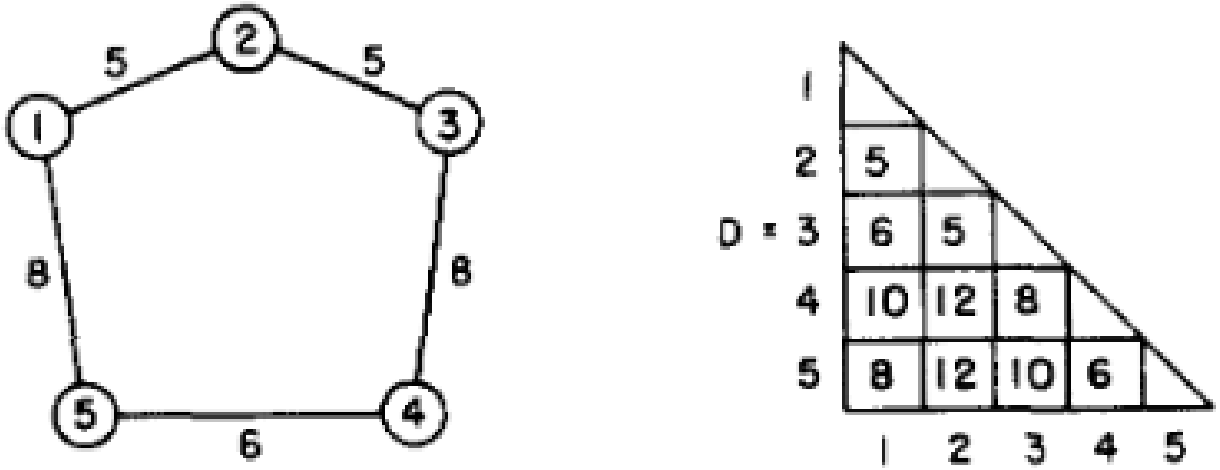


FIGURE 2

- Is it optimal?

Dual Prices

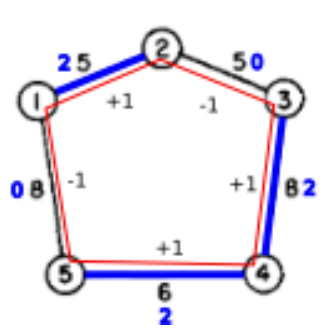
- To answer, DFJ begin with the following relaxation of the TSP.

$$\begin{aligned} \min \quad & \sum_{ij} c_{ij} x_{ij} \\ & \sum_{i:i<j} x_{ij} + \sum_{i:i>j} x_{ji} = 2 \quad \forall j \in 1, \ldots 5 \qquad (1) \\ & x_{ij} \geq 0 \quad \forall i, j \in 1, \ldots 5 \end{aligned}$$

- This is a linear optimization problem (LP) for which the tour is a (basic) solution.
- To prove the tour optimal, DFJ implicitly construct the "dual" of this LP by associating a "price" (DFJ call it a "potential") with each constraint.
- The "price" can be interpreted in a number of ways, but consider the economic interpretation as a sort of "toll" for traffic through a node.

Primal versus Dual

- From economics, the law of supply and demand says that the price should equal the marginal cost of increasing traffic through a node.
- Consider increasing the "traffic" through node 4 to 4 from 2.
- The solution to the system of equations becomes $x_{12} = x_{34} = x_{45} = 2$.
- This no longer represents a tour, but we can still consider the change in cost, which is 6.



- Thus, the "price" of a unit of traffic through node 4 is $(8 - 5 + 5 - 8 + 6)/2 = 3$.
- We now have two different notions of "cost."

> ***Primal***: the original cost in terms of the links.
> ***Dual***: the cost in terms of the node prices.

- Intuitively, these should be at equilibrium.

Deriving the Prices

- Suppose we impose that the direct cost of using a link should equal the prices associated with its endpoints.
- Then we can derive prices for all nodes by solving the system

$$\pi_i + \pi_j = c_{ij}$$

  for $(i,j) \in \{(1,2),(2,3),(3,4),(4,5),(5,1)\}$.
- This yields $\pi_1 = 5$, $\pi_2 = 0$, $\pi_3 = 5$, $\pi_4 = 3$, $\pi_5 = 3$.
- It is easy to check that this coincides with our earlier notion of price in terms of supply and demand.
- We have that primal and dual costs are equal.

$$2\sum_i \pi_i = \sum_{ij} c_{ij}x_{ij} = 32$$

  .
- So our two notions of cost indeed coincide.

Pivoting

- The question of whether the tour is optimal can be considered by comparing the quantity $\pi_i + \pi_j$ to c_{ij} for links *not* in the tour.
- If there is an (i,j) for which $c_{ij} < \pi_i + \pi_j$, this represents a link for which the cost of the link is at a discount with respect to the toll.
- The quantity $d_{ij} = c_{ij} - (\pi_i + \pi_j)$ is the "reduced cost" and represents the change in cost by "forcing" a link to be used.
- In this case, link $(1,3)$ has a reduced cost of -4.
- What happens if we try to include it in the tour?

"Reduced Costs" and Optimality Conditions

- Using modern terminology, we "pivot" x_{13} into the basis by increasing its value, adjusting values of other variables to maintain feasibility.

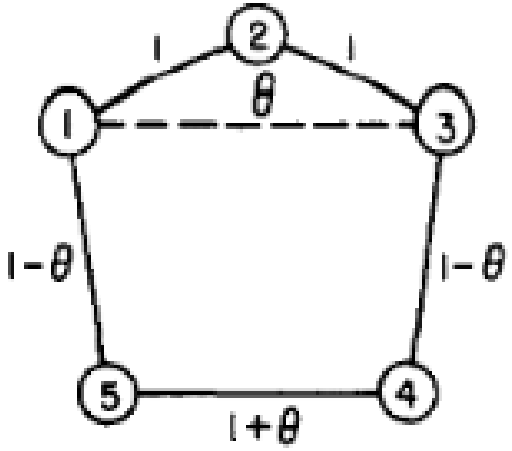


FIGURE 4

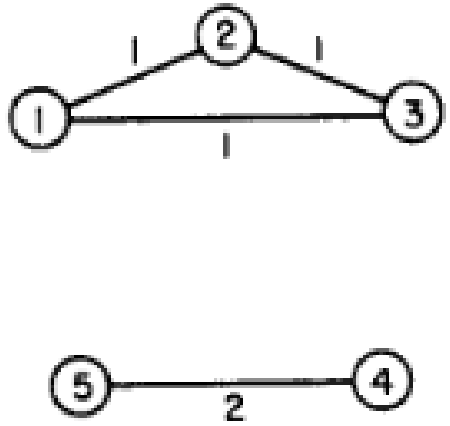


FIGURE 5

- The resulting solution is 28 and satisfies constraints, but it is not a tour.
- This is because the relaxation omitted upper bounds on the variables!
- We need the constraint $x_{45} \leq 1$.
- Adding this constraint to our relaxation, the resulting prices and reduced costs prove the optimality of the tour.

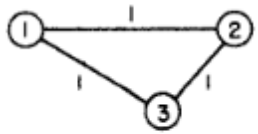
Primal Cutting Plane Method

- The method DFJ describe by example is to continue pivoting in this way, backtracking and adding a cut whenever the new solution is not a tour.
- This is what is now known as a *primal cutting-plane method.*

Basic DFJ Algorithm

1. Identify a feasible solution that is basic to some linear relaxation.
2. Perform a simplex pivot.
3. If the new solution is not feasible, add a cut, try again.

- The primary class of inequalities utilized is the so-called *subtour elimination constraints* (SECs).
- These eliminate solutions with multiple "subtours."



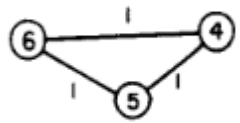


FIGURE 11

Reduced Cost Fixing

- The final piece of the puzzle is *reduced cost fixing*.
- They use reduced costs to obtain a lower bound on the optimal value.
- Given any basic solution $\tilde{x}$ with associated reduced costs $\tilde{d}$, we have

$$\sum_{ij} c_{ij} x^*_{ij} - \sum_{ij} c_{ij} \tilde{x}_{ij} = \sum_{ij} \tilde{d}_{ij} x^*_{ij} \geq \sum_{ij} \min\{\tilde{d}_{ij}, 0\}.$$

- Then for any (i,j) such that $d_{ij} > -\sum_{ij} \min\{\tilde{d}_{ij}, 0\}$, we must have $x^*_{ij} = 0$.
- This is a variant of the reduced costs fixing that is an important element of modern algorithms.

Important Observation

...in the latter stages often so many links are eliminated that one can list all possible tours...

The Solution

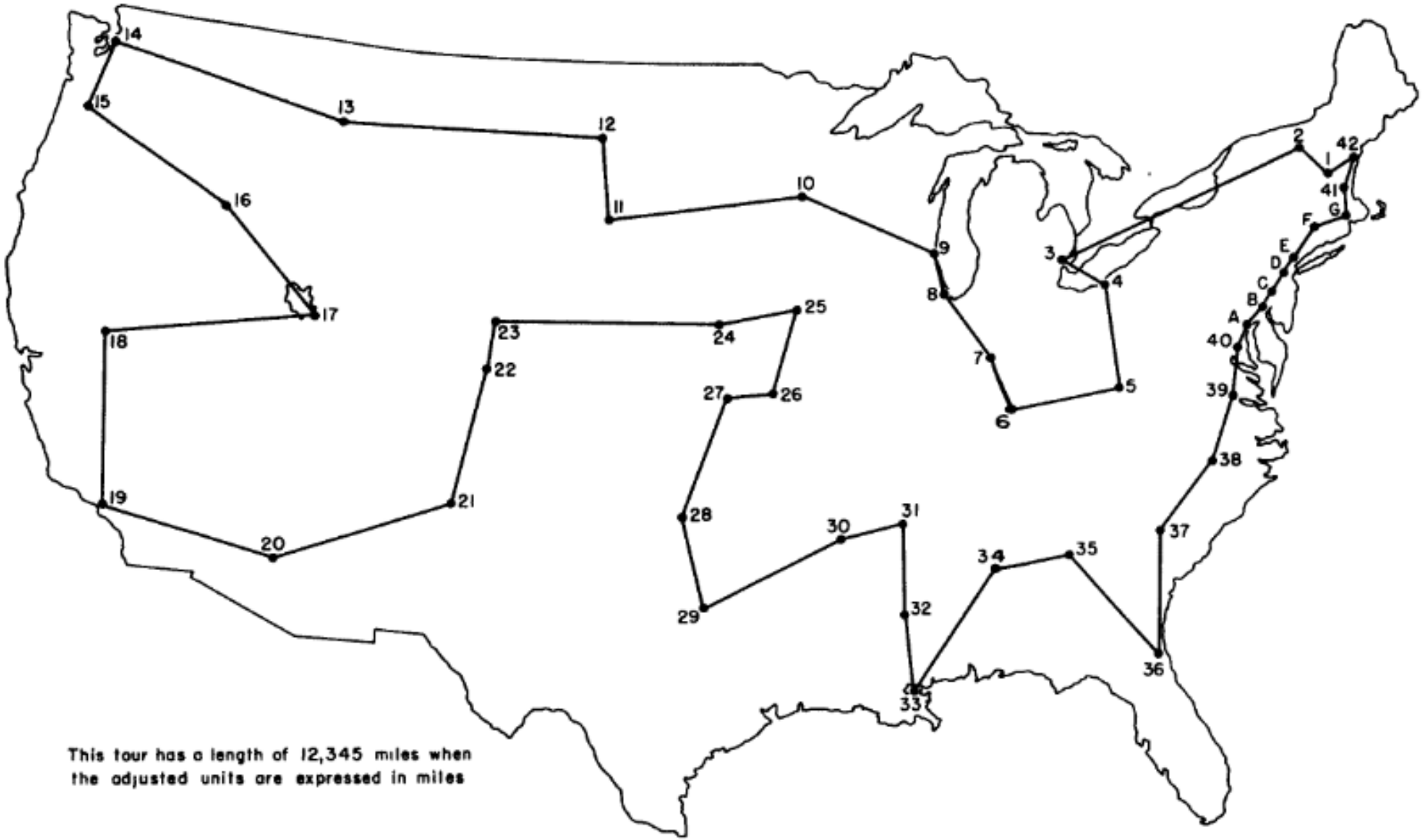


FIG. 16. The optimal tour of 49 cities

The Proof

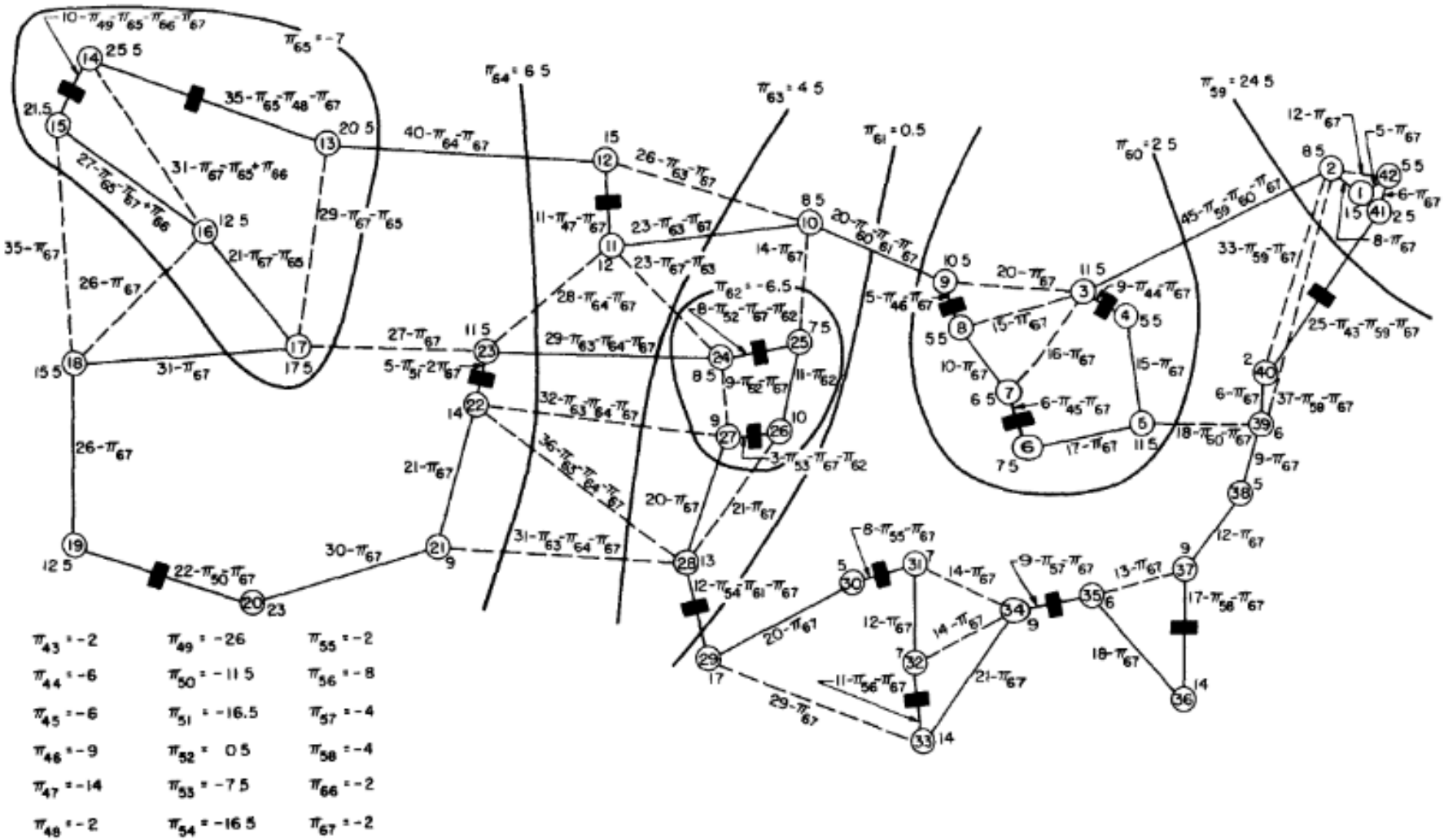


FIG. 17. Only the right-hand side of the equations satisfied by π_I are shown on the map; the left-hand side on line (I,J) is $\pi_I + \pi_J$. Dotted links (I,J) correspond to additional basic variables x_{IJ}.

Reflecting

- There is no clearly described, globally convergent algorithm.

> *It is clear that we have left unanswered practically any questions that one might pose of a theoretical nature concerning the traveling salesman problem; however, we hope that the feasibility of attacking problems involving a moderate number of points has been successfully demonstrated, and that perhaps some of the ideas can be used in problems of similar nature.*

- SECs are not enough for larger instances (even here, there are two mysterious non-SECs which are attributed to Irving Glicksburg).
- DFJ did, however, provide a formal, verifiable pure cutting-plane proof for the 49-city problem.
- This served as a template for what such a proof could/should look like and paved the way for much computational research that followed.

> *...we do not try to characterize the tours by the complete set of linear restraints, but rather impose just enough linear conditions...to assure the minimum...is assumed by some tour.*

Outline

The Path to Modern Solvers

- Although DFJ's proof for the 49-city problem did not involve enumeration, they suggested that it would be possible.
- Their paper was quickly followed by several more influential papers.
  - Markowitz and Manne [1957] coined the term "cutting plane," while suggesting an enumeration based strategy for solving MILPs.
  - Eastman [1958] suggested an enumeration-based method that can be viewed as a pre-cursor of branch and bound.
  - Land and Doig [1960] described the method we now call *branch and bound* (though they did not call it this).
  - Gomory [1958] suggested a "dual" cutting plane method.
- Fast forwarding, Crowder and Padberg [1980] was followed by Padberg and Rinaldi [1991], who introduced *branch and cut*.
- The latter combined all of the above elements into a powerful algorithm for solving large-scale TSPs.
- A decade later, Applegate et al. [2003] took these ideas still further and showed how to solve TSPs with *millions* of cities.

General-Purpose Solvers

- Until the late 1990s, it was widely believed that solution of large-scale combinatorial problems required problem-specific methodology.
- Balas, Ceria, and Cornuéjols [1993] and Balas, Ceria, Cornuejols, and Natraj [1999] questioned this belief and showed it was not well-founded.
- General-purpose cutting plane procedures, such as Gomory cuts and lift-and-project cuts revolutionized the solution of unstructured MILPs.
- Off-the-shelf solvers (mainly CPLEX, at the time) became effective tools for solving generic MILPs.
- Despite their generic nature, however, identifying and exploiting substructure was still a key technique.

Solvers Today

- Solvers today are *big bags of (very sophisticated) heuristics*, bound together by equally sophisticated control mechanisms.
- By carefully balancing these heuristics and paying close attention to numerical issues, great strides have been made.
- It is now reasonable to expect to be able to solve unstructured "large-scale" instances.
- Still, when you peel back the layers of the onion, modern solvers use an extremely well-tuned version of the early algorithms, including DFJ.
- Flow structure is perhaps the most important underlying substructure that is routinely identified and exploited.

Going Beyond Traditional Mathematical Optimization

- Today, traditional solvers for single instances are powerful and mature.
- There are a wide variety of commercial and open source solvers for all classes of deterministic mathematical optimization problems.
  - Linear optimization problems.
  - Nonlinear optimization problems.
  - Mixed integer linear/quadratic optimization problems.
  - Mixed integer linear/quadratic optimization problems.
  - Mixed integer nonlinear, convex and nonconvex.
- This opens opportunities to tackle "high-level" problem classes like
  - Multiobjective optimization.
  - Optimization under uncertainty.
  - Multistage stochastic optimization.
  - Robust optimization.
  - Multilevel optimization/Computational game theory.
  - Optimization problems with constraints derived from ML models.
  - Decomposition-based algorithms with unstructured subproblems.

Fulkerson's Continuing Impact

- It is easy to see the impact of Fulkerson's work on many aspects of modern computational methods.
- Almost every existing practical method of solving MILPs has components originally envisioned by Fulkerson.
- How can we continue to build on this legacy?
- What will the impact look like in another hundred years?

Outline

New Frontiers

The frontiers of research have shifted.

- Use of solvers as black boxes inside larger algorithmic frameworks.
- Use of mathematical optimization for “real time” applications rather than in just a planning mode.
- Focus on solving sequences of related instances efficiently rather than single instances (warm-starting, learning).
- Many new approaches to dealing with uncertainty and evolution over multiple time stages.
- Integration of mathematical optimization solvers with/into AI/ML algorithms.
- New wrappers and front ends that allow model development in a more “natural” and human-centered way.

New Interfaces

- Historically, the input to solvers has been in a flat file, such as `.mps` or `.nl` format.
- The user interface is typically through a modeling language or other front end that outputs a flat file.
- This has many implications, including that solvers only understand flat files in which knowledge of the problem structure has been stripped out.
- The most popular modeling languages have remained staunchly "algebraic," though this is slowly beginning to change.
- Constraint programming languages have long provided an alternative.
- Why has there not been more widespread adoption of these languages as front ends for our most powerful solvers?
- With the advent of LLMs, there are indications of change happening.
  - AMPL is beginning to support some constraints long standard in CP.
  - Alibaba demonstrated an LLM capable of formulating optimization models from a human language description.

Looking to the future

- More sophisticated algorithms for more realistic modeling.
  - It has long been realized that deterministic, single-stage optimization models are not realistic in many applications (Dantzig himself said this).
  - A result of the maturing of traditional solvers is the shift in focus to developing approaches for producing flexible/agile/robust solutions, not just a single "optimum" to an imperfect model.
- The integration of machine learning methods with optimization
  - incorporation of learning into optimization algorithms
  - Incorporation of machine learning into modeling (Gurobi Machine Learning and PySCIPOpt-ML)
  - the development of more "data-driven" methodologies.
  - The use of optimization to drive machine learning algorithms
- Use of new double-precision GPUs for solving MIPs and LPs (cheap, massive parallelism).
- More "human-centered" interfaces.
- Support for "real-time" optimization.

What might the future of combinatorial/integer optimization be?

- There is a need to put input into language of the user (not the modeler!)
  - How can we make it “easy” for a manager to provide the problem and allow problem instances to change with little change to the format?
- People expect fast answers… need solution in seconds from conceptualization to answer.
  - How do we make an optimization problem an app?
- Need to structure input as a manager would think about the problem and let software translate into:
  - the appropriate model formulation useful to the solver
  - the formulation may differ based on both the structure and the software to be used
  - the results need to be provided back to the user in the language of the user

Improving Optimization Technology

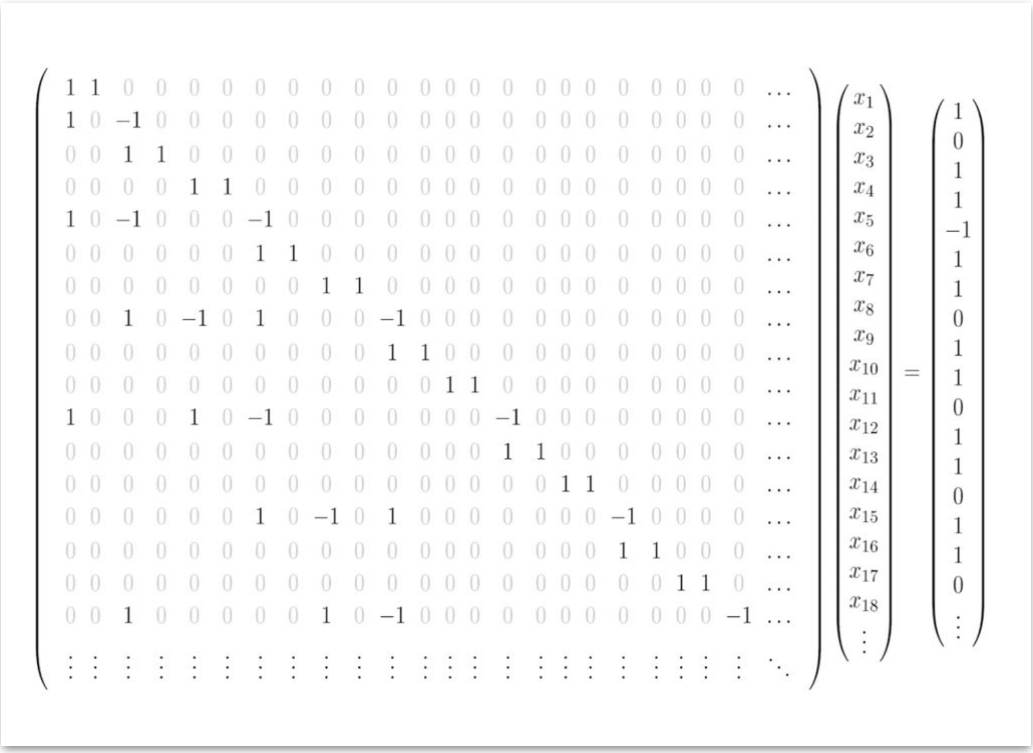
- Algorithms need to know the underlying structure without having to “find” it.
- Need APIs for important problem structures and then let the user choose the pieces that are needed.
- Create processes so that the entire effort is easy to transport from small test problems, to large examples, to cloud. From testing to production.

The way we were taught to build an optimization model

$$\begin{aligned} &\text{Maximize.} && c^T x \\ &\text{subject to} && Ax \leq b \\ & && x \geq 0, x \in Z^n \end{aligned}$$

- Translate business rules to linear inequality systems.
- Hand off to a solver.
- 🤞🙏
- Translate solutions back to business rules.

Thinking about problems in this manner results in the following structure:



Since the 1940s computers have been really good at linear algebra.

So it made sense to use matrices as the vehicle for optimization.

Downside: as modelers, we keep trying to force a linear-integer structure on logic constraints.

RESULT: We add big-M's and other structures that move the lp-bound far from the integer solution!
Equally important: the algorithms do not understand the mathematical structure of the original problem!

How does a manager think about a scheduling problem?

- Input:
  - Workers (name, days and times avail, type of work that x can do, restrictions on work assignments)
  - Jobs (jobname, time required, people who can work on given task, time windows)
  - Goals (worker preferences, completion times, fairness)
  - Precedences
  - Shift rules (e.g. min rest hours between shifts)
  - Added constraints
- Output = Solution(s)
  - Solution as shifts and assignments
  - Statistics
- Background testing
- Real-time solution monitoring
- Re-optimization evaluation

Many of these ideas are based on work done my NextMv.io (Ryan O'Neil and Carolyn Mooney)

How does a manager think about a routing problem?

- Input:
  - Stops (pickup name, location of pickup, delivery name, location of delivery)
  - Vehicles (vehicle name, vehicle capacity, vehicle preferences)
  - Orders (quantity, time windows, backlogs)
  - Precedences
  - Penalties
  - Goals (worker preferences, completion times, fairness)
  - Shift rules
  - Unique constraints

- Output = Solution(s)
  - Solution in terms of orders delivered, quality of deliver, happiness of drivers, safety issues (food and driving)
  - Statistics over multiple problem instances clustered by characteristics of the data.

- Background testing
- Real-time solution monitoring
- Evaluation of solutions over time

Many of these ideas are based on work done by NextMv.io (Ryan O'Neil (CTO) and Carolyn Mooney(CEO)).

Decisions should be:

Composable

- Small building blocks that are easy to understand
- Able to combine in many unique ways

Repeatable

- Input / output is production data
- Decisions are easy to re-create and debug

Testable

- Models look like other code
- State and transitions are easy to unit test

Scalable

- Models are binary artifacts
- Multiple deployment constraints

WHAT SOLVER? Gurobi, Highs, FICO, CP, ORTools, Pyomy, Heuristics, AMPL, etc.

HOW TO STORE AND UNDERSTAND OUTPUT:
Want understandable output, showing results by time of day and over time, under different conditions (goals, solvers, processors,etc).

MAKE IT EASY TO LEARN WHAT WORKS WHEN: Are there problem types that require different solvers, different parameters? (More on this in AI part of talk)

Many of these ideas are based on work done my NextMv.io (Ryan O'Neil and Carolyn Mooney)

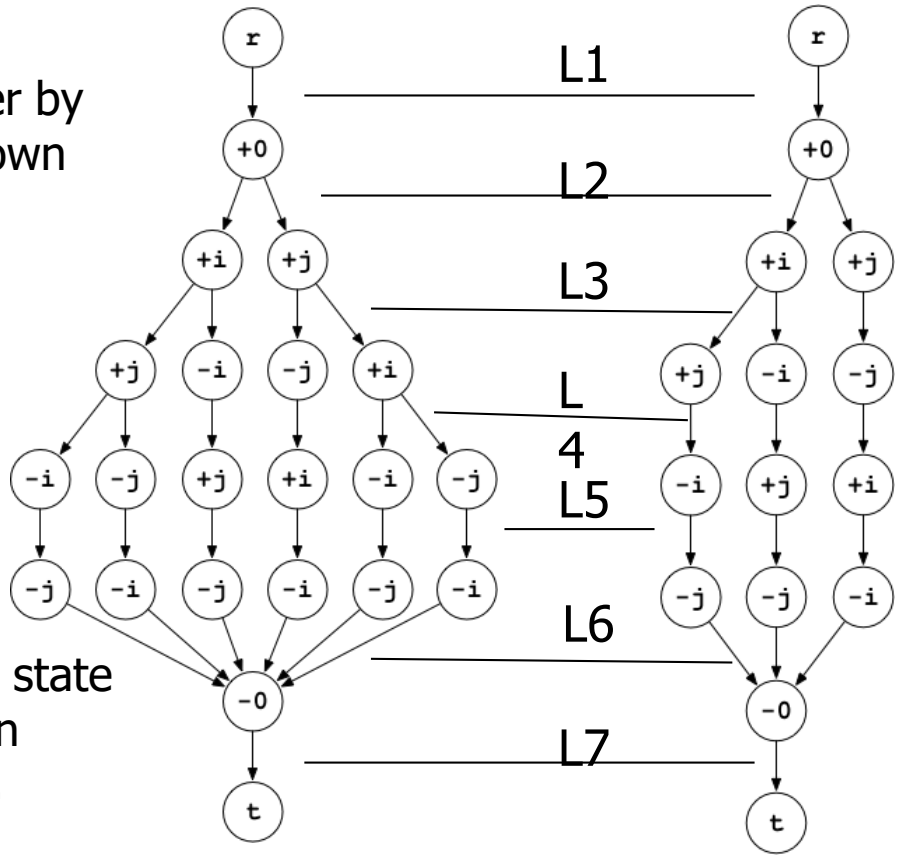
Some search tree ideas taken from Constraint Programming (CP), Decision Diagrams and Machine Learning

- We want a search tree that:
  - Allows input to be in the natural language of user
  - Treats the logic constraints directly (constraint programming)
  - Exploits the linear optimization bounding and cutting plane tools already built into our most successful MIP solvers
  - Determines the nodes most likely to produce good feasible solutions possibly through learning models.
- Decision Diagrams uses a concept that is imbedded in ML (and possibly in sophisticated MIP codes?):
  - Allows restricted branching (next slides);
  - Tightens constraints as well as domain states (similar to preprocessing in optimization and constraint programming)
  - Allows users to control branching rules
  - Has natural bounding concepts
  - Has logic-based branching (similar to constraint-programming)

Exact and Restricted (Restricted) Diagrams

Diagrams are constructed layer by layer in a top-down manner.

Exact diagrams fully explore the state space, which can grow very large.



Restricted diagrams act as a primal heuristic by dropping the worst states at each layer.

Dropped states can be reconsidered when time permits or feasibility requires search of other branches

What classic optimization models have properties exploitable using logic constructs and partial tree search?

- Vehicle Routing Problems
- Scheduling Problems
- Packing Problems
- Clustering Problems
- Budget Allocation Problems
- Matching Problems

ONE CAN MIX AND MATCH THESE APIs :

EXAMPLE: Constrained VRP is a packing problem mashed with assignment and routing components: "I want to pack as much as I can on a vehicle and end up with a sensible route."

Another issue: How to fix problems when things go wrong?

- If one understands the structure and data, then one can quickly determine the appropriate “neighborhood” of the problem. (e.g. What data has changed and what constraints/goals are impacted?)
- Given a well-specified “neighborhood”, search can be restricted to that neighborhood.
- Object: return to feasibility quickly and with the least disruption of the overall system.
- The user specifies the time allotted for the solution process; user can then evaluate results relative to time allocated to solving.

Another issue: How to fix problems when things go wrong?

- But, one needs to know an appropriate ”neighborhood”.
- Thus, the data should allow the algorithms to know where the change in the problem has occurred.
- Think about the VRP: what locations are impacted, what types of restaurants, what vehicles, etc.
- When a problem is translated into a MIP, this information is often lost, thereby making it difficult to have a more informed neighborhood search!

What might these concepts do for the solution and use of combinatorial optimization nmodeling?

- Modelers will be able to spend less time putting problems into integer-linear framework and more time in getting the model correct, analyzing the data, and providing intuitive output.
- Software should use the client's data directly and transform the data to produce different inputs for different solvers; similarly, software returns results to client's files.
  - APIs are built for specific problem structures (such as VRP, scheduling, clustering, assignment, etc.). Modelers work across decision domains. (COMPOSABILITY)
  - APIs use known methods to get bounds and handle logical restrictions common to the problem
- Modelers want to test different algorithms across the test set:
  - Which is best given a specific environment? Decision Diagrams, Constraint Programming, MIP, column generation, user heuristics
  - Choice is likely to be different if real-time versus planning… but planning makes real-time solving possible.

Importance of these concept to combinatorial optimization:

- Technology allows easy use of multiprocessing to allow lots of testing (Why not treat optimization problems as the deep-learning community does?)
- We need to collect MASSIVE data sets of problems and learn what works
  - Allows modeler to consider black swan events and then determine strategies to avoid them
  - Can shadow the solution technique and see how the decisions would change with alternative parameter choices
- Real-time problems need flexibility and contingency planning
- Algorithms can still be improved: e.g. need better bounds; need to think carefully about prioritizing alternative sub models
- We should always care about optimality! If using heuristics, evaluate solutions obtained against optimality criteria.

AI and OR technologies can complement each other.

- AI to help solve large, important MIP problems:
  - AI can help to determine “good” algorithms and parameters for optimization
  - AI might help direct the tree search
  - AI might be able to determine good feasible solutions
  - But… If one is using AI to learn how our algorithms should function and, possibly, provide “warm starts” for optimization, then one needs to be sure that the learning set is broad enough to not bias the algorithms
- Optimization to help deep learning:
  - Better optimization (Are there better nonlinear optimization techniques that can improve the tree search within deep learning?)
  - What have we learned from our long history of solving clustering problems, function determination, etc.
  - Can one train a NN using combinatorial optimization?

ML and OR Are Complementary

- It may seem as though machine learning and OR are in competition.
- Machine learning = function approximation and this has been in the OR toolbox since the beginning.
- Machine learning provides a very general methodology, techniques in OR are based on specific knowledge of structure.
- Many algorithms for optimization (including branch-and-bound) can be viewed as constructing a "dual function."
- Thus. optimization can be viewed as a specific form of machine learning in which we approximate a function of known form at a single point.
- Solving sequences of instances is a generalization which is closer to general machine learning in which we need agreement at multiple points.
- The development of general-purpose solvers obviated the need for customized solver development.
- Machine learning may serve a similar purpose—to fill in capability gaps.
- Machine learning is not a replacement for OR, but neither will OR methodology ever be as general as machine learning.

ML Can Augment Optimization Methodology

- We said earlier that a solver is a collection of heuristics.
- Many of these heuristics are ultimately trying to make predictions based on data gathered during the solution process.
- Machine learning may ultimately prove better at making these predictions than our current heurstics.
- Despite the massive amount of effort devoted to improving the performance of solvers, there are still aspects that are not well-studied.
- ML can help fill in the gaps.
- In some cases, ML may point the way towards development of improved algorithms (e.g., reinforcement learning).

Optimization Can Augment ML Methodology

- Ultimately, ML *is* optimization.
- Currently, methodology for nonlinear optimization dominates ML.
- However, many ML algorithms are solving what are actually discrete optimization problems.
  - The problem of constructing an optimal decision tree is fundamentally discrete.
  - RelU activation function enable approximation of step functions, etc.
- Does it make sense to tackle these problems directly using discrete optimization?
- There is some work in this direction already, but only time will tell.

Exciting time for OR tool developers, modelers and software developers

- Major problem areas that we have an advantage: supply chain optimization, real-time delivery, real-time scheduling, real-time re-pricing and re-distribution
- The field of integer nonlinear optimization has great potential, but we need modeling languages that make using our tools "easy".
- And… our software development needs to go past formulation to complete systems… from user input to explainable output in the user's language and with graphical presentations of results
- Next generation is being trained to expect answers in "zero time". To compete, we need much more experimentation. That means we need massive data sets!
  - How do we collect and distribute the data we need to develop the next generation of optimization software?
- Is this "doable"? Can we have an input process that maintains structure?

Questions and Comments

Characteristics of Decision Diagrams/Constraint Programming

- Constraints can be nonlinear, discontinuous, non-differentiable:
- Easy to handle time windows
- Easy to state precedence constraints (*in natural language*)
- Can provide priorities (goals) without specifying a specific penalty function
- Has statements similar to constraint programming
  - All-different;
  - If A not B; If A, not (B or C)
  - if/then;
- Sequencing constraints

References I

D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems. *Mathematical Programming*, 97:91–153, 2003.

E. Balas, S. Ceria, G. Cornuejols, and N.R. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1999.

Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical programming*, 58(1-3):295–324, 1993.

Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

H. Crowder and M.W. Padberg. Solving large-scale symmetric travelling salesman problems to optimality. *Management Science*, 26(5):495–509, 1980.

References II

G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large scale traveling salesman problem. *Operations Research*, 2:393–410, 1954.

G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

G.B. Dantzig, L.R. Ford, and D.R. Fulkerson. A primal-dual algorithm for linear programs. *Linear inequalities and related systems*, 38:171–182, 1956.

George B Dantzig. Programming in a linear structure. In *Bulletin of the American Mathematical Society*, volume 54, pages 1074–1074. AMER MATHEMATICAL SOC 201 CHARLES ST, PROVIDENCE, RI 02940-2213, 1948.

Willard Lawrence Eastman. *Linear programming with pattern constraints: a thesis*. PhD thesis, Harvard University, 1958.

References III

L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

L.R. Ford and D.R. Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958.

R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Monthly*, 64:275–278, 1958.

Isidor Heller. On the problem of shortest path between points. 1. In *Bulletin of the American Mathematical Society*, volume 59, pages 551–551. AMER MATHEMATICAL SOC 201 CHARLES ST, PROVIDENCE, RI 02940-2213, 1953.

Harold W Kuhn. On certain convex polyhedra. *Bulletin of the American Mathematical Society*, 61(557-558):8, 1955.

References IV

A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

Harry M Markowitz and Alan S Manne. On the solution of discrete programming problems. *Econometrica: journal of the Econometric Society*, pages 84–110, 1957.

M. W. Padberg and G. Rinaldi. A branch and cut algorithm for the solution of large scale traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

William R. Pulleyblank. Edmonds, matching and the birth of polyhedral combinatorics. *Documenta Mathematica*, Extra Volume ISMP:181–197, 2012.

Julia Robinson. *On the Hamiltonian game (a traveling salesman problem)*. Rand Corporation, 1949.