The SIAM John von Neumann Lecture

Margaret H. Wright

Computer Science Department
Courant Institute of Mathematical Sciences
New York University

International Congress on Industrial and Applied Mathematics

Valencia, Spain

July 16, 2019

I’m honored beyond words to be giving this lecture.

Thank you, SIAM!

This talk will describe a recently encountered real-world applied mathematics problem and its connections with the work of John von Neumann.

Born in Hungary in 1903, John von Neumann exhibited signs of genius at an early age and never ceased to do so. He is known for foundational contributions to quantum mechanics, geometry, ergodic theory, game theory, error analysis, duality, and the design of early computers. (This is a partial list!)

Hence we have room for only two items, chosen from many:

And we need a change of title:

A Hungarian mini-feast of applied mathematics

How it began: in late 2017, Russ Caflisch, Director of the Courant Institute, wrote:

I received a request from the Department of Sanitation in New York City for help with an optimization problem.

DSNY has 10^4 *employees, who can request job location changes. The priority of these choices is supposed to be ordered by seniority. But they have the issue that a job change by a lower priority employee could open up a position that is desired by a higher priority employee. So this requires iteration, which they now perform by hand. They are looking for a researcher who could work with them. Are you interested?*

Moved by Bell Labs nostalgia for unexpected down-to-earth real problems, I was very interested. Because DSNY had mentioned optimization, my initial impression was that their problem would include something like

$$\mathrm{minimize}_x\, f(x) \text{ subject to constraints on } x.$$

But early conversations with DSNY colleagues made clear that they were not asking about a “standard” optimization problem.

Hence we entered two familiar stages of work on real-world problems: learning new languages in order to define the problem.

Useful general information about the New York Sanitation Department:

- Largest sanitation company in the world;
- Among its responsibilities: garbage collection, recycling, street cleaning, snow removal in 6300 miles of streets;
- Approximately 10,000 employees, 275 collection trucks;
- Handles more than 10,500 tons of garbage and 1,760 tons of recyclables *per day*;
- More than 40,000 (!) people took the most recently offered exam for employment.

sanitation
DON'T
LITTER

sanitation
DON'T
LITTER
25DD-579
M.7
9A

25CW-529
M.1

L-81070
25CW-131
S.I.1
DSNY

Here are some facts relevant to the transfer problem.

- There are 59 “garages” in the five boroughs of New York City, each with multiple sections, adding up to approximately 1000 different locations.
- Each sanitation worker has a “home location”.
- Several times per year, sanitation workers can request to change their home locations.
- A sanitation worker seeking a transfer submits a (paper) form, listing up to three different locations in order of priority.

- A worker with a home location may request a transfer, but he/she cannot be forced to move if not offered a preferred location.

- The list of workers seeking transfers is prioritized by seniority (a unique positive integer for each worker).

- If a worker does not receive a preferred location in a given round of transfers, the request carries forward to the next round.

The mathematical question is: How does DSNY decide on the new assignments?

NB: This is not a "big data" problem!

Here is an exact quote from the official "DSNY Business Rules for Transfers":

The idea is to ensure that a person's highest available choice on their list is fulfilled, based on seniority, availability, and freed-up positions during the transfer process.

This language seems to define an objective function and some constraints, and thus appears to pose an optimization problem. But it is NOT a standard optimization problem.

This realization and its implications led to a happy "aha!" moment.

The DSNY problem is a matching problem.

This is the first link to von Neumann: matching problems fall into the field of game theory, and

modern game theory began with John von Neumann!!

His 1928 article “Theory of Parlor Games” proved the famous minimax theorem, and his 1944 book with Morgenstern, *Theory of Games and Economic Behavior*, is widely considered to have created the now-lively field of game theory.

Discovering that the real-world problem you want to solve corresponds to a mathematical formulation is one of the great joys of applied mathematics.

There is an enormous literature today about game theory, especially in economics, and about matching problems, *especially in computer science*.

Some well known instances of matching problems are: selection of locations for medical residencies; organization of donors/recipients of kidney transplants; assignment of students to schools; and allocation of housing by universities.

The DSNY transfer problem can be viewed as a certain kind of housing allocation problem.

So let's talk about housing allocation problems.

A (generic, oversimplified) housing allocation problem includes:

- a set I of “agents” (the term to be used from now on in this talk), the i-th denoted by a_i;
- a set H of houses, each denoted in this talk by a capital letter;
- A preference profile for each agent, expressing the agent’s (strict) preferences with respect to houses.

Note that the houses don’t care who lives in them. (In some matching problems, both sides have preferences.)

A matching is an assignment of houses to agents.

An allocation is a matching of agents and houses so that each agent is assigned at most one house and no house is assigned to more than one agent.

A mechanism is a systematic procedure (i.e., algorithm) that produces a matching.

What are the properties of a “good” matching?

A matching is Pareto-efficient if there is no other matching that makes every agent weakly better off and at least one agent strictly better off.

A matching is strategy-proof if no agent can benefit by misrepresenting his/her preferences.

A matching is individually rational if no existing agent strictly prefers his/her original house to the one resulting from the mechanism.

How to find a mechanism that produces matchings with these desirable properties?

Start with the simplest idea.

The most obvious mechanism is called *serial dictatorship*.

We go down the list of agents in priority order and give each agent his/her top-ranked house that is still available.

This simple mechanism has many desirable properties, but its assumptions (e.g., that all houses are initially available) do not reflect most real-world instances.

So we make the problem more realistic by assuming that some houses are occupied and some of the agents (“sitting tenants”) who live there would prefer to move.

One possibility: let agents who have houses and want to move treat their houses as available. Priorities are then assigned to those agents, and we apply serial dictatorship with the new priorities and a new list of available houses.

But this is likely to violate individual rationality, since an existing tenant may end up with a house that he/she likes less than the one he/she has now. This property of the mechanism would create a bad situation because unsatisfied sitting tenants would be unhappy.

We need a system where existing tenants can move, but can stay put if nothing preferable is available for them. (This policy is typical in university housing draws.)

There is such a system!

A 1999 paper in the *Journal of Economic Theory* by Abdulkadiroğlu and Sönmez, "House allocation with existing tenants", presented a mechanism with an evocative name:

You Request My House; I Get Your Turn

or, for short, YRMH-IGYT.

(I can't work out how to pronounce this acronym!)

Here is how YRMH-IGYT works.

We process the list of agents in a dynamic order, as follows. An initial list consists of the agents in priority order.

Repeat steps 1–3 until no more agents remain. . .

1. Assign the first agent on the dynamic list to his/her top available house; similarly for the second agent on the dynamic list; and so on, until reaching an agent (say, agent k) whose top choice is a house currently occupied by an agent (say, agent m) who is lower on the dynamic list, i.e., has a lower priority.

Agent k is requesting the house of agent m (who has lower priority), so that agent k is now "you" in the YRMH-IGYT terminology. Agent m, whose house is being requested by agent k, is "I".

2. This is the key step. Agent m “gets the turn” of agent k, and moves *just before* agent k in the dynamic list, which is now reordered. Then proceed from step 1.

3. If at any point a cycle forms, it must consist of agents, each of whom requests the house of the agent who is next in the cycle. Remove all agents in the cycle by assigning them to the houses they request and return to step 1 with the updated list of agents and houses.

A&S note that the YRMH-IGYT algorithm produces the same result as the “Top Trading Cycles” (TTC) algorithm, described by Scarf and Shapley in 1974 in the *Journal of Mathematical Economics*, where it is attributed to Gale.

The TTC algorithm is described and implemented in terms of a directed graph representing the agents and houses.

It sounds very different from YRMH-IGYT, but gives the same result as YRMH-IGYT. (This is a theorem.)

YRMH-IGYT is more intuitive to many people; using TTC makes proofs about the results easier.

Both are tricky to implement.

Example: 5 agents $(a_1, a_2, a_3, a_4, a_5)$, and 5 houses (A,B,C,D,E)

Priority order of agents: 1,2,3,4,5

House E is vacant; agent a_5 has no house (e.g., a new employee).

Agent preferences:

a_1	a_2	a_3	a_4	a_5
B	A	E	A	E
	D			C

Starting configuration:

Agents	a_1	a_2	a_3	a_4	a_5
Houses	A	C	D	B	*

Start YRMH-IGYT with the highest-priority agent:

1. a_1 ‘requests’ house B, occupied by a_4. So a_4 gets the turn of a_1.

2. a_4 requests house A, occupied by a_1, so a_1 and a_4 form a cycle and can swap. Remove a_1 and a_4 from the proceedings.

   Three agents are still in play, with priority list $(2, 3, 5)$. Recall that the original preferences for these three are

a_2	a_3	a_5
A	E	E
D		C

The current configuration is

Agents	a_2	a_3	a_5
Houses	C	D	$*$

3. a_2 (the highest-priority remaining agent), who occupies house C, has the next turn. a_2's first choice is House A, but it is out of the picture.

   a_2's second choice is House D, which is occupied by a_3. Hence a_2 is requesting the house of a_3, and a_3 takes the turn of a_2, i.e., goes before a_2.

4. In a_3's turn, he/she wants house E, which is vacant. So a_3 moves out of house D and moves into house E. Now a_2 can move into D and move out of C.

5. a_5 is next (and last). a_5's first choice is House E, but it is occupied by a_3. a_5's second choice is House C, which is vacant, so a_5 gets House C.

   Here is the final configuration.

Agents	a_1	a_2	a_3	a_4	a_5
Houses	B	D	E	A	C

How does this compare with the starting configuration?

- a_1 is strictly better off (gets B, first and only choice)
- a_2 is strictly better off (gets second choice)
- a_3 is strictly better off, gets E, his/her first and only choice)
- a_4 is strictly better off (gets A, first and only choice)
- a_5 is strictly better off (gets second choice, House C); initially had no house.

Looks good!

Is the DSNY problem "solved"?

Unfortunately, no.

On the bright side, the YRMH-IGYT mechanism gives a matching that is Pareto-efficient, strategy-proof, and individually rational.

Full disclosure: there are other special features of the DSNY problem, not discussed here, that make it more complicated.

In any case it turns out, for “personnel” reasons, that YRMH-IGYT does *not* always produce results that are “acceptable” for the DSNY.

This phenomenon (discovering an unexpected, undesirable property of a mathematical model) often emerges when the model doesn’t include all the complicated features of reality. (Typically, the model does not have enough constraints.)

Consider the following example 1, which is totally unrealistic but illustrates a complication.

3 agents (a_1, a_2, a_3) and 4 houses (A,B,C,D), where house D is initially vacant.

Here is the initial configuration.

Agents	a_1	a_2	a_3	$*$
Houses	A	B	C	D

The priority of the agents is $1, 2, 3$.

Here are their housing preferences.

a_1	a_2	a_3
C	D	D

Using YRMH-IGYT,

1. We start with a_1, who occupies house A and wants house C. House C is occupied by a_3, so a_3 takes a_1's turn and goes first.

2. Since a_3 wants (vacant) house D, a_3 gets house D, freeing house C. This means that a_1 can have house C. This takes care of a_1 and a_3.

3. The only remaining agent is a_2, who wants house D, which was initially vacant. But D is now occupied by a_3, so a_2 does not get his/her choice.

The result is

Agents	a_1	a_2	a_3
Houses	C	B	D

The blue color indicates that there was no change for a_2, who is especially unhappy because the house he/she wanted most was given to a lower-priority person.

But a_1, the highest-priority agent, is happier because he/she has moved to house C.

And a_3 is happier because of being able to move to house D.

Is the difficulty caused because house D was initially vacant? Consider a slightly modified problem with 3 agents and 3 houses, with initial configuration:

Agents	a_1	a_2	a_3
Houses	A	B	C

The preferences are different:

a_1	a_2	a_3
C	A	A

Using YRMH-IGYT, when a_1 requests house C (initially occupied by a_3), a_3 takes the turn of a_1. This means that agents a_1 and a_3 will swap, and a_3 will get house A.

The result is

Agents	a_1	a_2	a_3
Houses	C	B	A

Once again, there was no change for a_2.

House A, which a_2 wants, is now occupied by a_3, and a_2 must stay in his/her original location.

a_2 might regard this as “unfair”, even though a_1 got what he/she wanted.

DSNY’s official policy office ruled that the ‘optimal’ solution in this case is that no one moves.

Further thinking is needed about the overall DSNY goal quoted at the beginning:

The idea is to ensure that a person's highest available choice on their list is fulfilled, based on seniority, availability, and freed-up positions during the transfer process.

Even a solution that is Pareto-efficient, individually rational, and strategy-proof does not appropriately respect the seniority of agents in all circumstances.

Something to ponder: Is there a mathematical characterization of what DSNY really wants from the transfer process?

Being continued.

At this stage perhaps you are wondering: is there only one major item (game theory) in this mini-feast, connecting John von Neumann's contributions and the DSNY problem?

By no means! (But there's only time for one more.)

Von Neumann's influences on computing are broad, diverse, and consistently at the highest level: he was a major architect for the EDVAC (Electronic Discrete Variable Computer) computer in the mid-1940s; he was an early authority on issues in computer arithmetic, etc., etc.

Von Neumann's work with EDVAC also provides an inspiring "deep dive" example of addressing the need to produce software that actually solves the problem. A theorem, even a published algorithm, is not usually enough.

Von Neumann's sometimes overlooked legacy is:

HE WROTE CODE.

In a fascinating 1970 paper “Von Neumann’s first computer program”, Donald Knuth describes in detail a program written by John von Neumann for sorting—probably not the topic you expect—in the context of his work during 1944–45 with EDVAC.

Von Neumann chose to write a program for sorting because he wanted to know whether the proposed EDVAC instructions were adequate (and fast enough) to perform logical control of complex processes.

He not only wrote his own code, he also created the algorithm—a “divide and conquer” algorithm, today called “merge sort”.

Knuth’s paper, a tour de force, reviews von Neumann’s code line by line, explaining certain choices and respectfully commenting that a few aspects could be improved (!).

What’s the connection to DSNY? Sorting is an essential ingredient in all the code written (so far) for the transfer problem because of the need to keep track of multiple dynamic priority lists.

The burden for today’s programmer who wants to include a sort in his/her code is tiny compared to what von Neumann needed to do.

Anyone in this audience who writes code and needs to sort will almost certainly be using a high-level language (perhaps one of the following), all of which provide library routines for sorting.

- Matlab. *A multi-paradigm numerical computing environment and proprietary programming language.*

  The command `B = sort(A)` would almost certainly be adequate, but one also has the choice of sorting along different dimensions and of specifying additional parameters for the sort.

- Python. *Python is an interpreted, high-level, general-purpose programming language. . . Python's design philosophy emphasizes code readability. . .*

  [from documentation of `sort`] The sort() method sorts the elements of a given list in a specific order. . . You can also use Python's in-built function sorted() for the same purpose.

- `R.` *R is a programming language and free software environment for statistical computing and graphics.*

  To sort a data frame in R, use the order( ) function. By default, sorting is ASCENDING. Prepend the sorting variable by a minus sign to indicate DESCENDING order.

- `Julia.` *Julia aims to create an unprecedented combination of ease of use, power, and efficiency in a single language.*

  [about sort] By default, Julia picks reasonable algorithms and sorts in standard ascending order

- `Fortran.` *Fortran is a general-purpose, compiled imperative programming language that is especially suited to numeric computation and scientific computing*

  Basic sorting and searching routines for vectors on github.

In contrast, Von Neumann wrote a 23-page memo, then classified as “top secret”, describing essentially every aspect of his code.

Knuth says, after his dissection of von Neumann’s program,

> *Like nearly all programs, this one has a bug. . . . If von Neumann had had an EDVAC on which to run his program, he would have discovered debugging!*

Here (from Knuth’s paper) is a facsimile of the first page of von Neumann’s program, which “represents a significant step in the evolution of computer organization as well as of programming techniques”.

⑤

(g) We now formulate a set of instructions to effect this 4-way decision between (α)−(δ). We state again the contents of the short tanks already assigned:

$\bar{1}_1)\ N\ n'_{(-30)}$ $\quad\bar{2}_1)\ wN\ m'_{(-30)}$ $\quad\bar{3}_1)\ wN\ x^0_{m'}$ $\quad\bar{4}_1)\ wN\ y^0_{m'}$

$\bar{5}_1)\ N\ n_{(-30)}$ $\quad\bar{6}_1)\ N\ m_{(-30)}$ $\quad\bar{7}_1)\ wN\ 1_{\alpha(-30)}$ $\quad\bar{8}_1)\ wN\ 1_{\beta(-30)}$

$\bar{9}_1)\ wN\ 1_{\gamma(-30)}$ $\quad\overline{10}_1)\ wN\ 1_{\delta(-30)}$ $\quad\overline{11}_1)\ \dots \to C$

Now let the instructions occupy the (long tank) words $1_1, 2_1, \dots$:

$1_1)\ \mathrm{T}_1 - \bar{5}_1$	σ) $N\ n'-n_{(-30)}$	
$2_1)\ \bar{9}_1\ s\ \bar{7}_1$	σ) $wN\ {}^{1_\gamma}_{1_\alpha}{}_{(-30)}$	for $n' \substack{\geq \\ <} n$
$3_1)\ \sigma \to \overline{12}_1$	$\overline{12}_1)\ N\ {}^{1_\gamma}_{1_\alpha}{}_{,(-30)}$	for $n' \substack{\geq \\ <} n$
$4_1)\ \bar{1}_1 - \bar{5}_1$	σ) $wN\ m'-m_{(-30)}$	
$5_1)\ \overline{10}_1\ s\ \bar{8}_1$	σ) $wN\ {}^{1_\delta}_{1_\beta}{}_{(-30)}$	for $m' \substack{\geq \\ <} m$
$6_1)\ \sigma \to \overline{13}_1$	$\overline{13}_1)\ wN\ {}^{1_\delta}_{1_\beta}{}_{(-30)}$	for $n' \substack{\geq \\ <} m$
$7_1)\ \bar{2}_1 - \bar{6}_1$	σ) $N\ m'-m_{(-30)}$	

$8_1)\ \overline{13}_1\ s\ \overline{12}_1$	σ) $wN\ {}^{--\overline{13}_1)--}_{--\overline{12}_1)--}$
	i.e.
	σ) $wN\ {}^{1_\gamma\ 1_\beta}_{1_\delta\ 1_\alpha}{}_{(-30)}$
		for $m' \substack{\geq \\ <} m$
		
		for $\substack{m' \geq m,\ n' \geq n \\ m' < m,\ n' \geq n} \quad \substack{m' \geq m,\ n' < n \\ m' < m,\ n' < n}$
		i.e for $\substack{(\gamma)(\beta) \\ (\delta)(\alpha)}$, respectively.

$9_1)\ \sigma \to \overline{11}_1$	$\overline{11}_1)\ 1_\alpha, 1_\beta, 1_\gamma, 1_\delta \to C$	for (α),(β),(γ),(δ), respectively.
$10_1)\ \overline{11}_1 \to C$		

~~[illegible]~~

Now

$\overline{11}_1)\ 1_\alpha, 1_\beta, 1_\gamma, 1_\delta \to C$ for (α),(β),(γ),(δ), respectively.

Thus at the end of this phase C is at $1_\alpha, 1_\beta, 1_\gamma, 1_\delta$, according to which case (α),(β),(γ),(δ) holds.

(h) We now pass to the case (α). This has [redacted] 2 subcases (α_1) and (α_2), according to whether $x^0_{m'} \geq$ or $< y^0_{m'}$. According to which of the 2 subcases holds, C must be sent to the place where its instructions begin, say the (long tank) words $1_{\alpha 1}, 1_{\alpha 2}$. [redacted] Their numbers must

~~[illegible] instructions [illegible] follows~~

The many layers of von Neumann's program make it a perfect dessert in our Hungarian mini-feast.

Summary:

Trying to solve real-world problems is one of the things that SIAM (and all the societies in ICIAM) do best.

Tracing intellectual threads is complicated, especially when the source of ideas is von Neumann, who made so many contributions to so many different areas.

But it seems clear that, after more than 70 years, John von Neumann's work, ranging from the foundations of game theory to nitty-gritty coding, is visible and remains deeply influential in an amazingly wide range of areas in mathematics and computer science (including the DSNY transfer problem).

Thank you, John von Neumann!