



Smithsonian  
*National Museum of American History*

*Lemelson Center for the Study of Invention and Innovation*

**Computer Oral History Collection, 1969-1973, 1977**

**Interviewee:** Dr. Philip Wolfe

**Interviewer:** Robina Mapstone

**Date:** November 28, 1972

**Repository:** Archives Center, National Museum of American History

**MAPSTONE:**

It's October the--October? It's November the twenty-eighth, and I'm talking to Dr. Philip Wolfe, W-O-L-F-E, right?

**WOLFE:**

One "l" in Philip.

**MAPSTONE:**

One "l" in Philip. At IBM Research Lab. OK. Maybe the best thing to start off with is your education, you know, going back to university and coming up to Princeton and ... being there, and what you did on the machine.

**WOLFE:**

OK. I did all my college work at the University of California at Berkeley, starting in '43, with an interruption for military service. I took an A.B. in Physics and Math in 1948. And then worked slowly for a Ph.D. in Mathematics, which I got in 1954. My first job was instructor with the Department of Mathematics at Princeton University from 1954 to 1957. Following that, I went back to California as a mathematician with the RAND Corporation in Santa Monica and worked there from 1957 until about the summer of 1965, at which time I was free-lance for about a year and a consultant to IBM Research in Zurich, Switzerland and then to the RAND Corporation in Santa Monica. And, finally, in 1966 I took up my present job, which is mathematician member of the Research staff at IBM's Thomas J. Watson Research Center in Yorktown Heights, New York. Ever since I was so high I was interested in computing.

**MAPSTONE:**

How high's so high? [Laugh].

**WOLFE:**

To me one of the fascinating things about science was, in

addition to understanding how things worked, you could actually make predictions about how things were going to work. And to do that you had to have numbers and understand formulas. And for a long time I wasn't interested in the mathematics as such, but started to learn it because these college physics books that I used to puzzle over when I was in high school had all these funny mathematical signs and I eventually got some college friends of mine to teach me what they were. I was attracted by, kind of, the beauty of mathematical ideas, but I didn't really understand the beauty of mathematics itself until sometime late in my undergraduate college career; and suddenly I realized that this was somehow the language of the universe and the stuff that was really worthwhile. But--also, I've always liked to work with my hands. And somehow the parallel in mathematics to working with your hands and working with numbers seemed--seeing the results of calculation based on theory actually come out and say something interesting about the world, that kind of thing has always been--I think the specific event that got me into my present line of work happened sometime in 1948, when there was an article on the theory of games in Astounding Science Fiction magazine. I'd never heard about the theory of games before. As soon as I heard about it sounded absolutely fascinating, because here was a kind of union of something rather abstract in mathematics with something that's tremendously concrete and practical.

**MAPSTONE:**

Do you remember who wrote the article?

**WOLFE:**

No, I don't. It was a--I can't even remember the article. [Most likely] it was a very slick popularization of something, quite possibly, the author didn't understand very well. As soon as I heard about it, I went out and bought the book by Von Neumann and Morgenstern and started to study that. That was at the time I was getting my A.B. And after a year or so I was quite certain I wanted to work in that field and I searched through the Math Department at Berkeley to find somebody who could advise me to do the work. And finally I got Ed Bracken, who had done a little work in game theory, and he reluctantly consented to advise me in my thesis work. Meanwhile, I think it must have been as a first year graduate or sometime around then, I began to hear about computing machinery. The first thing to turn me on was the differential analyzer

**MAPSTONE:**

Was this

**WOLFE:**

by Bush.

**MAPSTONE:**

Yeah.

**WOLFE:**

In fact, I think it was in 1948, UCLA gave a short course in working on the differential analyzer and then what have you. I took the course for three or four days.

**MAPSTONE:**

Do you remember the date?

**WOLFE:**

No, I don't. I didn't know anything about digital computation at the time and I didn't have a chance to learn. The next great influence, my thesis advisor was a friend of George Dantzig's, who took his degree at Berkeley in the Department of Statistics. And in the summer of 1951 I was looking for a summer job and Bracken got me in touch with George Dantzig, who at that time was working at the headquarters of the U.S. Air Force in a group called roughly Mathematical Modeling and Planning; so I got the job, and that was my first contact, I think, with the real world of applied mathematics, where people were trying to use mathematics for very definite and practical problems. Part of the excitement of that time, I think, for everybody involved was that the applied mathematics that was being developed for these problems, and in which Dantzig had a big hand, was somehow much more direct than classical applied mathematics. You could move immediately to your model and from your computer solution right down to the man on the street, so that you could tell him, according to your figures, he should be planning his business in such and such a way. It wasn't a matter of designing turbines, or figuring out how the chemical reactions turn out in some little piece of the business, but rather the whole business. One of the earliest problems that Dantzig's group had was how to keep the Berlin Airlift going. How to schedule the planes in and out of Tempelhof and into the fields and back in the Western Zone. How to maximize the amount of coal and food and all that stuff that they could bring in during the Russian blockade. And that, I understand, was a very successful application of linear programming.

**MAPSTONE:**

Did you work with it?

**WOLFE:**

No. No that was--the blockade must have been in '48, '49. I worked on a theoretical problem that Dantzig posed while I was there, and pretty well solved it. That later resulted in, I guess, my first paper with Dantzig-Orden [?]. It was published several years later. That experience, I think, really set me up in the line of work that I've been doing ever since. And, as I said, the excitement of having mathematical theory that would lead to a problem that could be solved numerically and whose answers could be used almost

immediately, were really meaningful to somebody, was a fascinating prospect. One kind of irrelevant anecdote which I already told you, Bobbi, was at the Pentagon for the first few days when I was wandering around trying to find out what the people in this group were doing. I walked into the office of a man whose name I think was Hortby [?] and asked him what he was doing. And he looked up from his desk and said, "I'm making a detailed model of the American economy. Please excuse me now. I'm terribly busy."

**MAPSTONE:**

[Chuckle].

**WOLFE:**

But that was the kind of ambition that they had. And I think people still have it, it's a sort of force that carried on this kind of work, the work in linear programming and mathematical programming, was the idea that you maybe could make a detailed model of the American economy that would actually make sense. And you could put it on a computer and see it run, or devise a method that would make it run as well as possible or at least maximize something in particular you had in mind.

**MAPSTONE:**

Hadn't Leontief already got started in this field?

**WOLFE:**

Leontief was working with this group.

**MAPSTONE:**

Oh, was he?

**WOLFE:**

As a consultant, I suppose, when I first saw him around, at the Pentagon. And several other figures who became important in this area: Abraham Charnes. I understand John Von Neumann was around at the time, but I didn't actually run into him. Well, I did my thesis work and, as I said, got my degree in '54. My thesis topic was on games of infinite length; they tend to extend the Von Neumann theory kind of in a mathematical direction which, although I proved a number of true theorems turned out to be of very little interest later. But at the same I had started working on some other problems. The linear programming solution of the traveling salesman problem, which is something that has exercised a lot of mathematicians.

**MAPSTONE:**

What's that?

**WOLFE:**

The traveling salesman problem is like this. Given all the capitols of the states of the United States, given all the distances between them, plan the tour that will hit each of these places just once, or at least once, that covers the minimum distance.

**MAPSTONE:**

Oh my.

**WOLFE:**

Now that's a problem, if you take the forty-eight continental capitols of the states, for direct attack it has an absolutely fantastic number of possibilities if you have to try it. Something like, if you have a marvelous computer that can try one every micro-second it would still take you a million years to just come up with the answer.

**MAPSTONE:**

I never would have thought about it. That's incredible.

**WOLFE:**

And so, for such a problem the intellectual challenge is to devise short-cuts ideally to prove theorems about problems like that. Theorems that would say "well, all you have to check is the following possibilities." Or: "all you have to do is the following reasonable amount of calculation and you will get the right answer." Well, the traveling salesman problem has proved still to be almost impossible. There have probably been two or three hundred papers written on attempts at that numerical solution of the traveling salesman problem and it still is a staggeringly big problem. It's kind of a classic intractable problem, one that even our best methods involve just sitting down and slopping through vast amounts of computations. But the success of the field of linear programming and mathematical programming in general is probably due to the fact that there are problems that look almost as bad that are vastly easier to solve. And it was, I think, essentially Dantzig that--certainly assisted by the ideas of other people--who isolated this classic problem successfully, and that's the problem that is the linear programming problem. The situation where you are content to write down a system of linear equations that describes all the goals you have to meet--linear equation meaning that every variable that enters into the model does so in just a proportional or an additive way. If I double the quantity of meat I eat in a meal, then I double the amount of vitamins and calories I get from the meat. And I can just take those vitamins and calories and add them to the vitamins and calories I get from potatoes, and I know what I've got for the whole meal. Now many models you can set up do that... . Many models you can set up don't. There is no such way of looking at the traveling salesman problem, for example. You can't write

down just a bunch of these linear equations which tell you that the salesman is indeed in all forty-eight cities; and that's the trouble. But for those problems that you can describe in this way it's a tremendous and interesting practical class and it turns out that there are mathematical solution methods that will work in a reasonable length of time. And that's the reason for the whole success. Because this is based on linear equations and we understand how to solve those pretty well. It has the additional feature that the variables in the equations can be solved for a required-- have some other constraints, usually they're required to be positive, since I do not want, when I work out my output, to be told that I have to eat minus

**MAPSTONE:**

minus quantity.

**WOLFE:**

two chops.

**MAPSTONE:**

[Laugh].

**WOLFE:**

That's a kind of non-classical feature that no mathematician with a computer had considered very seriously before Dantzig. And then Dantzig evolved a particular method of computing these things, called "simplex method," which has turned out to be very, very efficient in practice. There is an interesting side anecdote: It appears to be, oh, so far as I know, the one place where Johnny went very badly wrong.

**MAPSTONE:**

Oh?

**WOLFE:**

Because he, as I understand it, always contended with Dantzig that the simplex method would take an absurdly long amount of time to solve linear programming problems. In fact, Johnny published at least one paper on an alternative method. I think in 195[2] he was offering as an alternative to the simplex method calculation and everybody that's tried that method has given it up very rapidly.

**MAPSTONE:**

In contrast to the simplex.

**WOLFE:**

I don't know how it's worked in practical problems. Having been connected pretty closely with Dantzig for a long time I've naturally thought about what the factors in his career are due to. And I suppose there's luck in choosing, in lighting on something that is as successful as that. But I don't think there's very much luck in it. I think it's a real, kind of a piece of his particular genius to have got the right abstraction, to have identified the mathematical problem which would cover a lot of interesting practical cases and at the same time be one that you could handle on the computer. Even, now distinguishing again between linear programming and the traveling salesman problem, for interesting linear programming problems, even so you do need a computer. There's just no question. As soon as the model involves a few dozen variables or a few dozen equations. You set this up, the problem, by hand, would take at least a day to solve. Probably there is that much arithmetic work, so that without the computing facility, again this area would be of no interest, because the real problem you choose formally could not be solved in practice.

**MAPSTONE:**

So once again we see the machine and man working--one feeding the other.

**WOLFE:**

Exactly.

**MAPSTONE:**

This approach in mathematics would not have developed as much as it has if it hadn't been for the computer.

**WOLFE:**

I think it really wouldn't have developed at all; it would have remained an academic curiosity, or probably not even an academic curiosity, because it deals with such kind of primitive notions. The computer had to exist in order to make the problems practicable, but beyond that, the industrial application of it had to exist to make it worth doing at all. And this--I suppose I came just a little bit late in the peak of the development of linear programming. Much of it was going on at the RAND Corporation before I arrived in 1956. Essentially, I think, I was hired as a replacement for Bill Orchard-Hayes who had worked closely with Dantzig. Dantzig joined RAND, by the way, I think, in 1952. He, and Orchard-Hayes as his programmer, had got a package for linear programming working, I think, about 1955; this was sort of the first reasonably sized, reliable computer routine that could do linear programming problems and it became very widely popular very quickly. It was distributed by the SHARE organization.

**MAPSTONE:**

Oh yeah. Do you know what?

**WOLFE:**

It was for the IBM 701.

**MAPSTONE:**

Ah. Do you know what specific--the project was? What the projects were that they first--the problems they first solved?

**WOLFE:**

I know one of the early ones was the Quintance [?] diet problem. The problem of, I think there were nine nutrients required, so many calories, so many units of vitamin A, B, C, a few other things like that. Large lists of foods, a few hundred foods, knowing their constitution, how much of each of the nutritive requirements they could meet, the problem you propose is to meet all these requirements by choosing the diet of foods that will cost the least. This problem was actually kind of a classic. It was formulated before the war by Mhanssen [?] and Stigler [?] who used hand computing methods over many months on the data he had at that time to get near a palatable answer. This is the sort of problem that nowadays, or probably in 1955, is just gobbled up in a few seconds by an actual computer. It was by no means a phony problem at RAND in about 1961 or '62. I did a fair amount of work with the Kern County Land Company, who were feeding large herds of cows, and were able to pick up some money from the fact that the feed prices would change daily, or at least weekly, where they had to buy their feed. And about twice a week they would phone in whatever their dietician said their cows would be a little short of this time, and they would phone in the complete price list for the available feeds, like various types of corn and meal, and so on and we would run this data into our linear programming code and phone back what they should be buying.

**MAPSTONE:**

[Laugh]

**WOLFE:**

and what proportions they ought to be mixing to feed their cows.

**MAPSTONE:**

Oh great.

**WOLFE:**

That was really kind of a small scale thing, even though they had, I think, maybe a

hundred thousand cows to feed or so, it was really a tiny linear programming program. One you could almost do by hand now. You'd asked me before to say something about--in fact, I skipped a period, I see, the three years I was at Princeton from '54 to '7. Sometime along in there, oh, I think about the summer of '55, I completed a phase of work that I'd been in for a couple of years on the so-called quadratic programming problem, which is an extension of linear programming in which the goal function as it is called, the thing that you are trying to optimize is a more general function than the linear function, it is a quadratic function instead, and I had devised an algorithm that seemed interesting to me; but I very much wanted to see if it would work in practice on reasonable sized problems. I heard about the Institute computer, and decided to find out about it, and that was my introduction to digital computing machinery.

**MAPSTONE:**

Up to then you had still been working with pencil and paper and hand calculator?

**WOLFE:**

With pencil and paper, and I had tried to do a little work with an analog computer at Princeton, but somehow it wasn't congenial. It was not a very accurate machine and so far, although in principle it perhaps could do interesting jobs, the analog machines have not been successful in optimization of problems.

**MAPSTONE:**

Do you remember whose machine it was?

**WOLFE:**

A fellow named Pine in the Electrical Engineering Department was in charge of the machine at that time.

**MAPSTONE:**

Was it a commercially built machine, or had someone built it at Princeton?

**WOLFE:**

I think perhaps it was Princeton-built; but I don't know.

**MAPSTONE:**

Hm.

**WOLFE:**

But the Institute machine was a whole world in itself. And a kind of fascinating world. I read the fundamental papers on this design and whatever coding manuals they had around the place, [and] sat down to program my algorithm for it. I never did. I didn't realize what a terrible job programming even a fairly simple calculation was for a machine of that kind.

**MAPSTONE:**

Laugh].

**WOLFE:**

Because those were the days, at least around the Institute--well, I think more advances had been made elsewhere--but around the Institute they were the days of coding in absolute binary. You take your IBM card and, having mapped out exactly where you wanted everything in the program to reside in the 1000 words of core storage, you would then have to punch into the card exactly where all the bits of information you wanted to reside in storage were.

**MAPSTONE:**

That's incredible. I can't visualize that--for us, today. Just incredible.

**WOLFE:**

It's so hard that you can only do the most primitive things when you are calculating in that fashion. I did do some interesting things. I sort of got interested in the computing art itself and I think I lost sight of wanting to program my algorithm. So I programmed and did complete a general purpose trace program. One that would run through a second program and follow it and print out the result of all its steps as it was being executed. This wasn't of much value in itself, but once you have done something like that, you absolutely, thoroughly know the machine, and you know everything it can do. So I had the--well, I guess I didn't score too well there, because, although I learned the machine thoroughly and got some programs to run, I didn't do any useful work.

**MAPSTONE:**

You didn't. [Laugh].

**WOLFE:**

But I learned a hell of a lot.

**MAPSTONE:**

Was anybody working at that time on ways to come up with program languages other

than basic binary at Princeton?

**WOLFE:**

Yes. And I had heard that there was a version of an assembler working for the Institute machine. That's all I remember about it.

**MAPSTONE:**

Yeah, because by this time, what are we talking about, 195-

**WOLFE:**

'56.

**MAPSTONE:**

1956--we've got 701, we've got 704 and we've got—

**WOLFE:**

Assemblers were a firmly established concept.

**MAPSTONE:**

Right. And yet, here people at the Institute still had to struggle with basic binary.

**WOLFE:**

Yes. I think that's essentially because the Institute was dying--the Institute machine was dying at that time anyway.

**MAPSTONE:**

Yes. And Von Neumann was--died by that time--when did he die--in '56, didn't he?

**WOLFE:**

I think so.

**MAPSTONE:**

So it was right around that period. And Goldstine, was he still at the Institute?

**WOLFE:**

**Computer Oral History Collection, 1969-1973, 1977**

Philip Wolfe Interview, November 28, 1972, Archives Center, National Museum of American History

So I understood, and I think I must have met him there, but he wasn't, as far as I knew, visible on the computing scene. Bigelow was there. He was the principal person I had contact with. I thought he was, he was running it. Hans Maehly was there, I guess, during my last year at Princeton. Working at the Institute, I guess, primarily interested in the Institute computer.

**MAPSTONE:**

But by this time the computer had become sort of like a research--just something to become familiar with computing techniques on, maybe.

**WOLFE:**

I think so, yes. And otherwise it was really—

**MAPSTONE:**

No serious work was being done on it.

**WOLFE:**

I believe they were still working on weather problems or just computing with weather problems. Continuation of something that was started quite a while before.

**MAPSTONE:**

Did you know Jim Charney? Did you run across him?

**WOLFE:**

No.

**MAPSTONE:**

Because he did a lot of weather work. Oh, I think he did it on the ENIAC.

**WOLFE:**

Well, I think then, finally, that my tastes were, for a mixture of mathematical research and computing work, were pretty well set during that period, and I left Princeton to go to RAND in 1957. It wasn't entirely clear what my role was going to be. Dantzig figured that I was going to be his programmer. And I had a somewhat different view--somewhat higher view of what my position would be. That took a year or so to resolve. I did join the Computer Science Department rather than the Mathematics Department. And stayed with the Computer Science Department during my time there, which I think I probably benefited from, because I was--I made then the top mathematician in the Computer

Science Department

**MAPSTONE:**

Ah [Laugh].

**WOLFE:**

and I would not have been in the Mathematics Department.

BOTH:

**MAPSTONE & WOLFE:**

[laugh].

**WOLFE:**

Well, during that time, I continued studying linear programming. I did, especially during the--well, during the years, I think 1959, 1960, did a lot of personal programming on both the JOHNNIAC machine at RAND and also the 704, which I think they must have just brought in about 1957. Mostly I was doing this--well, I had some responsibility for maintenance of the programming work in linear programming. Although the real work was carried on by Leola Cutler at RAND who had been working with Bill Orchard-Hayes up until the time he left. And we made a natural association. She maintained the linear programming production routines on the ... There was something that I couldn't understand. It was written in 704 machine language and it was a very long, very sophisticated, complicated code, especially because the 704 for which it was written, and I think the one we had when I arrived, had a 4,000 word high speed memory and tapes. And the program itself was something like 15,000 instructions, so it was necessary to use fairly sophisticated techniques to bring in parts of the code when you needed them, and what with the jumping around of the data. So I never understood that routine. I did understand much more easily what was going on in the JOHNNIAC. It had 8K core and no tapes at all, so everything you did had to fit and reside in core. It had a fairly good high speed drum; but I somehow managed to avoid using that in any of my work. And I programmed a number of small exercises--did some work in linear programming on that, just to begin to understand the computational part, but I didn't really get into programming in a big way until the arrival of FORTRAN.

**MAPSTONE:**

I was going to ask you about that. Let's backtrack a minute. When Dantzig started working on computers--you were talking about linear programming--what languages--how are you getting this information into the machine? Did Dantzig and his people come up with their own machine language or--?

**WOLFE:**

No. No. They used standard languages. In case there is any confusion about the word programming, I want to say it has nothing to do with computer programming.

**MAPSTONE:**

Okay. That's what I was trying to understand.

**WOLFE:**

"Programming" was the word that the Air Force used to refer to any kind of plan, because they were producing what they called programs. Plans of activity. And the full title of the project that Dantzig started--it couldn't be called "Planning Using Linear Equation Models." It was called "Programming Using Linear Equation Models." It was then contracted to linear programming.

**MAPSTONE:**

Linear programming. Okay, because this is something I have always been a little murky on, you know, because linear programming somehow being caught up in the computer generation you figure it must have something to do with using the method of linear mathematics, but the programming was using it with a machine. But that's all right, obviously.

**WOLFE:**

The computer programming was like that for any scientific calculation. And one uses whatever language it's best to have. Well, as I said, I had a lot of trouble with 704 assembly language. I heard about FORTRAN. As a matter of fact, it was I who introduced the RAND Corporation to FORTRAN.

**MAPSTONE:**

Oh, really?

**WOLFE:**

Nobody there believed in it. FORTRAN had very hard slogging among professionals at the start. Because it was in those days such great fun, and it still is, but it is not so important anymore, it was tremendous fun to get down right into the machine, use the machine language, and try to use it in an optimum way. I've done it myself and I've seen people spend days trying to save a few microseconds.

**MAPSTONE:**

[laughter] Yeah.

**WOLFE:**

It was part of the professionalism of the game, and the idea that a language would come along in which absolutely any fool could write programs that would work, was, I think, sort of distasteful. Nobody believed that it would work.

**MAPSTONE:**

Oh, and sort of ...

**WOLFE:**

Yes. Bill Orchard-Hayes told me on several occasions that I would never be able to solve linear programming problems using the FORTRAN code.

**MAPSTONE:**

Why?

**WOLFE:**

I don't remember the reasons. And they sort of evaporated. Well, as I say I--well, they evaporated in view of the success of FORTRAN and other languages in doing these tasks.

**MAPSTONE:**

How is it that you were turned on, felt that FORTRAN would be useful when you were surrounded by people who weren't, including--was Dantzig skeptical of FORTRAN?

**WOLFE:**

I don't remember that he had a view on this. In computing matters he usually, I think Dantzig's views were generally kind of second hand. As far as I know, he has never done any hands-on computing himself.

**MAPSTONE:**

Oh, he has always had somebody doing programming, his computer programming.

**WOLFE:**

Yeah. Well, after making some kind of effort to understand the linear programming routine that was written in the machine language that we had, I got so discouraged, I

figured that I would never learn the computing art that way. I heard about FORTRAN. I talked our Computing Center into getting a copy of the system, and read the manuals and started to use it. Not with the belief that I could do any serious computing--but that it would be sort of good training for me. At least it was very easy to formulate algorithms and write them down and execute them using FORTRAN. And, in fact, from the point of view of the mathematician, let's say, the computing mathematician, the number engineer, there really isn't any difference. FORTRAN is translated into machine language and will execute exactly the things you would like to happen with the numbers. The thing the professional will gripe about is that the FORTRAN logic is supposed to be very general purpose and is set up to accommodate any kind of calculation anybody might want to do. It is much more elaborate than the logic that you'd use if you were writing your own hand-coded code. If you know the problem, you can anticipate what's going to happen; if you know the structure of it in detail, then you can achieve considerable efficiencies over some sort of a more general purpose scheme.

**MAPSTONE:**

Well, you still have control over each of the flip flop elements.

**WOLFE:**

Right. But it is not a matter of--numerically, it makes no difference. You still have to add numbers together and multiply them using the registers of the machine. It is just the convenience if you have complete control over all the code that is formed, of knowing exactly where the given data is going to be at a given time and being able to use it. Whereas the way FORTRAN has written the machine program, you may have to wait for it or do it in some kind of organized fashion that you wouldn't like to. And it might be slower for that reason. And it turns out, in the evolution of the art, that this was really no handicap at all; but it wasn't obvious from the beginning. [interruption] I lost my train of thought.

**MAPSTONE:**

Are you pressed for time?

**WOLFE:**

No, as long as I have between eleven and twelve to work up my ... lecture.

**MAPSTONE:**

Oh, today is Tuesday.

**WOLFE:**

Yes, and I've got a lot of students to see. I'm enjoying this so much--should I try to

compress more?

**MAPSTONE:**

No. No, don't.

**WOLFE:**

I haven't rambled like this for a long time.

**MAPSTONE:**

No, this is fine.

**WOLFE:**

Okay, so, probably about 1959, I began just doing linear programming in FORTRAN because I was interested in the process and I wanted to understand better how linear programming problems got solved, you see in simplex method. One of the big open questions, and it is still wide open, is the question of being able to predict how much time it is going to take you to solve a given program. There is a lot of good, solid empirical evidence, but almost no theory to tell us why it works as well as it did. And that's sort of, I guess, where Johnny went wrong. There was no theory that could say that it was going to be a very successful experiment, and that is irritating if you are a mathematician, especially one with Von Neumann's power.

**MAPSTONE:**

Yes.

**WOLFE:**

In other words, well, it's still not understood. It is one of those things we live with and we know it's terrific and we can make an empirical prediction about how well, about how much time it is going to take you to solve a problem. It is a pretty good prediction, but it is only based on experience. It is not based on the more interesting structured theory, the one we would like to have, because it is really involved, it must be a pretty complicated subject. Anyhow, as I say, I pioneered FORTRAN at RAND--perhaps being the only person in the Computer Science Department who had that much trouble programming for the machine; but I wanted to go that route. For about a year, I think, we struggled along on the first version of FORTRAN which was very hard to use because it would compile your whole program every time, even if you only changed one instruction. You had to go back and compile the whole program, which, on the 704--I mean, my compilations started running between fifteen, twenty minutes, maybe a half an hour. One day I had mistaken my instructions for setting switches on the machine and they called me and said, "You have just been running three hours," and they were wondering whether to

about the job.

**MAPSTONE:**

[Laugh]. Oh, my God.

**WOLFE:**

FORTRAN II came along, which had subroutine features, so that you could break your code up in little pieces and only have to recompile a little piece when you made a change. Then my project went ahead very well. So I built this fairly sophisticated, in terms of the arguments of the time, linear programming package, wrote some beautiful documentation for it, submitted it to SHARE, talked about it to my friends, and that's probably what made my reputation in the computational art in this area. Even though I did that work, I reduced that in 1960, I still find versions of it popping up here and there. The latest version that somebody gave me was at least a fourth generation descendant. In the meantime, it has been put into FORTRAN IV and then released by CDC,

**MAPSTONE:**

Oh, that's crazy.

**WOLFE:**

translated back into something more appropriate to the 360.

**MAPSTONE:**

PL/I or--Oh, that's lovely.

**WOLFE:**

It became very widespread, thanks to the beautiful work of the SHARE organization at that time--without SHARE, I am sure, the number engineering would have been held back very substantially. SHARE was quite an exciting business at that time. Everything new and solid in the way of algorithms was usually programmed fairly quickly, was talked about at SHARE meetings, submitted to the SHARE distribution agency, and got very good advertising and distribution. I'm sure it must have had a lot to do with the growth of IBM itself as an organization, because that program pool was of tremendous value.

**MAPSTONE:**

Yeah.

**WOLFE:**

Not only to the Corporation, but to individuals like me, who were getting that kind of work from the public. There was really no other outlet for it, almost no other outlet. ACM was not publishing algorithms at that time. What's probably not obvious to people who haven't worked much in the numerical art is what getting algorithms to work really means. I think it's still true, with the exception of one book, a kind of difficult book, written by Bill Orchard-Hayes: There may be a hundred books on linear programming; all of them describe the simplex method. In no case could you take their description, program it, and have a working code.

**MAPSTONE:**

It doesn't sound like the books are very useful.

**WOLFE:**

Well, they are useful to a mathematician.

**MAPSTONE:**

Who already is into the method.

**WOLFE:**

But they still won't teach him how to run the method. And among other things, perhaps the most important thing, especially in linear programming, it's very critical to decide whether a number is positive or negative. Now if you've taken a batch of numbers, and you mess them up in the computer, and you come out with one that is really, really, tiny, it's very important to know whether it should be positive or negative, if there is not ground out there introducing this number by this time. If you really can't make an intelligent decision as to what it would have been had it been computed perfectly by an infinitely accurate machine. So you have to devise heuristic dodges to make intelligent decisions and for getting yourself out of traps if the decisions happen to have been wrong. And these are the things that you only understand--well, they are problems that have been proposed to you by a computer when you start computing. And they can only be understood by a kind of hands-on tinkering to see what kind of answers are actually going to work. And these are the things that a number engineer knows and can do, that are extremely hard to teach very well. And, in general, in the academic world nobody's thought about teaching them.

**MAPSTONE:**

It's just, you know, we want to get over this spot, how did this machine do it, sort of thing.

**WOLFE:**

Yes. How did I get off the track here? Well, I think that's most of the story of my involvement with computers. It kind of tapered off around '62 and '63. I worked on the designs of successors to the linear programming routine I had, but perhaps have done more theoretical work and theoretical work of non-linear programming problems, which are much harder than linear programming problems. They still aren't very well understood.

**MAPSTONE:**

I thought you pretty much stayed with the linear—

**WOLFE:**

No, actually, I made my name in non-linear programming, the work that I started at Princeton on quadratic programming. At least that was my first kind of substantial contribution to the art. Then the work that Dantzig and I did in '59 and '60 on so-called decomposition algorithms, an attack on very, very large linear programming problems, on linear programming problems with millions of algorithms and constraints. Those two have probably been my major contribution to the computational art. So I guess one has been non-linear programming and the other linear programming.

**MAPSTONE:**

During the period at RAND, what were some of the concrete end results of the work that was being done? In other words, what were some of the projects being worked on there that you can think of that might be effective, like feeding the cows or diets?

**WOLFE:**

There were almost no specific applications. RAND Corporation, at that time, was funded almost entirely by the Air Force. And the contract involved skimming off something like ten percent of the total contract to for what was called RAND sponsored research. That was not committed to any specific Air Force need or anything else. Whatever RAND wanted to do. And since the--well, since 1950, perhaps, RAND had a very strong tradition of promoting work in the theory of games and mathematical programming. I understand Johnny was around a lot during that time. And RAND had--well, at that time--was given bales of money for summer consultants who'd come in, well, I think, during several summers, '51, '52, around that area, the place was full up with statisticians, mathematicians, and economists, who were working in mathematical programming.

**MAPSTONE:**

And also getting their hands on to JOHNNIAC, which was—

**WOLFE:**

Yes.

**MAPSTONE:**

I'm not sure if my dates are right, but it seems it was completed by about '52 or '3.

**WOLFE:**

That sounds quite right.

**MAPSTONE:**

Sounds about right.

**WOLFE:**

Oh, I should interpolate. I loved JOHNNIAC. I thoroughly understood that machine. It was probably partly responsible for the demise of my marriage.

**MAPSTONE:**

[laughter]

**WOLFE:**

there were times when the janitorial staff, around eleven thirty at night, would say, "Please, we've got to--you've screwed up our schedule enough--we've got to clean up in here--will you turn off that machine and go home."

**MAPSTONE:**

[Laugh]. I wonder how JOHNNIAC would feel about that. It was just that much fun working on it, was it?

**WOLFE:**

It was. Yea; it was an elegant machine, in a way that I nostalgically feel the present machines aren't one bit. You could sort of see its whole innards in front of you. And you could access any part of it and understand exactly what it was doing at that moment if you wanted to.

**MAPSTONE:**

Yeah.

**WOLFE:**

And, of course, it wasn't very efficient. It wasn't high speed, and there was very little demand for it. So I could sit down at the console in the evening and just play it myself, like an organ.

**MAPSTONE:**

You mean it wasn't well used?

**WOLFE:**

It was used pretty thoroughly during the day, I think. But this, as I said, and it was in competition with the 704, which was many times faster,

**MAPSTONE:**

Oh sure.

**WOLFE:**

and had larger capacity, and also, again, the work you did on the 704 could be communicated to other 704 users, which was an extremely important part of what we were doing at the time. We were trying to build up the whole computing art and not just write programs for JOHNNIAC. But I had about a year during which all I wanted to do was write programs for JOHNNIAC.

**MAPSTONE:**

[laugh] Dear JOHNNIAC. SHARE of course, although it grew out of the mess with the 701--you know, eighteen machines and eighteen different languages--assembler languages and all of that junk. It really took the 704 to give SHARE its chance to be effective. And, as you say, nobody was writing programs for JOHNNIAC but the JOHNNIAC users. There was not the interrelationship between the Princeton machines that there was between the 704.

**WOLFE:**

Right.

**MAPSTONE:**

We're just about to run out of tape.

**[End of Interview]**