A PROGRAMMING AND PROBLEM SOLVING SEMINAR

by

Kenneth A. Ross and Donald E. Knuth

This report contains edited transcripts of the discussions held in Stanford's Computer Science problem solving course, CS304, during winter quarter 1989. Since the topics span a large range of ideas in computer science, and since most of the important research paradigms and programming paradigms were touched on during the discussions, these notes may be of interest to graduate students of computer science at other universities, as well as to their professors and to professional people in the "real world."

The present report is the eighth in a series of such transcripts, continuing the tradition established in STAN-CS-77-606 (Michael J. Clancy, 1977), STAN-CS-79-707 (Chris Van Wyk, 1979), STAN-CS-81-863 (Allan A. Miller, 1981), STAN-CS-83-989 (Joseph S. Weening, 1983), STAN-CS-83-990 (John D. Hobby, 1983), STAN-CS-85-1055 (Ramsey W. Haddad, 1985) and STAN-CS-87-1154 (Tomas G. Rokicki, 1987).

Contents

Chapter 1

Introduction

The "Data Sheet" for the course, including a statement of each of the course problems, appears on the following pages. This material was handed out on the first day of class. In the problem discussions we assume that the reader is familiar with the particular problem statement.

1.1 Data Sheet

1.2 The Participants

	Professor
DEK	Donald E. Knuth
	Teaching Assistant
KAR	Ken Ross
	Students
ESC	Edward Chang
RC	Roland Conybeare
AG	Adam Grove
UH	Urs Hoelzle
DK	Dinesh Katiyar
RK	Robert Kennedy
PL	Patrick Lincoln
SM	Sanjoy Mahajan
AM	Arul Menezes
SJP	Steven Phillips
DQ	Dallan Quass
DS	Dan Scales
MY	R. Michael Young
	Auditors/Others
MA	Knut Almgren
ADG	Ashish Gupta
AH	Andy Hung
AL	Alon Levy
RM	Rebecca Moore
DP	Dan Pehoushek
NS	Nirmal Saxena
BT	Becky Thomas
ANT	The Ant Colony

Chapter 2

Late-binding Solitaire

Tuesday January 10

DEK arrived in suit and tie, his traditional dress for the first class. (He wore a *dark* suit, in fact, because it was also his birthday.) He gave a brief history of the course, describing how George Pólya originated the idea of a problem solving course many years ago, and how George Forsythe had promoted the idea in the computer science department. The course existed 21 years ago, when DEK joined the department, and DEK has taught it every other year since 1977.

Four years ago, the class was videotaped, and the tapes were disseminated to various parts of the globe. Similar courses at other institutions, such as CMU, have recently been set up following the success of CS304 at Stanford.

DEK then started discussing the mechanics of the course. Most of the information discussed can be found on the "Data Sheet" that was handed out (see Chapter 1). DEK emphasized that he will try to make CS304 a "world-class course for world-class students"; the department has always given this course special status. DEK stressed cooperation rather than competition, except for the trivia hunt which will be every group for itself. (The trivia hunt questions will be handed out at the end of class on Tuesday January 24, and the answers will be due in class the following Thursday.)

As far as machines go, we have special permission to use `polya` for work on this course. For Problem 5, we will be using the Mac II's in Sweet Hall, where there is a development lab full of them.

DEK then started getting to know the class by proceeding around the room and asking students to give a brief description of themselves. He observed that in 1977 about 60% of students in the course played musical instruments, but that recently almost none of the students do so. The popular hobbies in the class this year seemed to be volleyball, reading, and backpacking.

After all this DEK raced out of the room and returned a short time later with a deck of cards. He proceeded to discuss Problem 1, by demonstrating a game of late-binding solitaire. Having not previously rigged the cards, he "cheated" a little in order to demonstrate all the rules. However, he discouraged students from cheating in their programs, saying "it's not official Stanford policy." DEK said he knew that this version of solitaire had not been investigated before, since he had made up the game himself.

The original game requires one to make a legal move whenever possible. There are still choices involved, as there may be several possible moves at any particular time. DEK described the original game as using a "greedy algorithm".

In order to make it more interesting, DEK once tried laying out all 52 cards and, after seeing all the cards, begin to make moves. In this version, he found he could almost always win by arranging to get four cards of the same rank on top of the rightmost piles, and then proceeding methodically. But with only ten cards laid out, it was hardly ever possible to win. In between these extremes, with 18, it was usually interesting.

The statement of Problem 1 requires a program that will determine (as quickly as possible) whether a winning sequence of moves exists, and if so to demonstrate one. We are not interested if the cards can be reduced to 2 or 3 piles, although this may be an interesting extension of the problem.

DEK asked if there were any simple conditions that would demonstrate the impossibility of winning. DQ said that one such condition is whether there is a visible card which has no match in suit or rank. DEK agreed, and suggested that students look for early tests for failure. He also suggested looking for "restricted" sequences of moves that would be sufficient, rather than all possible sequences.

Some quotes from this class:

"This is a class on how to do research. In this class we do intimate problem solving." — DEK.

"It is fun to go all out for a limited amount of time on challenging problems." — DEK.

"I will try to only ask a person's name ten times during the quarter." — DEK.
"You asked me about twenty times last quarter!" — SJP.
"Yes, *Steven*, but I get ten more tries this quarter." — DEK.

"Computers are a great time-saving device for playing solitaire. You can say 'Play me 100 games and tell me how many I won.' " — DEK.

"When you start a PhD thesis you take baby steps, but soon enough you are taking giant steps, and you are the only person in the world who understands the problem domain as well as you. It is then your responsibility to solve all of the related problems before you forget it all." — DEK.

Thursday January 12

DEK arrived and asked whether anyone had found a solution to the solitaire position in the handout. Nobody had, although some students had practised playing the game. No groups had been organized, so DEK dealt out one card to each student, and mused about how they could be used to allocate groups. The allocation was finally done by ad hoc means, grouping people with "similar" cards subject to dividing the non-comp-takers evenly among the groups. The groups chosen were:

- AG, ESC, MY, RK
- DQ, RM, SJP, UH
- AM, DK, RC
- AL, DS, SM

It was decided to use initials rather than full names in these notes to preserve a degree of anonymity in case somebody said something incorrect. When it was pointed out that DEK hardly could claim his initials ensured anonymity, he replied that it doesn't matter if he makes mistakes, since he already has tenure. A key of names to initials appears in Chapter 1.

DEK remarked that the data generated for Problem 1 was random. He had no idea whether there was a win or not, except for the first set, for which he knew there was a win. The data is available on `polya` . (There are actually more deals there than are listed in the Data Sheet.)

At this point DEK decided to take a closer look at the sample deal from the second page of the Data Sheet. The first obvious thing to notice was the absence of any ♢'s. AM suggested that the 4♣ would never be the topmost club, because it is leftmost. After some discussion it was agreed that it may be improbable, but certainly not impossible without further analysis of the other cards. DEK then divided up the cards according to suit, noting matching ranks between suits.

♡ only:	6
♠ only:	7
♣ only:	J 8
♡ & ♣ :	Q T 9 4
♡ & ♠ :	A K
♣ & ♠ :	3

DEK introduced his student Pang Chen, and described Pang's method for estimating the number of nodes in a tree. We may construct a search tree for our present problem by assigning a nodes to positions. The root node is the initial 18-card position, and the children of this node will be all the 17-card positions that can be reached by a single legal move from the initial position. For example the initial configuration given in the data sheet allows 15 possible moves, and so the root node of the corresponding tree would have 15 children.

The trick that Pang uses is to identify nodes that are somehow "approximately the same" and thus simplify the counting process. We say that such similar nodes "have the same color" or "belong to the same stratum." The colors are ordered; all children of a node N must belong to a "lower" color than N does. At one extreme, all nodes on a particular level could be considered the same color, but this might give a poor estimate of the tree size. At the other extreme, all nodes could be considered a different color, which would mean we would end up traversing the whole tree.

In our case, we may identify the color of a node by the ordered pair (x, y), where x is the number of visible cards in the configuration, and y is the "mobility", i.e., the number of legal moves from this position. So, our initial position would have the color (18,15), and all its children would be of the form $(17, y)$, for various y.

Using this coloring system, we may fill out a 3-column table. There's one row for each color. The first column contains its "weight," and the second contains a typical example of a position with this color. The weight is intended to estimate the number of nodes in the tree with the given color.

There's also a third (boolean) column, which says whether or not this color has been expanded.[1] Initially, the table looks like this:

color	weight	example	expanded?
18,15	1	A♠7♠ ... K♡	no
all others	0	—	no

To update the table, we find an unexpanded row of non-zero weight having the highest color among all such rows. The example column of that row represents a node N of the tree. Suppose the weight of that row is w. Then for each child N' of N, in any order, we do the following:

> Let the color of N' correspond to a row with weight w'; change the example column in that row to N' with probability $\frac{w}{w+w'}$, using a random number generator. (Thus, if $w' = 0$ the example is always changed to N'; if $w' = w$, the example is changed half the time, so we could flip a coin; in general we can generate a random real number r in $[0, 1)$ and change the example to N' if $r < \frac{w}{w+w'}$.) Then replace the weight w' by $w + w'$.

Once this has been done for all children N' of N, mark the original row as "expanded." Proceed in this way until all rows are expanded. Pang Chen has proved (among other things) that the expected values of the weights in the resulting table are the actual numbers of nodes having a given color. So the total size of the tree can be estimated as the sum of all the weights.

DEK began to construct the table, flipping a coin when he got to the second node of color (17,14).

SM noted that we can't just flip a coin to determine whether to replace the typical example, as this would tend to favor nodes that were encountered towards the end. DEK agreed, saying that the actual probability rule $\frac{w}{w+w'}$ was the secret that made Pang's method work.

RC then suggested that the representation of the game as a tree may not be best, since many of the subtrees are actually identical. (You can get the same sequence of piles by making captures in different ways.) DEK replied that this was a good point, and that we could reduce the size of the problem by taking commutativity into account.

DEK had actually implemented Pang's method for this problem, and in 3 runs had obtained the estimates

$$\begin{array}{r} 678,206,500,000 \\ 1,454,101,000,000 \\ 1,335,456,000,000 \end{array}$$

for the size of the tree. These numbers were much too big for an exhaustive search.

DEK asked under what conditions we could ensure commutativity. If we knew two moves commute, we could insist on doing one of them first and thus prune our search space. AM

[1] The "expanded" column can actually be eliminated and replaced by a pointer to the "current" color, because all rows above the current row being expanded will have been expanded (unless their weight is zero). Thus, after the example node of the current color has been fully expanded, the current color is decreased to the next color whose weight is nonzero; then the example node of the new current color is expanded, etc.

suggested that, apart from the skip-two move, all moves commute. RC quickly refuted this with the example 4♣9♣9♡.

RK backtracked a little by asking whether we care about the positions of the non-visible cards. We agreed that the hidden cards do not affect winning in any way.

SM suggested that moves would commute if they were separate. DEK then introduced the notion of a "dividing line", to be placed somewhere in the sequence of cards. If one move lies entirely to the left of the line, and another entirely to the right, then the two moves commute. So, we can always choose to do one of them (say, the leftmost) first.

AM wondered whether a backward search rather than a forward search (starting with the final card and going to the two that preceeded it, then to three etc.) would have a smaller branching factor, but this line of discussion was not pursued.

The dividing line approach to commutativity implies that we can assume every sequence of moves has the following property: *The card that makes a capture equals, or lies to the right of the card that made the previous capture.* Any sequence of captures that doesn't follow this rule can be converted by commutativity into a sequence that does, because we can do the leftmost captures first when they don't overlap. DEK said he tried Pang's method with this variant, and he obtained three estimates of tree size:

$$\begin{array}{r} 157,960,849 \\ 99,217,842 \\ 330,153,554 \end{array}$$

The interesting thing about the second estimate was that the method actually found a winning sequence of moves.

So it seems that commutativity saves about a factor of 5000. We should actually be careful here, as not all commutativity has been considered, for example the commutativity in 4♣9♣Q♣.

What is needed is more heuristics to cut off the search early. For example, in one of the above searches there is a node with color (8,1), and weight 1,824,580 with the position 7♠3♠3♣Q♣4♡J♣9♣K♡. For this position, the hearts cannot be combined with the rest of the cards, since there are no other 4's or K's, and so there is no need to explore further. It should be possible to get the search tree down to size 1,000,000 or perhaps even thousands.

It was suggested by DP that we make more complex moves, by composing two ordinary moves. That way, the tree would have half the depth. DEK pointed out that this could square the branching factor, and the likelihood of any gain is unclear.

RC commented that all cases of commutativity could be recognised if we kept track of every position we'd seen before. There may be too many of these, but we could perhaps get most of the benefit by keeping a cached hash table of some sort.

Tuesday January 17

The class began with DS revealing a solution to the configuration in the data sheet. His solution is as follows (using the terminology of the data sheet)

3♠×7♠×A♠, K♡×K♠, 6♡××10♡, 9♣×8♣×J♣, 3♣×Q ♣××10♣×4♣×3♠, A♡×6♡×4♡, 9♣××3♣, 9♡×9♣, K♡×Q♡×A♡×9♡

The idea behind this solution was to get the problem down to two suits, by wiping out the spades as soon as possible. With only two suits, there is a better chance of a win. DS was the only one to come up with a winning sequence; SM said that he tried doing it in his head, but by the time there were only a few piles left he had forgotten the moves he used.

MA then approached the blackboard and outlined a tree construction method for which *all* non-interacting moves are performed simultaneously. This would lead to a higher branching factor - he stated that in the worst case there could be 9! children of a node. DEK pointed out that in the previous class we had considered eliminating certain move orders on the basis of commutativity. MA's method does essentially the same thing, but with a less simple control mechanism.

ESC said that our example above suggested that it is not sufficient to consider the first four or so cards alone; one has to look at all the cards. DEK then posed a question:[2] Can we deal four cards that will allow us to conclude that no win exists, no matter what cards come after?

"Extremal examples" such as $2\heartsuit 3\clubsuit 4\spadesuit 5\diamondsuit$ and $2\heartsuit 2\clubsuit 2\spadesuit 2\diamondsuit$ were suggested, but it was quickly realized that these would not suffice, unless further restrictions were placed on the cards that appeared after the initial four. DEK suggested $2\heartsuit 2\clubsuit 2\spadesuit 2\diamondsuit$ with all subsequent cards being hearts. When it was pointed out that there were not enough hearts to make a total of 18 cards, DEK suggested allowing either hearts or clubs. Is it possible to win then?

At first we thought it would be impossible. But then DK came up with a winning idea. Suppose we get down to $2\heartsuit 2\clubsuit 2\spadesuit 2\diamondsuit 3\heartsuit 3\clubsuit$. We can then win after $2\diamondsuit\times 2\spadesuit$, $3\heartsuit\times\times 2\heartsuit$, $3\clubsuit\times\times 3\heartsuit$.

KAR asked what is the minimum number n of cards such that there is an initial sequence of length n for which no win exists for any continuation of this sequence. The group convinced itself that $n \le 17$, and it seemed that $n \le 15$ might be provable (because some configurations appeared "penetrable" at most 5 places from the right); but this question was not pursued further.

The discussion meandered along until SJP changed the subject by announcing that he had implemented Pang's method with the following additional heuristic: Prune a node for which the "suit/rank graph" is disconnected. DEK reviewed the heuristics for early pruning discussed in the previous class. DK pointed out that even if we know a node does not "work," we haven't *proved* anything since we are dealing only with a single example from the class of all nodes having a given color. DEK agreed, saying that this was only a heuristic applied to a statistical process. One makes certain simplifying assumptions about how representative the examples are of their respective classes, just as the Gallup poll tries to infer things about millions of people after interviewing only a few thousand.

SJP stated that the "suit" graph (formed by associating nodes of the same suit) is connected if and only if the "rank" graph (formed by associating nodes of the same rank) is also connected. This is because both graphs are "compressions" (of cliques) of the "suit/rank" graph, the graph in which no groups of cards are identified. SJP observed a reduction in the size of the tree by a factor of 10, which was less than he had hoped (since the time to test connectedness might increase the computation per node by more than a factor of 10). On the brighter side, Pang's method seems to find a solution (for the first

[2] "Research is the art of asking questions." — DEK.

problem) approximately one third of the time.

DEK had done a similar thing; the estimated tree size was still about 40 million. After 200 tries of Pang's method, 44 of them had turned up a solution.

Having considered some necessary conditions for losing, DEK asked whether there are any necessary conditions for winning. PL suggested a Hamiltonian path in the suit/rank graph. "Not quite," said DEK, and gave the example of A♡×K♡ followed by A♡×Q♡. UH suggested the simple condition of whether all remaining cards are of the same suit. Thus, we can stop as soon as there's a suit with no connections to other suits. (Either this suit is the only suit left, and we win; or the graph is disconnected and we lose.)

SJP then suggested looking for a spanning tree for the suit/rank graph in which each child is captured by its parent. DEK then proceeded to draw the spanning tree corresponding to DS's earlier solution. The tree can be described in LISP-like fashion thus:

> (K♡ (K♠ Q♡ A♡ (6♡ (10♡) 4♡) 9♡ (9♣ (8♣ J♣ 3♣ (Q♣ 10♣ 4♣ 3♠ (7♠ A♠))))))

AG pointed out that every connected suit/rank graph has such a spanning tree, so the existence of a spanning tree per se didn't tell us anything we didn't already know. DEK then described a refinement of the method in which cards are assigned a subscript corresponding to the position (from left to right) in which they start. The tree given above then becomes

> ($K♡_{18}$ ($K♠_{17}$ $Q♡_{16}$ $A♡_{11}$ ($6♡_{13}$ ($10♡_{10}$) $4♡_{7}$) $9♡_{6}$ ($9♣_{15}$ ($8♣_{14}$ $J♣_{12}$ $3♣_{9}$ ($Q♣_{8}$ $10♣_{5}$ $4♣_{4}$ $3♠_{3}$ ($7♠_{2}$ $A♠_{1}$))))))

DS conjectured that every child of a node must have a smaller subscript than the node itself. This hypothesis did not strictly hold, as demonstrated by the 9♡ and 9♣ in the constructed tree.

Although it seemed not to work, DEK said that the idea was good. After all, we do capture from right to left. To patch up the conjecture, we can say that the "value" of the root of a subtree is the minimum value of all its descendants. Alternatively, we could require the capturing card to inherit the value of the captured card. Then each card must have an initial subscript higher than the values of its children.

With the extra value information, we can find connected suit/rank graphs that may be proved unsolvable. The configuration ...8♣9♣...8♢9♠... (in which all the cards not shown are clubs) is such a case.

Here's the proof: Since the only card which matches the 9♠ is the 9♣, the tree must contain the 9♠ as either a parent or child of the 9♣. The second alternative, in which the 9♠ is a child of the 9♣ is impossible, according to our restriction on node values. Hence the 9♠ is a parent of the 9♣, and the 9♠ is the root of the tree. A similar analysis holds for the 8♢ and 8♣, demonstrating that the 8♢ is also the root of the tree! This contradiction demonstrates the non-existence of a win.

This tree condition is a necessary, but not sufficient condition for our game. It is actually a necessary and sufficient condition for a different game: "Skip-any Solitaire." (A question that DEK posed to be answered off-line was whether there exist any simpler necessary and sufficient conditions for the skip-any game.) The connectedness condition is necessary and sufficient for "Skip-any both-ways Solitaire."

DEK described an observation he made when using Pang's method on these problems. Big search trees tend to have solutions; conversely, small search trees, which tend not to have solutions, are small enough to be exhaustively searched. SM suggested looking at the highest mobility children first, in order to get to a solution fastest.

UH wondered what advantage Pang's method had over exhaustive search, i.e., what was "magical" about the method. DEK's response was that we don't know exactly why, but Pang's method is very good at finding different parts of a tree, and described its success on a chess problem that he had previously spent many CPU hours on. In a search tree with more than 10^{10} nodes, only 6 of which were on level 36, Pang's method found one of these on its third try (!).

DEK finished by asking how one can check connectedness efficiently. Standard garbage collection type algorithms could probably be improved upon by taking into account the particular structure of this problem.

Thursday January 19

ESC started the discussion by suggesting that we use a table to store the configurations with, say, 7 cards in them. Whenever we get down to 7 cards, we then simply look up this table "in constant time." He mentioned in passing that there would be about 150,000 entries in such a table.

RC objected to this procedure on the grounds that when looking for a single solution we are only going to see a very small number of positions.

DEK picked up on the figure of 150,000, and suggested that we would have to calculate the number as $18 \times 17 \times \ldots \times 12 \approx 1.6 \times 10^8$. (We only use the cards from the initial 18, choosing seven in any order.)

PL made the comment that there are many symmetries that we could exploit to save time. At this point ESC revealed that he had already taken into account such automorphisms when calculating this number 150,000. DEK then said that you can reduce to considering graphs on seven vertices, with nodes adjacent iff they can capture each other. But this still apparently requires $2^{\binom{7}{2}}$ slots.

ESC described how he calculated the number 150,000. He first chose a card, and labelled it with a variable for its rank and a variable for its color. Say the first card is R_1C_1. Then the second card can be one of R_1C_2, R_2C_1 or R_2C_2, and we may continue this process for subsequent cards. The actual number of non-isomorphic configurations obtained for various sizes are given in the following table.

1	1
2	3
3	15
4	113
5	1101
6	12657
7	162863
8	2.28×10^6
9	3.42×10^7
10	5.46×10^8
$\vdots$	$\vdots$

Based on the (seemingly) manageable value of 162,863,[3] ESC had chosen seven-card positions for his lookup table. DEK realized that his earlier figure was an overestimate because he had not taken into account that there are only four suits.

DEK then returned to ESC's earlier statement about using "constant time" to look up the table. What he really meant was "a short time." One has to be careful when using the $O(1)$ notation that the context is clear. Everything we will see in our lifetimes will take O(1) seconds. Even the U.S. National Debt is O(1) dollars, although we don't have enough time to examine a search tree that big.

RC wondered whether it was worth all the effort to reduce our tree to a dag (directed acyclic graph) by identifying all equivalent nodes.[4] It was unclear to him whether there was much more "fat to trim" from our search. He suggested a variant of Pang's method in which we keep track of the number of parents of a node in order to help the search.

DEK then asked how one may determine the parents of a given configuration. RC said that one would need to know the original configuration. RK pointed out that in this framework we would have to distinguish positions on the basis of the hidden cards. RC still thought it was a nice idea, and accepted DEK's suggestion to program it on a smaller problem to see how it fares.

SJP changed the subject by suggesting replacement of the table of strata by a hash table to determine the amount of duplication. Doing this for, say, 10-card configurations might give an idea of how much commutativity is left.

DEK then commented on an idea that AH had brought up after the previous class. His suggestion was to use as a heuristic the prompt elimination of a whole suit; perhaps it wouldn't even matter which suit is eliminated first. DEK could not solve the problem from the data sheet by eliminating clubs first, but he did manage to find a solution whose final card was the K♠.

AG then reported on the results of a program he wrote to find solutions. He described the program as a simple depth first search, and observed that it solved between a third and a half of the configurations given in the Data Sheet. For example, its solution to number 20 was

> Q♣××J♣, A♡×9♡, 6♡××A♡, 4♠××4♡, A♣××Q♣, 8♠×4♠, 6♢××6♡, K♣××A♣, J♠××8♠, 3♠××J♠, 2♢××6♢, K♠××3♠, K♢×K♠×2♢, 7♣××7♡, K♣×7♣, K♢×K♣.

DEK seemed relieved that some of the random deals did in fact have solutions; he had not been sure what the chances of winning would be, or whether the first (constructed) deal was just a fluke.

DEK then turned the discussion to the topic of connectedness. SJP began to describe a method that he and RC had come up with independently. He suggested representing cards as a bitmap, in which the third bit would be set, for example, if the card was a three. There would be thirteen bits for cards, and four for suits, giving the (unfortunate) total of 17 bits. Now one card can capture another if and only if their bitmaps have a nonzero intersection.

---

[3]DEK said "That number looks familiar ...," and proceeded to factor 162,864 into $2^4 \cdot 3^3 \cdot 13 \cdot 29$. He then remarked "I guess not." RK remarked that $13 \cdot 29 = 377$ is $2^8 - 1$ in octal ....

[4]As observed by DQ, we do not have a general dag, but a variety in which nodes are on successive strata. DEK compared this feature to a similar feature of modular lattices.

In order to get a representation of the cards in a suit, all the cards of the same suit may be combined together using a logical **or**. By comparing the bitmaps for suits, using a logical **and**, one can determine connectedness using at most six comparisons.

DEK then asked if it would be possible to optimize further by only considering those comparisons that are relevant given the suit and rank of the card just captured. This may be a significant saving, if this component of the program is in the "inner loop."

He went on to describe how "if-then" comparisons tend to be the bottleneck in parallel machines, as processing has to be held up until the result of the comparison is known. He told the story of how he wrote a merge sorting routine for one of the early Cray computers, which employed a pipelined architecture. An ordinary merge sorting routine would have to do three comparisons: one on the keys to be compared, and two to determine if the input buffers had been exhausted. When such a routine was run, there were fast floating point processors sitting idle. He rewrote the program so that in parallel with normal computation those processors were multiplying differences between the buffer pointers and manipulating the variables to give a result that was either zero or nonzero. If nonzero, which was the normal case, the loop could be executed once more without further tests. Otherwise, some other section of code was executed. Since this other section did not have to be executed very often, he gained a factor of almost three in performance.

AG then described how he tested connectivity. He used a pre-compiled suit graph, and maintained counters for the number of links between suits, a counter for each suit, and one overall counter. Connectivity holds then if the following two conditions are true:

- Every suit has degree at least 1.
- If there are n suits then there are at least $n-1$ links.

AG stated that it was easy to maintain these counters, and the graph did not have to be altered at each step. (The condition can fail when $n=5$, but we have $n \leq 4$.)

AM disagreed with the statement that the graph did not have to be changed at each step, giving the following example. Suppose there are three sixes, the $6\diamondsuit$, $6\heartsuit$, and $6\spadesuit$. Suppose that the first move is $7\spadesuit \times 6\spadesuit$ which should decrease the spade link count by 2. If the second move is $5\heartsuit \times 6\heartsuit$ then, unless we have modified our graph, we will further decrement the spade link count. This is not sound, as by this stage the $6\heartsuit$ has no connection with the spades. AG was convinced by this argument, and concluded that things were more complex than he first thought.

SM described how he maintained a connectivity graph using four types of edges. An edge was labeled with a two digit binary number; 00 signified that neither suit had a card of this rank, 11 signified that both suits had a card of this rank, 10 and 01 signified that only the first (or second, respectively) suit had a card of this rank. Such a mechanism makes it easy to update the graph. He also maintained counts of the links. AG said that you could get away with fewer counts by taking advantage of his observations mentioned above.[5]

DEK then said it may be better to check for an isolated suit, rather than for connectedness. The condition is weaker; but maybe it is strong enough, in the sense that the extra tests for connectedness may cost more than they gain.

SM said that his different types of link allow us to test just that.

[5] "If my method had worked, it would have been faster . . . ," - AG

At the end of class, a delegation lobbied DEK to have the deadline for Problem 1 extended, since the applications comprehensive exam will take place the night before. After careful consideration of the alternatives, it was decided to move the due date back one week, but to otherwise leave the schedule as it was. There will therefore be a five-day period in which we'll be working on both Problems 1 and 2.

Tuesday January 24

DEK entered, and in show-and-tell style asked "What did you all learn over the weekend?"

AG, who did not have the comps to worry about, described how he did some further commutativity testing, which reduced the size of the tree by about 25%. It was quite easy to implement, with additional gains for configurations of the type 2♣3♣4♣ where all cards are of the same suit or the same rank. For example, from ...2♣3♣4♣... one can move 3♣×2♣ followed by 4♣×3♣, or 4♣×3♣ followed by 4♣×2♣ with the same result. We eliminate, say, the second possibility from consideration. This type of commutativity is not eliminated by the dividing line approach. He also was able to avoid extra work in cases like 2♣3♣J♡K♡4♣.

DEK then asked whether AG had considered positions like 2♣J♡K♡3♣4♣ in which the commutativity involved a "dual" skip-two move. AG said that he hadn't, but that it would be easy to implement along the same lines as in the previous case.

RK then wondered whether you could make similar observations with skip two on both sides, for instance on the clubs in the configuration 2♣J♡K♢3♣Q♠A♡4♣. It was quickly realized that it doesn't work in this case, as once the 3♣ takes the 2♣, the 4♣ can't take the 3♣. RK then said that it was a stupid idea.[6]

AG then pointed out an additional feature of his approach, which is illustrated by the following example. Remember that with the configuration 2♣3♣4♣...8♣ we ruled out the sequence of moves 4♣×3♣, 4♣×2♣. Suppose that we play 4♣×3♣, and then make several other moves, whose net effect is to capture the 4♣ with the 8♣. Then we are still not permitted to play 8♣×2♣ for the same reasons as before. AG implemented this by maintaining a table of bits, one for each card, saying whether capture of the left neighbor was permitted.

DEK then demonstrated another case in which there were two equivalent ways to reach the same position. Consider the position $abcdef$, where the following captures are possible: $f \times\times c$, $e \times\times b$, and $d \times\times a$. Then the sequence of moves (as just given) is equivalent to the sequence $e \times\times b$, $d \times\times a$, followed by $f \times c$. DEK thought that this observation wouldn't save very much time, as it was one uncommon case, among (not too many) such equivalences.[7]

SM, who also hadn't been bothered by comps, then described his experience searching for solutions to the deals in the Data Sheet. Without the dividing line, his program could not solve the problem in hours. With the dividing line, the program solved all the examples in thirty to forty minutes. Augmenting his program with a test for isolated suits resulted in a running time of 2 minutes to solve all of the examples. SM mentioned that he tried rightmost moves first in his dividing line strategy, since these moves seemed to affect other moves the least. The class speculated that this was largely responsible for the fast running

[6] DEK objected to calling such ideas stupid — it is part of the normal problem solving process to make conjectures that don't pan out. RK replied that "stupid" was not such a perjorative term. DEK then asked what was; "brain damaged"?

[7] "A good idea can save you several orders of magnitude. Other good ideas can cost you." — DEK

time of SM's program.

SM's results are as follows: the deals which have wins are 1, 3, 4, 5, 6, 8, 16, 18, 19, 20, 21, 22, 24. (DEK had found all but 18, 20, 22 and 24 using 100 repetitions of Pang's algorithm.) For the deals without solutions, it seemed that Pang's method had estimated the size of the tree quite well. The following table summarizes this data.

Deal Number	Actual Size (SM)	Pang's method estimate (DEK)
2	56K	43K
7	265K	261K
9	579K	553K
10	93	93
11	0	15K*
12	79	89
13	3.8K	5K
14	447K	481K
15	60K	64K
17	30K	27K
23	5K	6K

Deal 11 is interesting because it has a card that's disconnected from the rest, and so it's easily seen to be unsolvable. SM's programs checked for disconnectivity at the start, and after each iteration; DEK's only checked after each iteration, and then only on the rank/suit involved in the capture. Hence DEK missed the early cutoff and estimated a large tree size.

DEK then described how, back in the good old days of punched cards, he once constructed a program to solve the original solitaire puzzle.[8] After having waited all night for the machine to punch out an answer, DEK stopped it at 8am to find it hadn't backtracked past the first 24 moves. He later calculated that it would have taken 35,000 centuries to run to completion. The moral of this story is that instrumenting a program is a good idea, to obtain knowledge on how it is performing.

RK then returned to the apparently good estimates given by Pang's method, and pointed out that we don't know how many nodes it actually expanded, and so it could have seen quite a lot in 100 iterations. (The estimates above were averages over 100 runs of Pang's method.) DEK concurred, saying he wished he had time to instrument his program better. Other statistics he wanted to see were the estimates given by Pang's method for the number of nodes on each level of the tree, not just the total. The level by level statistics typically have an approximate "log-normal" distribution; this means that the number of digits in these numbers tends to make a bell-shaped curve.

DEK also pointed out one interesting point about methods that are proved to give the correct expected value. He described how an algorithm could generate many underestimates, and with a very small probability generate a huge estimate. This would give the correct expected value, but would have a large variance. Pang's method has a relatively small variance, compared to DEK's earlier method in which all nodes at the same level had the same color.

[8]The one with the pegs in which the aim is to capture the pegs one by one, leaving only one peg at the end.

SM announced that of the fifty random deals in the supplied data file, twenty-four were wins. So it looks as if the game can be won about half the time. DEK expressed disappointment that he never seemed to win nearly that often when he tried it by hand. AM remarked that he had played the first deal for two days, and still didn't find a win!

DEK then returned to the "skip-any" version of the game, in which moves still had to be from right to left, but any number of cards could be skipped. The simple connectivity test is necessary but not sufficient. But in the two-suit case, connectivity is sufficient except when there's only one link between the suits, and one of the link cards is the leftmost, and that card isn't the only one of its suit. For example, you can't win from 4♠...4♣... when the cards not displayed are all clubs or spades, with at least one spade, and with no two cards of the same rank.

RC suggested a strategy of choosing "destination piles" for each suit, and then moving "connectors" onto these piles at the last minute. The connectors could then themselves be joined.

DEK wondered whether one could reconstruct from the final piles how the game had been played. It was agreed that this was not possible. DEK then described another problem in which all cards had the same suit; he tried to count how many possible orderings would occur in the deck after the rules of skip-two solitaire were used to reduce everything to one pile in all possible ways. (For example, starting with A♢2♢3♢4♢5♢ you can get five final orderings, two of which have the 3♢ on top.) He was unable, though, to solve the recurrence. He justified looking for answers to such esoteric problems on the grounds that he's always looking for instructive questions to put on exams.

In solving this problem, most of the groups had written programs to do an exhaustive search of the non-redundant parts of the game tree. SM and DS ran their program (which was judged by KAR to be the fastest) for deals from 2 cards to 35 cards. Their results were as follows.

Cards	Deals	Chance of Winning
2	2000	29.7 %
3	2000	9.9 %
4	2000	6.3 %
5	2000	4.7 %
6	2000	3.4 %
7	2000	3.2 %
8	2000	4.1 %
9	2000	4.8 %
10	1000	4.8 %
11	1000	8.2 %
12	1000	9.0 %
13	1000	12.6 %
14	1000	16.3 %
15	1000	21.2 %
16	1000	26.9 %
17	1000	38.1 %
18	1000	45.4 %
19	500	55.6 %
20	500	65.4 %
21	500	73.4 %
22	500	83.6 %
23	500	88.4 %
24	500	93.0 %
25	200	95.5 %
26	200	97.0 %
27	200	99.5 %
28	200	100 %
29	200	100 %
30	200	100 %
31	100	100 %
32	100	100 %
33	100	100 %
34	100	100 %
35	100	100 %

Given this information, DEK's choice of 18 cards seems remarkable. DEK maintains, though, that it was only the appealing property $18 + 17 + 17 = 52$ that influenced this choice.

Chapter 3

Toetjes Revisited

Tuesday January 31

Before the start of class, DEK presented certificates, typeset with a historic system called POX, to the trivia hunt participants. He noted that this may well be the last time that POX would ever be used, since it will disappear when the machine SAIL is decommissioned.

Having spent the previous Thursday discussing the trivia hunt solutions, today was the first occasion that Problem 2 was discussed. It was pointed out that the references in the Data Sheet did not have matching entries in the bibliography (and in fact that there was no bibliography). DEK told the class that [1] refers to Tech Report STAN-CS-87-1154, while [2] refers to Tech Report STAN-CS-88-1208. The latter of these was written by Tomás Feder, who was present in class at DEK's request.

DEK then presented the notation used in the Data Sheet. The function $f(x_1, \ldots, x_k)$ represents the optimum move for the $k+1^{st}$ player. The random variable $g(x_1, \ldots, x_k)$ takes values that are "optimal continuations." We need to include a permutation π in the continuation to be able to tell which of two (or more) choices at the same point in the interval is greater.

This notation is obtained by taking the limit of guesses arbitrarily close to a certain number x. Consider, for example, the two player game in which the first player chooses the point $\frac{1}{2}$. The second player would like to choose either $\frac{1}{2}+\epsilon$ or $\frac{1}{2}-\epsilon$. The smaller ϵ, the better the move; however we cannot let $\epsilon = 0$, since $\frac{1}{2}$ is already taken. Hence we take the limit as $\epsilon \to 0$, preserving the relationship

$$\lim_{\epsilon\to 0}\left(\frac{1}{2}-\epsilon\right) \le \frac{1}{2} \le \lim_{\epsilon\to 0}\left(\frac{1}{2}+\epsilon\right).$$

This generalizes to the many player case. DEK stated that we shouldn't worry too much about the epsilons. Tomás pointed out that the intuitive results are obtained by taking these limits "last player first."

AM wondered whether there would ever be a situation in which a player was "boxed in," i.e., had players choosing numbers ϵ below and ϵ above his choice. DEK said that we couldn't discount this possibility at this early stage, but that it could well be possible that in an optimal game, boxing in does not occur. It certainly does happen, though, that optimal games have two players choosing points ϵ apart.

DEK then started on an extended example.

In the above scenario, we can calculate the payoff for each player as follows:

$$\begin{array}{lcll} P_1 & = & \frac{x_1+x_2}{2} & = 0.25 \\ P_2 & = & \frac{x_3-x_1}{2} & = 0.25 \\ P_3 & = & \frac{x_4-x_2}{2} & = 0.3 \\ P_4 & = & 1-\frac{x_3+x_4}{2} & = 0.2 \end{array}$$

In general, with n players, moving at $x_1 < x_2 < \ldots < x_n$, we have

$$\begin{array}{lcll} P_1 & = & \frac{x_1+x_2}{2} & \\ P_k & = & \frac{x_{k+1}-x_{k+1}}{2} & 1 < k < n \\ P_n & = & 1-\frac{x_{n-1}+x_n}{2} & \end{array}$$

RK observed that the formula for $1 < k < n$ would work also for $k = 1$ and $k = n$ if we imagined "mirror image" players who have chosen $x_0 = -x_1$ and $x_{n+1} = 2 - x_n$.

Now suppose that we have one extra player, who must choose a number given the configuration above. Player 5 can seem to get a payoff of 0.2 in two different ways: anywhere in the interval (0.3..0.7), or by choosing arbitrarily close to 0.2 (approaching from below). PL and RK pointed out that the latter of these alternatives can be discounted since the problem statement forces us to choose numbers uniformly over all optimal numbers, and $\{0.2\}$ is a set of measure zero.

DEK added that there was another reason to avoid the second choice above. The second alternative involved choosing a point arbitrarily close to 0.2. But no matter how close we get (except in the limit), this choice will be strictly worse than the first alternative. Thus DEK proposed to allow the choice of "limit moves" only if all the moves with the optimal payoff are limit moves.

DEK suggested that an infinite set of optimal points could be represented as a finite union of disjoint intervals. He said open intervals would suffice, since the endpoints could only occur with probability 0.

In the example above, the random function g may be represented as the integral

$$g(0.2, 0.3, 0.7, 0.9) = \int_{0.3}^{0.7} \langle t; 12534 \rangle \frac{dt}{0.4}$$

Based on this move, the second player may estimate his payoff as

$$\int_{0.3}^{0.7} \left(\frac{t-0.2}{2}\right) \frac{dt}{0.4} = 0.15$$

If players 3 and 4 had swapped their choices (i.e., player 3 chooses 0.9 and player 4 chooses 0.7) then the payoff for player 4 would be

$$\int_{0.3}^{0.7} \left(\frac{0.9-t}{2}\right) \frac{dt}{0.4} = 0.2$$

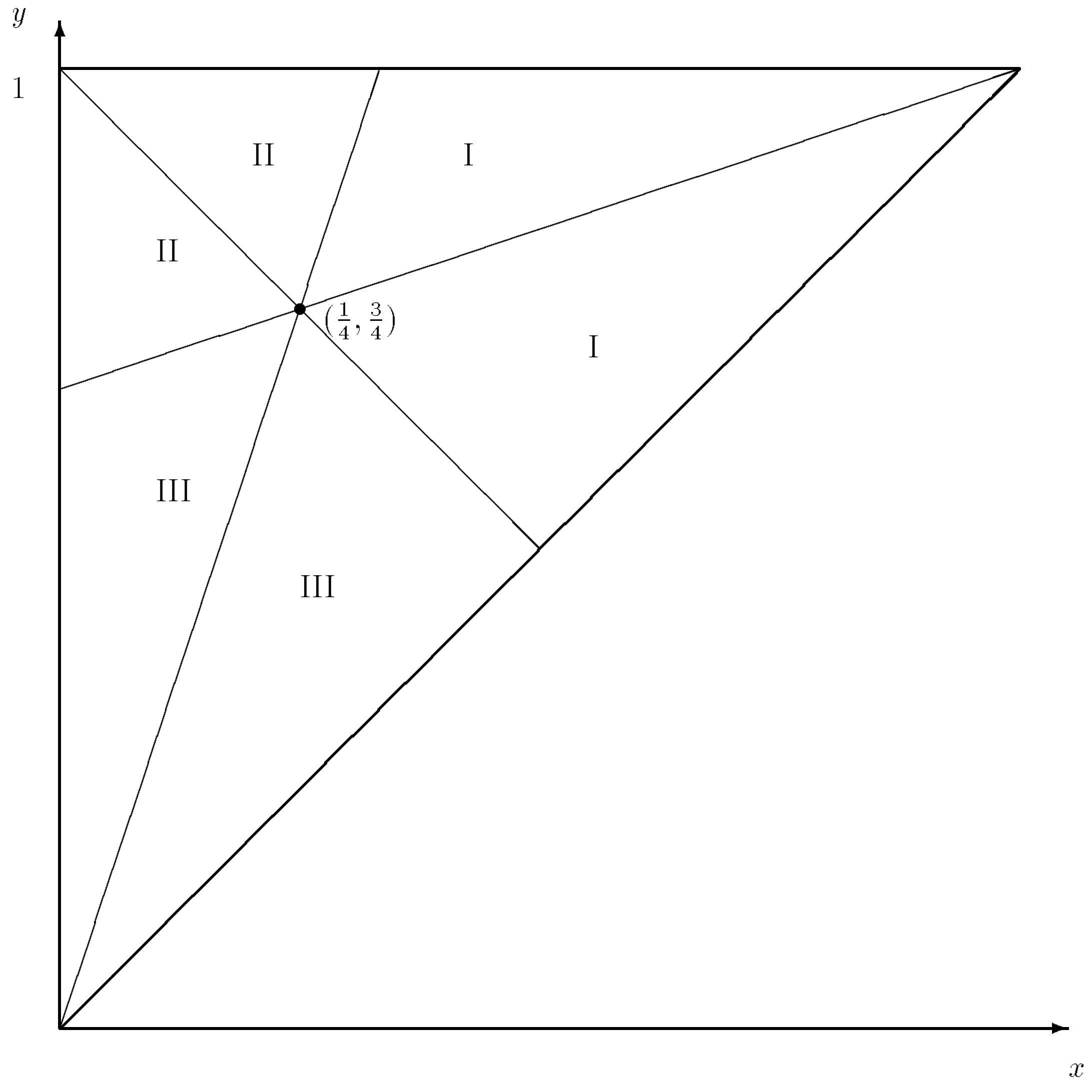
In both cases the integral is evaluated by considering it at the midpoint 0.5, since the integral is linear.

RC suggested that we may not need such a complex representation of f and g as integrals. He said that based on a preliminary analysis of the problem, f is a piecewise linear function. RK pointed out that this function need not be continuous.

DEK went on to describe the analysis of the three person game in Appendix 2. He mentioned in passing that the order of the previous moves does not affect the choice of the current player. Suppose player 1 plays at x, and player 2 at y. Since the order is not important, we assume without loss of generality that $x \leq y$. Suppose that player 3's choice is denoted by z. Then the payoffs for various moves are summarized in the following table.

Region	Range	Best Choice	Payoff
I:	$z < x$	$x - \epsilon$	$x - \frac{\epsilon}{2}$
II:	$x < z < y$	Any	$\frac{y-x}{2}$
III:	$y < z$	$y + \epsilon$	$1 - y - \frac{\epsilon}{2}$

We may then compare the relative merits of each of these moves based on the actual values of x and y. It is easily shown that region II is better than region I precisely when $y > 3x$; when $y < 3x$, region I is better. Similarly, region I is better than region III when $y > 1 - x$, and region III is better than region I when $y < 1 - x$. Finally, region II is better than region III when $y < \frac{2+x}{3}$, and region III is better than region II when $y > \frac{2+x}{3}$. This information is summarized in the following diagram.



The enclosed areas of this graph are labelled with the best region for player 3. AM wondered whether we had to consider whole areas, or whether we could restrict ourselves to only consider "corners." Tomás replied that you do need the full generality of areas, since every move can be optimal in some regions.

DEK then suggested reviewing (or learning) a little linear programming. His favorite book is one by Dantzig.

PL reported that he had analysed the four player game, and had calculated a payoff of $\frac{7}{24}$ for the first player. DEK said he was looking forward to going through the four player game next class, and he wished everybody fun playing with inequalities.

The groups for Problem 2 are:

- PL, DQ, DS
- RC, UH, RK
- SM, AM, SJP, MY
- ESC, AG, ADG, DK

Thursday February 2

The class started with PL describing his analysis of the four-player game. He said that the first player should choose to play at $\frac{1}{6}$ or $\frac{5}{6}$; the second player should then choose the other. The third player should then choose $\frac{1}{2}$, with the fourth player playing in either of the two intervals of length $\frac{1}{3}$. This gives player 1 a payoff of $\frac{7}{24}$.

PL conjectured that in the five-player game, the first player should play at $\frac{1}{8}$, and said that he was in the process of writing a program to handle the five-player game. DQ went further, and conjectured a formula for the general case. Suppose there are n players. Then the first player should play at $\frac{1}{2n-2}$, which we denote by x. The second player should play at $1-x$ and the third at $3x$.

At this point, DEK paused to emphasize the need for proofs of these conjectures.[1] It is appealing to suggest symmetric strategies simply by following one's intuition; stronger arguments are required.

DQ thought that, apart from the last player, there would probably be only one or two optimal moves. DEK replied that such an observation would make a program to solve the problem easier to write.

SJP then mentioned that Tomás had analysed a version of the game where the ends were already taken.[2] Hence his analysis of the case $n = 7$, when the random-choice rule seemed to diverge from the deterministic rules considered in his paper, doesn't hold for the version we are looking at.

RK discussed his analysis of the four-player game, commenting that some of the inequalitites that are generated are redundant, i.e., they are implied by other inequalities.

DEK said that his intuition at this point was to go general rather than to consider a number of special cases. He highlighted two subproblems to finesse: handling symbolic inequalities and symbolic integration.

PL suggested that by choosing "representative" points in an interval, one could avoid the need for symbolic integration. DEK said that the crucial situation to avoid is the case where there are two optimal intervals of different lengths. If this situation occurs, then we need to evaluate an integral of the form

$$\frac{\int_a^b \cdots + \int_c^d \cdots}{(b-a)+(d-c)}.$$

This would lead to problems since a, b, c and d would be variables from higher level integrals. But he suspects that we can get by with cases where the inteval sizes are commensurable with each other.

At this point, AG said that in "real life" there would be interaction amongst the children, such as a spiteful move by a later player made to punish an earlier player. DEK then compared this to the game of Risk, in which the optimal strategy seems to be to wait until most of the opposing forces have annihilated each other, and then to pounce.[3]

[1] He actually conjectured that computer scientists recognised different standards of proof twenty years ago, although he didn't offer a proof of this.

[2] In fact there were a number of different variations on the game considered in CS304 two years ago, such as playing on a ring rather than an interval. See that technical report for further details.

[3] The discussion then degenerated into various strategies for playing Risk. It was amusing to see how many avid Risk players there were in the class.

SJP then brought up an interesting example, which was based on the notes for CS304 two years ago. Consider a five player game, in which the first two players play at $\frac{1}{8}$ and $\frac{7}{8}$. One may suggest (as does DQ above) that the third player play at $\frac{3}{8}$. After this move, player 4's best move is at $\frac{5}{8}$, and player 5 should play in any of the three intervals of size $\frac{1}{4}$. This gives player 3 a payoff of $\frac{5}{24}$.

However, consider the situation if player 3 plays at $\frac{3}{8} - \epsilon$. Player 4's best move is now at $\frac{5}{8} - \frac{\epsilon}{2}$. Player 5 will now play in one of the two intervals of length $\frac{1}{4} + \frac{\epsilon}{2}$. The important difference, as far as player 3 is concerned, is that when he plays exactly at $\frac{3}{8}$, there is a $\frac{2}{3}$ probability that player 5 will take some of his space; but when he plays at $\frac{3}{8} - \epsilon$ the probability drops to $\frac{1}{2}$. The payoff for player 3 in the latter case is $\frac{7}{32} - \frac{5\epsilon}{16}$.[4]

Note that we *don't* want to take ϵ all the way to the limit here, because at the limit the payoff is worse. In the discrete case, say choosing integers between 1 and 1000, player 3 would want to choose the smallest possible non-zero ϵ. For example, if the first two players choose 125 and 875, then player 3 should choose 374.

KAR pointed out that if RC's intuition about the payoff being a (possibly discontinuous) picewise linear function was correct, then the case illustrated above may be visualized as a line segment in the graph of the payoff function whose higher endpoint is a "hole." A simple example of this is the function f defined by

$$f(x) = \begin{cases} x & x < \frac{1}{2} \\ \frac{1}{4} & x \geq \frac{1}{2} \end{cases}$$

RC suggested using a symbolic language such as LISP to write a recursive program, and hope that you can get the answers that way. He thought that this was more promising that using C, for example.

RC then asked a question about handling the max function in linear programming. DEK suggested the following: Suppose we want the constraint $a < \max(f, g)$. Then we add an auxiliary function h, and new constraints $a < h$, $h \geq f$, $h \geq g$. We then minimize h in the objective function.

So suppose we are trying to minimize $3a + 4b$. Then after adding h as above, we now try to minimize $3a + 4b + \infty h$. The coefficient of ∞ ensures that h is minimized ahead of $3a + 4b$.

PL gave a reference to a paper by Hodes in IJCAI 1971, describing methods to handle such inequalities. DEK pointed out that various algorithms perform differently on different applications, and so one cannot expect to have a method that is best for all problems. He referred to his "Sorting and Searching" volume, in which he included 25 different sorting methods because no one of them was completely dominated by any other for all applications.

Tuesday February 7

DEK opened the class by asking what the consensus was on the Toetjes problem. Silence.

DEK proceeded from group to group to find out what they had done. It soon became clear that the problem was very hard, despite its apparently simple formulation.

ESC described his group's attempt at formulating an algorithm, although they had nothing that worked.

[4] Note that Tomás considers a version of the game in which, given intervals of equal length, a player chooses to play nearer the earliest players, and so this type of situation does not arise.

DQ and PL described a program their group had written which worked for $n = 2$, and for $n = 3$ (modulo some "junk"), but did not work (yet) for larger n.

UH described how his group got part way to writing a LISP program to generate the whole game tree.

SM and SJP described how their group attempted to write a program to solve the problem for general n. Their main obstacle was formulating ϵ-moves symbolically.

The g function in the problem statement didn't preserve enough information. It gave the limiting value, but did not discriminate between cases where the limit is never actually achieved.

Namely, suppose $g(x_1, \ldots, x_k) = \langle x_{k+1}, \ldots, x_n; \pi\rangle$. The limiting payoff to player k if he plays at x_k is $p_k(x_1, \ldots, x_n; \pi)$, some linear combination of $x_1, \ldots, x_n$. But ϵ-moves might make the actual payoff strictly less or greater than this.

For example, suppose $n = 3$ and $k = 1$. If $x_1 = \frac{1}{3}$ then player 2 will play at $\frac{2}{3} + \epsilon$, after which player 3 will play at $\frac{1}{3} - \epsilon'$; we have $g(\frac{1}{3}) = \left\langle \frac{2}{3}, \frac{1}{3}; 312 \right\rangle$ and $p_1(\frac{1}{3}, \frac{2}{3}, \frac{1}{3}; 312) = \frac{1}{6}$ since $p_1(x_1, x_2, x_3; 312) = \frac{1}{2}(x_2 - x_3)$. But the actual payoff to player 1 will be

$$p_1(\frac{1}{3}, \frac{2}{3} + \epsilon, \frac{1}{3} - \epsilon'; 312) = \frac{1}{6} + \frac{1}{2}\epsilon + \frac{1}{2}\epsilon',$$

always slightly more than $\frac{1}{6}$. Another move for the first player that gives a payoff $\leq \frac{1}{6}$ must be non-optimal. (If the payoff had been $\frac{1}{6} + \frac{1}{2}\epsilon - \frac{1}{2}\epsilon'$, we still would have claimed it was $> \frac{1}{6}$, since $\epsilon' \to 0$ faster than $\epsilon \to 0$.)

Our discussion concluded that this problem could probably be resolved by keeping track of which variables $x_{k+1}, \ldots, x_n$ are equal to their limits, which are slightly higher, and which are slightly lower; for example, $g(\frac{1}{3}) = \left\langle \frac{2}{3}^{+}, \frac{1}{3}^{-}; 312 \right\rangle$. Variables with a "+" or "−" adornment are called "tainted." To decide whether $p_k(x_1, \ldots, x_n; \pi)$ is attained exactly, multiply the adornment of the first tainted variable with a nonzero coefficient in $p_k(x_1, \ldots, x_n; \pi)$ by the sign of that coefficient. If this quantity is positive, then we are slightly higher than the limit, and if it is negative then we are slightly below; $p_k(x_1, \ldots, x_n; \pi)$ is attained exactly if no tainted variables have nonzero coefficients. In the example $p_1(x_1, x_2, x_3; 312) = \frac{1}{2}(x_2 - x_3)$, the variables with nonzero coefficients are x_2 and x_3, and the first tainted one is x_2.

While discussing the apparent intractability of the Toetjes problem, DEK referred to Marshall Hall's axiom, which says that if you can solve a combinatorial problem of size n by hand, then a computer might just be useful for a problem of size $n + 1$. Hall said this in 1960; modern computers might be able to go to $n + 2$ before combinatorial explosion takes over.

DEK then brought up the topic of linear programming. SJP had approached him during the week about handling strict inequalitites. The simplex method is only applicable if the given inequalities are not strict. SM explained that in order to get around this problem they had kludged the inequalities by using non-strict inequalities, shifted by some small amount ϵ.[5] SM had tried $\epsilon \approx \frac{1}{1000}$. This was interesting, because DEK had just asked a colleague of his, whose main area of research is also algorithms, how he would solve a linear programming problem with constraints like $x + y < 1$. His colleague replied that he would probably try $x + y \leq 0.999$.[6]

[5]This ϵ is not to be confused with the variety of epsilon discussed previously.

[6]This colleague remains anonymous, as he does not yet have tenure.

RK wondered whether we were dealing with ordinary real numbers, or what he termed the "hairy" reals, i.e., the reals augmented with various ϵ's. DEK replied that he was just thinking about the ordinary reals, although the interested student may want to read his book "Surreal Numbers."

The simplex method is good at solving systems of the form

$$\mathbf{A}\tilde{x} \geq \tilde{b}$$

or deciding that no solution is possible. (Here $\mathbf{A}$ is a matrix, and $\tilde{x}$ and $\tilde{b}$ are vectors.) Can we also do

$$\mathbf{A}\tilde{x} > \tilde{b}\ ?$$

It seems that the strict form has a solution if and only if the set of solutions to the non-strict form has a positive "enclosed volume". It is not immediately clear whether there is a simple algorithm to test whether the volume is positive.

DEK went on to describe the Fourier-Motzkin elimination method for solving strict inequalities over the reals. Consider the system of inequalities given by

$$\begin{aligned} 8a+4b &< 9 \\ 2a-3b &> 1 \\ 4a+6c &> 0 \\ a+b-c &> 2 \end{aligned}$$

We may rearrange the third and fourth inequalities to get

$$a+b-2 > c > -\frac{2}{3}a$$

after which we can eliminate c leaving a system with one fewer variables:

$$\begin{aligned} 8a+4b &< 9 \\ 2a-3b &> 1 \\ \tfrac{5}{3}a+b &> 2 \end{aligned}$$

We may write this in the form

$$2-\frac{5}{3}a < b < \begin{cases} \frac{1}{3}+\frac{2}{3}a \\ \frac{9}{4}-2a \end{cases}$$

Eliminating b gives

$$2-\frac{5}{3}a < \begin{cases} \frac{1}{3}+\frac{2}{3}a \\ \frac{9}{4}-2a \end{cases}$$

or

$$\frac{5}{7} < a < \frac{3}{4}$$

which has a solution. So there's a solution to the original set.

That worked pretty well, but in general the method can lead to a large system of inequalities. If there are k inequations of the form $f_i < c$, and l of the form $c < g_j$ then eliminating c will yield a system with kl inequations (in addition to those from before that did not involve c). RC and RK noted that they had used this method themselves, but were appalled at the combinatorial explosion that resulted.

DEK then switched the topic to the circular version of the game, which is more symmetric since there are no longer any special endpoints. PL said that he had derived an optimal strategy for the last three players in such a game; their moves depend only on the relative sizes of the largest three intervals after the $(n-3)^{rd}$ player has moved.

According to PL, in the four player game on a circle the second player should play $\frac{1}{3}-\epsilon$ of the way round from the first player, in either direction.[7] The third player should then play at $\frac{2}{3}-\frac{\epsilon}{2}$ exactly. The final player will play in one of the two largest remaining intervals.

There are a number of other geometries which could be considered. We briefly considered extending the problem to higher dimensions, by playing in a square, or on a sphere for instance. (If the first player chooses the north pole and the second player chooses the south pole, it seems that the third player may always control $\frac{1}{4}$ of the globe.[8])

PL suggested that in 2 dimensions the children should be choosing lines rather than points, and that the winner should be the child with the closest perpendicular distance to the mother's point. RK objected, saying that the mother should also choose a line; RC suggested a "least-squares" measure of closeness, while UH suggested using the angle between lines as the measure.

Like many interesting problems, this one was easier to generalize than it was to solve. Having failed to make much progress in two goes at the problem, DEK wondered whether he should use it again for his next CS304 class; perhaps this class could be renamed!

We finished by dividing into groups for Problem 3. The groups are

- ESC, UH, SM
- RC, DQ, MY
- AG, PL, SJP, DS
- DK, RK, AM

There once was a team of programmers
Who became ardent don't-give-a-damners.
They quit their Symbolics'
for haunts more bucolic
then went after their TA with hammers. — MY

There was a TA for Knuth
Whose students wrote lim'ricks, forsooth!
He graded their projects
Devouring their toetjes
And ended up losing a tooth. — KAR

[7]Obviously, by symmetry, it does not matter where the first player plays.
[8]This reminded us of Risk again.

Chapter 4

Label Placement

Thursday February 9

DEK began by announcing that TeX macros for Problem 3 would be made available on `polya`.

DEK stated that the wording of Problem 3 is deliberately vague. He said that when doing research, it is always legitimate to change the problem specification into a form that is more interesting or more interesting to solve. In the case of Problem 3, part of the problem is to specify what is required. Two possible avenues are formal methods, and heuristic methods, although given that this is a practical problem, the proof of the pudding will be in the pictures.

RK had earlier suggested an interactive approach, in which a user could say, for example, "this subset of cities is too cluttered," and the program would then compensate. DEK said that it is not always necessary to have a totally automated system, although for this problem automation would certainly be desirable.

AM wondered if it was legitimate to change the font size. DEK said that this was only justified if the labels just wouldn't all fit at the given font size. He pointed out that five point type is not exactly $\frac{5}{7}$ the size of seven point type; a small correction is added so that human eyes can read it. AM then suggested varying the font from city to city, in order to fit the names in. DS said that residents of cities with small type may get offended if the size of the font was based on something so trivial.

Even for remote cities, there may be some interaction. For example, the configuration

$$\bullet\ \boxed{\text{San Diego}}\qquad\boxed{\text{Springfield}}\ \bullet$$

would not be as good as

$$\boxed{\text{San Diego}}\ \bullet\qquad\qquad\qquad\qquad\qquad\qquad\bullet\ \boxed{\text{Springfield}}$$

although UH did not think that such optimizations were relevant.

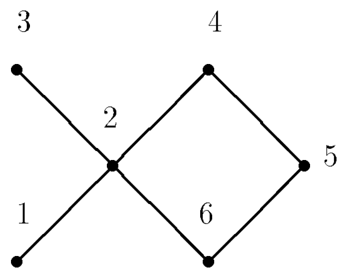
KAR wondered whether the fact that English-speaking readers scan from left to right makes any difference for our placement of labels. DEK thought not, since this does not tell us whether the labels should be to the left or right of their corresponding points. Some others suggested, though, that there may some benefit in attempting to be consistent in this regard. No consensus on this point was reached.

MY then suggested a method based on an artificial potential field. If we imagine the set of points as a physical system, then we may invent "forces" between points and labels. In our situation, we would like points to repel all labels except their own. They should attract their own labels, but only until the labels are sufficiently close; when they are too close, the labels should be repelled since we don't want a label overlapping its point.

Once such a system is set up, it can be iterated, with the positions of the labels changing in proportion to the forces on them. RC and PL discussed the utility of iterating one label at a time rather than once for all labels, although nothing could really be proven without some empirical results.

DEK likened this approach to what has been called "simulated annealing." The idea of this type of method is to formulate some combinatorial problem as a physical process, with some measure of "energy" or "temperature" determining the mobility of the current state. As the temperature is reduced, the state stabilizes into some crystal-like structure; hopefully the order inherent in this structure represents some desired minimum-energy solution to the original problem.

DEK then described a problem he had worked on as an undergraduate. Suppose you have a circuit with arbitrary connections between components. The aim is to arrange the components in a *linear* fashion in such a way that the total length of wire is minimized. For example, consider the circuit



It may be linearized as

1 2 3 4 5 6

which would give a total wire length of 10. Using the "force" idea, we may say that node 2 is being pulled to the right for this linearization, as it is connected to nodes 4 and 6 by relatively long wires. Longer wires are weighted higher, since it is the total length that we want to minimize. We iterate by computing the "center of attraction" of all nodes according to the forces on them, and then sorting the nodes so that their centers of attraction are in ascending order.

This method seemed to do well for the first few iterations, but subsequent iterations tended to alternate between the same configurations. In retrospect, DEK said that introducing some sort of randomness, as exemplified by the temperature in simulated annealing, would have been a good idea to help avoid local minima that are not global minima.[1]

DEK went on to describe the "memistor," which was invented in the 60s by a professor at Stanford. The purpose of this device was to predict whether it would rain at a certain

[1] This layout problem was later shown to be NP-complete, although for some special cases such as the n-cube efficient algorithms are known.

location based on the barometric pressure measured at a grid of points in the vicinity. The memistor worked by combining the barometric pressures according to some linear function, the coefficients of which were stored in the machine. If this linear function was greater than a certain threshold, the memistor would say "rain," otherwise it would say "no rain." The memistor was fed a moderate amount of data, and its predictions were compared to actual outcomes. The crucial feature of the device was that it could learn; when it was wrong, it would tweak its coefficients slightly to make it right next time.[2]

Before allowing any further discussion of Problem 3, DEK wanted to tell one more anecdote, this time concerning the travelling salesman problem. Back in the 60s, AT&T had an application involving punching holes in metal plates. Each plate may have needed up to 200 holes, and a good ordering of the holes to punch would save time and hence money.

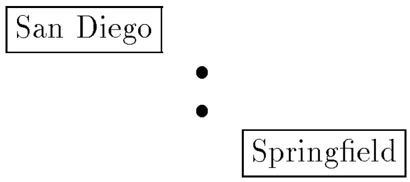
This is the classical travelling salesman problem, which is known now to be NP-complete. Shen Lin, working for AT&T, found the first good heuristic for solving this problem. His technique was basically as follows. Suppose there are n points. Let k be a small number (Lin used $k = 3$), and start with some arbitrary path. Consider all possible improvements in the path length that may be made by permuting up to k pairs of points in the original problem. Choose the permutation that gives the *smallest* improvement, and change the path accordingly.

We repeat this procedure until reaching a local minimum. Doing the computation several times with different (random) starting points, and taking the best solutions, would show cetain edges e common to all the best solutions known. At this point, we include all such e in the final path, and recursively solve the problem for the remaining graph.

This method has running time of order n^{k-1}. Of course, it is possible to construct examples for which this method fails to find the correct tour; however, at the time nobody could do better.[3]

RC then explained the phenomenon known as histeresis, which is encountered sometimes with the simulated annealing process. At low temperatures, points don't want to move, and so it is hard to converge on a good solution, or even to avoid local minima. At high temperatures, there is a lot of random variation, and so convergence in the general sense is not possible.

AM wondered about how one would format the labels of two points directly above one another. RK suggested



DEK remarked that we should have taken vertical separation into account in our earlier example, so that one city would be higher, and the other lower.

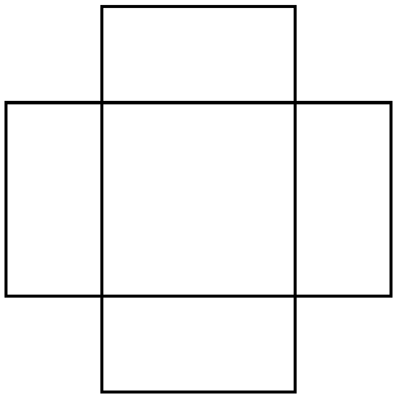
We considered the case with four points in a vertical line. Should the items alternate left/right/left/right, or should they be all on one side? If the points are close together then

[2]At the time, the memister actually outperfomed the weather bureau.

[3]When this method was discovered, AT&T placed numerous advertisements in various journals advertising the fact, and so Shen Lin became somewhat famous.

alternation is required. It isn't clear when, if ever, alternation begins to lose when the points are moved apart.

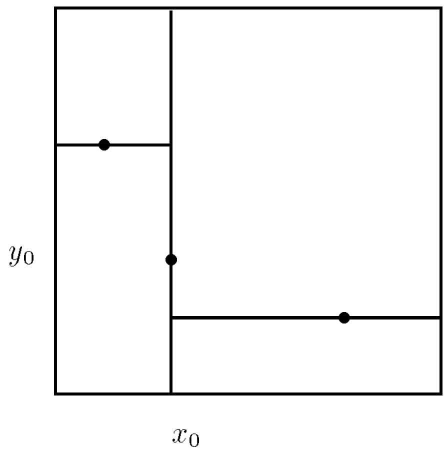
DEK then turned the discussion to data structures. If we represent a rectangle by a 4-tuple consisting of its upper, lower, left, and right coordinates, then how can we tell whether two rectangles overlap? It was suggested that one may look for a corner of one rectangle being enclosed by the other. However, DQ came up with the counterexample



It was conjectured that in our case this cannot happen since all boxes have the same height. RK complained that this was "easy," and that we should spend our class time on the hard parts.

RC suggested the use of k-D trees, which have been used extensively in VLSI design. By total coincidence, DEK just happened to have brought a reference on a related method, which is mentioned below.

A k-D tree is an ordinary binary tree whose nodes are k-tuples, except for the following feature: at the top level, the partitioning is according to the first coordinate; at the second it is according to the second, and so on, cycling around at depth $k+1$. This representation allows efficient search for regions as well as points. RC pointed out that rotation within this structure was difficult, since the two subtrees of a node (x_0, y_0) are not compatible, as illustrated by the diagram



Using a 4-D tree, we can choose one of a subtree's branches if the region we are interested in appears totally to one side of the root of that subtree; otherwise we have to explore both branches.

DEK mentioned a paper by McCreight[4] that appeared in the SIAM Journal of Computing 14 (1985) on pages 257–276. This paper solves the problems alluded to above by placing an ordering on the nodes in the tree of the following form: the root node of a given subtree, which branches on the coordinate x, must have the smallest y-coordinate of all remaining nodes.

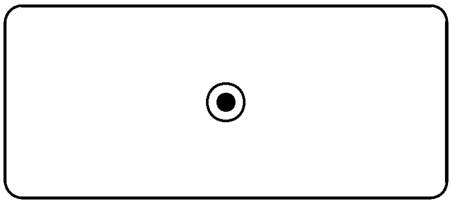
Tuesday February 14

DEK asked what progress had been made on Problem 3.

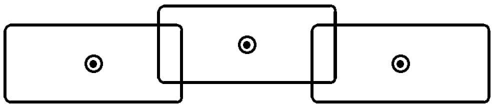
DK said that his group had decided to implement the idea of forces, with partitioning of the data set into components which strongly affect one another.

DEK asked how we can tell when two cities are sufficiently close that they should be in the same component, or "cluster." He said that given a set of pairs of points which interfere with one another, one can use the union-find method to obtain the connected components (which are cliques).

In order to determine whether two points interfere, we can define the "possible label placement area" as the region surrounding a point that may contain a label under some reasonable positioning of the label. DEK gave a simple geometric argument to show that this region is a LaTeX oval[5] and is illustrated below.[6]



SJP argued that using the union-find method may be an overkill; it may not be necessary to construct an equivalence relation. For example, in the situation



we want to say that the middle point affects the two side points, but we may not want to say that the end points affect each other. This information could be stored easily in a simple graph data structure. KAR noted that one could investigate the benefit of, say, looking two edges away rather than just one edge away, and tune the program accordingly.

[4]Pronounced "MacWrite."

[5]This is not quite true, as in a LaTeX oval the curved parts are made as large as possible subject to the dimensions of the oval.

[6]For computational uses, a rectangle is probably sufficient.

AG mentioned that his group was using a force approach, and that it may not be necessary to consider far-away labels, as the force would be small compared to those of points closer to the labels.[7] His group seemed to be using the idea mentioned by KAR above.

AM noted that the graph would need to be constructed only once, at the start of a run, since the points themselves do not move.[8]

DEK then remembered about something he had discussed earlier with KAR regarding test data. We decided that each group should submit a data set corresponding to some (real or imaginary) map, in the same format as DEK's data set in the Data Sheet. All groups will then run (a single version of) their programs on all the data sets. No "tweaking" is allowed on the "mystery" data sets. The data sets will be available to test the programs at about 10am on the day the project is due. Hence, data sets **must** get to KAR by 9am on Tuesday.

DEK described how he transcribed his data set from an atlas in the library. It took him less than two hours, including the time spent converting coordinates. SJP, who had earlier stated that this data set was going to be almost impossible to draw acceptably, said that he now saw some room for improvement, and that things were not so bad. Others were not so optimistic.[9] According to SJP, the hard part was not to separate the boxes from the points and from each other, but to be able to associate points with their correct labels on the map. UH mentioned that using landscape mode for output may give a less cluttered result, because we could scale x and y.

RK then described a "random temperature" approach he had tried, which used a complex evaluation function involving distances from the label under consideration to all other points and labels. He said, though, that this method would probably not do as well as the force method, since you can't tell how to make a reasonable solution better, even if you do have a good estimate of how good it is.[10]

SM pointed out that, strictly speaking, no group was "doing forces" correctly, as they used no concept of velocity. PL countered that they were, but in a system with infinite friction. DEK said that it was not really necessary to model a real system so closely, preserving laws such as $F = ma$. Many forms of potential energy reduction would suffice.

AM thought that it may be reasonable to do horizontal positioning and vertical positioning separately. This is motivated by the asymmetry between them: labels are of constant height but varying width. He also suggested considering only a limited set of positions around a point at which a label may be placed.

KAR suggested that, in the worst case where labels for some points just won't fit, numbers could be assigned to these points, with the label being made explicit in a footnote. (This suggestion was greeted with hisses, for some reason.) PL said that another possibility was generating arrows pointing to such congested points so that labels may be placed in more open areas.

DEK then brought up the topic of random numbers, and conjectured that every good program uses a random number generator. He was mildly embarassed when somebody pointed out that TEX did not have one. "But METAFONT does," he retorted.

[7] "So there's a weak force and a strong force." — DEK

[8] SJP noted that this property holds only if one does not have a buggy program.

[9] "The East Coast is a disaster when looked at in a formal sense." — AM

[10] DEK did not think this was necessarily important; after all "Even though you're stirring scrambled eggs non-uniformly, they seem to get evenly done."

He suggested using a pseudo-random number generator with a user-selectable seed. This is essential when it comes to reproducing results, which is why pseudo-random number generators are sometimes preferable to "real" random number generators.

DEK went on to describe how he had tried varying a set of fonts by changing a small number of bits in the character images, at random. He found that a little randomness actually makes the letters more attractive; a standard deviation of about .005 em in the positions of key points looked best.

DEK then turned the discussion to minimum spanning trees. The method of generating a minimum spanning tree is to iterate as follows: Include the smallest remaining edge that does not join nodes already connected by some path. If there are n nodes in the graph, then the spanning tree will contain $n-1$ edges. We can use this method to generate partitions of the data into clusters. Instead of iterating to $n-1$, iterate to $n-k$ to get k different clusters.

A discussion of annealing, and "simulated annealing," then followed. It was stated by DQ that using simulated annealing techniques, people have been able to solve the traveling salesman problem efficiently within an empirically measured accuracy of 1%.

DEK then asked RK about the cost function he used in the method he described earlier. The denominator of the individual terms of the cost function for city x looked like

$$1 + (d(x, l(x)) - 20)^2;$$

here $d(x, l(x))$ was the distance from x to its label. RK chose this because it could not be zero and it tended to force $d(x, y)$ towards 20. DEK described how he used a similar "badness" factor in TEX to measure whether lines of text were either too sparse or too cluttered. It was not essential what precise form the calculation for the badness took, as long as it had the desired properties of being small for "good" spacing, but high for "bad" spacing. This flexibility allows for a choice of cost function that is easy to evaluate.

DK wondered whether one could improve the convergence behaviour of the force method if one chose a good starting position. He suggested a method of ordering the points from left to right and placing the labels on the opposite side of the "middle." DEK said that the idea of choosing a good starting position was good, and that there may be even better methods for constructing initial positions if we use other orderings besides left-to-right. He likened this process to processes in numerical analysis where the order of operations is carefully selected in order to keep matrices sparse.

Thursday February 16

RK began by describing his attempt at plotting the east coast as "somewhere between not very good and completely invalid." Referring to RK's evaluation function from the previous class, DEK said that it did not distinguish between labels which were one unit too close to, and one unit too far away from their points. He said that it was a worse "sin" to get too close than to be the same amount too far away.

RK then described his idea of starting with only the points initially "out there" and letting the labels out of a "corral" one by one, from the middle point outwards.

This was followed by a discussion of worst-case scenarios on grids of points. AG claimed that we can't have the best of both worlds; we can't have both long range interaction between isolated points and short range interaction within clusters. He said that these two aims are, to an extent, incompatible.

DEK then said that you could still have some relatively small long range interaction even in the presence of the more local interactions. He went on to describe the difference between "arbitrarily small" and "zero," giving several examples.

The first example was physics, where different mathematical models arise depending on whether a particle is thought to have some positive (but unmeasurably small) mass, or thought to be massless.

The second example was related to insurance, where more convenient mathematical models arise from assuming a nonzero probability of a person reaching the age of 500 years.

His final example was in METAFONT where he was experimenting with methods to draw curves through points in such a way that perturbations in one or two of the points would only affect the curve in the immediate vicinity of the change. Trying for a zero change on all non-local regions, he managed to prove mathematically that no such method exists. However, he did find a method that perturbed non-local parts of the curve by about 10^{-20} units, an amount negligible for the particular application.

DS mentioned that being close to its point is not necessarily the most important goal for a label, since this often results in overlapping labels. AG concurred, saying that it takes some effort to read their maps, working from the outside in to determine which labels go with which points.

The discussion then returned to the topic of labelling points consistently. It seems far easier to read labels that are placed in a similar orientation to their points, slightly further away, than labels placed as close as possible to their points. This becomes particularly noticeable when the points are congested.

SM expressed frustration with the annealing process,[11] saying "it has a mind of its own." He said he couldn't get it to stop where he wanted. Somebody added that simulated annealing does not tell us where the constraints are breaking. PL added that it is possible to satisfy many of the constraints, and yet get an unpleasant result.

In passing, DEK asked how much time the programs were taking. DS said that his group's program did 200 iterations in one minute of (elapsed) time. DK said that his group's program took 20 minutes to do 250 iterations.[12]

AG expanded on his group's formula for calculating the force on a label. The force F is proportional to

$$\frac{160000}{d^2} - d^2,$$

where d is the distance from a point to its label. This expression gives a zero force at $d = 20$, and pushes the labels towards $d = 20$ from either side of 20. AG remarked that this expression for the force performed better than a quadratic expression.

RC then returned to the phenomenon of bouncing. He said that this problem could be alleviated by reducing the time-step so that objects pushed by large forces don't get moved

[11] RC noted that the force method being implemented was not, strictly speaking, simulated annealing, since there are built-in heuristics in our problem. Such observed phenomena as "bouncing," where labels seem to bounce off one another as the iteration progresses, are not typical of simulated annealing.

[12] There followed a discussion of elapsed time versus computer time. DEK reminisced, "If it weren't for timesharing I would be a healthy man today."

too far. The principle he had in mind was similar to the one used to draw the Mandelbrot set on personal computers. The idea is simple to state: Spend extra time on calculations that are critical, and do the calculation in "chunks" where they are not critical.

DEK was unclear which of Mandlebrot's many contributions to the theory of fractals RC was referring to. RC explained as follows. Consider the iteration scheme

$$z_{n+1} = z_n^2 + 2$$

over the complex numbers. Let M denote the Mandlebrot set. For a given initial value z_0, $z_0 \in M$ if and only if the iteration given above does not diverge.

This reminded DEK of the fractal globe problem that he set for CS304 in 1983. The interested reader is referred to the report of that class. Our class then diverged into a discussion of fractals, Peano sets, Pascal's triangle and the Cantor ternary set.

Eventually, the discussion returned to Problem 3. SM described how he might do this problem by hand. Put the labels near their points, one by one, in any vacant space. If no vacant space exists, then adjust some neighboring point (and recursively adjust its neighbors) until there is space. Of course, there are common-sense heuristics about label placement that would be hard to code. DQ argued that there were more heuristics in this problem than annealing was designed to cope with.

DK came up with an interesting idea. He suggested forming the convex hull of all the points. Labels could be placed for all points on the convex hull, and then the remaining points could be solved recursively. DS said that this method may need partitioning in order to be useful, in order to avoid grouping different clusters within the same hull. KAR pointed out that after building up the sequence of convex hulls it may be better to attach the labels from inside to outside, since the points closer to the outside are likely to have the most room around them.

Tuesday February 21

The first ten minutes of class were spent voting on the outputs from Problem 3. KAR arranged the outputs of the three groups that had submitted in time in random fashion, so that the votes would be "blind."[13] See Appendix B for some of the "winning maps."

DEK asked whether there were any concluding remarks about Problem 3. RK and UH said that each of their groups turned off the randomness altogether, since it did not seem to improve the performance at all.[14]

ESC compared his approach to that used by the Waltz algorithm.[15] The Waltz algorithm is a method for recognising three dimensional figures from two-dimensional edge drawings. ESC's group's method is a form of constraint propagation. They classify the neighborhood of a point into a small number of regions; testing membership in these regions may be done efficiently using binary operations. Those labels that cannot be placed in this way are handled in a second pass that uses forces.

[13] As pointed out by SJP, these votes aren't really blind, since most group members were able to recognise their own outputs.

[14] RK qualified his statement, saying that the randomness did in fact help an earlier version of his program that still had bugs.

[15] This algorithm was on the comprehensive exam reading list; refer to Rich's book "Artificial Intelligence" for further details.

DS said that, in contrast to ESC's description, his group used a first pass with the force method, followed by a second pass in which labels are brought back near their points. These two passes were iterated.

DEK wondered how the groups debugged their programs. DS had written a previewer, which allowed his group to observe the movement of cities in real time on a Sun workstation. MY, RC and DQ had done a similar thing on a Symbolics machine.

DEK was impressed by this, and said that he would like to see these previewers working some time. He went on to describe how, in the early days of timesharing, a video display was made showing how the scheduler performed: which jobs were making progress and which jobs were thrashing. The display immediately provoked a number of good heuristics that could be implemented, and when implemented they did improve performance considerably.

> This problem three programming team
> knows Common Lisp, Pascal and Scheme
> Our problem arose
> because none of us knows
> how to program in all of the threeme. — MY

> I once met a CSD chap
> Whose task was to draw well a map
> Annealing and forcing
> Heuristic contorting
> Did not leave a big enough gap. — KAR

Chapter 5

AND-OR Tree Formulas

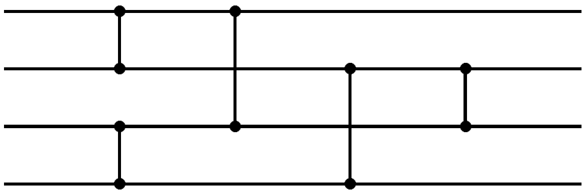
At this point, it was time to decide the groups for Problem 4. In order not to make sure the constraint of no pair working together more than twice was not violated, groups for Problem 5 were worked out at the same time. The groups are

Problem 4	Problem 5
UH PL AM	RC RK PL SJP
ESC DK SJP	ESC SM MY
RK SM DQ	UH DK DS
RC AG DS MY	AG AM DQ

DEK quickly moved on to Problem 4, expanding on the material from the Data Sheet. He described how complexity theorists have tried to answer questions on circuit building from basic logical gates.

For n inputs, determining whether all inputs are 1, or whether some input is 1 is easy: Simply use $n-1$ AND or OR gates respectively. The problem of determining whether half the inputs are 1 is a harder problem; the function that is 1 precisely when at least half the inputs are 1 is called the majority function for n variables.

DEK described a sorting network, which consists of a series of parallel lines joined by pairwise comparators in such a fashion that no matter what the inputs to the lines, the output is in sorted order. For example, the four-element sorting network looks like



To calculate the majority function, construct the sorting network of the appropriate size[1] and check whether the center wire[2] contains a 1. The difference between this and the type of circuit we will be looking at is that we are interested in trees, while the circuit above is

[1]This can be done with $O(n \log n)$ comparators.

[2]Or the wire just below the center if n is even.

a directed acyclic graph (dag). In a dag, outputs from a circuit may be used many times (arbitrary fanout) as inputs to other parts of the circuit.

To simplify the problem, we assume, by symmetry, that

$$p_1 = p_2 = \ldots = p_n = p$$

so that $p_0 + np + p_\infty = 1$. We also adopt the convention of numbering nodes level by level, from top to bottom, and left to right within a level. The root will be numbered 1, and the rightmost leaf will be numbered $2^k - 1$ where the tree has depth k.

So consider an AND/OR tree of depth 4, whose root node is an AND gate. There are 15 nodes altogether, with the leaf nodes each being 0, 1 or one of the n inputs. Define $P_j = \Pr(f(x_1, \ldots, x_n) = 1)$ where f is the function being computed at node j. It is not difficult to see that we can express P_j recursively. For example, $P_1 = P_2 \cdot P_3$ and $(1 - P_2) = (1 - P_4) \cdot (1 - P_5)$.

The probabilities at the leaves are given by

$$P_8 = \ldots = P_{15} = p_1 x_1 + \ldots + p_n x_n + p_\infty = p(x_1 + \ldots + x_n) + p_\infty.$$

If m of the n inputs are 1, then we write this expression as $p^{(m)}$.

So suppose $x = p^{(m)}$. Then $P_8 = x$, $P_4 = x^2$, $P_2 = 1 - (1 - x^2)^2 = 2x^2 - x^4$, and $P_1 = (2x^2 - x^4)^2$, which we abbreviate by $t(x)$.

Given this, we can list the possible configurations of bits with their associated probabilities of being $\neq \theta_{n/2}(x_1, \ldots, x_n)$ as follows. (We suppose for this table that $n = 6$.)

000000	$t(p^{(0)})$
000001	$t(p^{(1)})$
000010	$t(p^{(1)})$
000011	$t(p^{(2)})$
$\vdots$	
000111	$1 - t(p^{(3)})$
$\vdots$	

If the sum of the probabilities in the second column is less than 1, then there is a nonzero probability that the computed function is in fact the majority function. Note that the sum of probabilities is actually a conservative estimate of the function not being the majority function, as the rows are not independent.

We may make the recursion more uniform by considering AND and OR gates as duals, writing $P_{AND} = \Pr(f = 1)$ and $P_{OR} = \Pr(f = 0)$. In this case, for both AND and OR gates, $P = (1 - P_{CHILD})^2$. Thus the P function at the root of a k-level tree is $g_k(x)$ for k even, $g_k(1 - x)$ for k odd, where $g_0(x) = x$ and $g_k(x) = (1 - g_{k-1}(x))^2$ for $k > 0$.

The graphs of $g_k(x)$ for various depths reveal an interesting pattern. See the graphs in Appendix C. At a critical point α, the function changes from near 0 to near 1 very rapidly. The slope of this change increases with k.

One may wonder where the fixpoint α of such a recursion lies. The fixpoint is simply the solution of

$$(1 - x)^2 = x$$

between 0 and 1. It turns out to be $1/\phi^2$ where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. One can show that

$$g_k'(\alpha) = -2(1-\alpha)g_{k-1}'(\alpha)$$

For a graphical representation of $g_k(x)$ see Appendix C.

Thursday February 23

DEK began by reviewing some of the material from the last class. Something that somehow escaped the notes last time was that for trees of even depth (where depth is measured to the last AND or OR gate; not to the nodes which take on the input values) the AND/OR tree is equivalent to a tree in which all nodes are of one type. In particular, an AND/OR tree with an AND root can be replaced by a structurally identical tree in which all nodes are NOR gates. Similarly, an AND/OR tree with an OR root can be replaced by a tree consisting only of NAND gates. (You have to complement the leaves if they're at an odd distance from the root.)

If we take $p = 1/n$ (we use the terminology from last class) then we get

$$f(x_1, \ldots, x_n) \approx \theta_{n\alpha}(x_1, \ldots, x_n).$$

Suppose $n = 100$, so that $38 < n\alpha < 39$. If k is even, then the probability of f not being the $\theta_{n\alpha}$ function is

$$g^k(\frac{38}{100})$$

when exactly 38 of the inputs are equal to 1, and

$$1 - g^k(\frac{39}{100})$$

when there are 39 inputs of 1. If k were odd, then the arguments of g^k would be $\frac{62}{100}$ and $\frac{61}{100}$ respectively.

DEK was waiting for a certain observation to be made.[3] The observation was that every AND/OR tree must necessarily be a monotone circuit. Changing a 0 to a 1 at the bottom level can only result in 0's becoming 1's at higher levels; the transition from 1 to 0 cannot occur as a result. This means that we can get away with looking at the cases 38 and 39. Therefore, a conservative estimate of the probability that $f(x_1, \ldots, x_n)$ is not the function $\theta_{n\alpha}$ is obtained by summing the failure probabilities over all cases with 38 and 39 input 1's, namely

$$\binom{100}{38} g^k(\frac{38}{100}) + \binom{100}{39}(1 - g^k(\frac{39}{100}))$$

when k is even.

DEK gave a brief summary of previous work on the problem. Leslie Valiant proved that the existence of a circuit could be guaranteed in $O(n^{v+\epsilon})$ gates, where $\epsilon > 0$ and

$$v = 2 + \frac{\ln 2}{\ln(\sqrt{5} - 1)} \approx 5.27056.$$

[3]It had in fact been made in passing last class, but nobody was pursuing it at the moment.

The ϵ comes from the observation that the exponent of n in Valiant's paper is $(2 + 2\log_\gamma 2)$ where $\gamma < 4\alpha$, and α is as before.

Boppana stated that Valiant had "implicitly proved" that the number of gates is $O(n^v)$. But DEK didn't believe this.[4] Boppana went on to prove that the number or gates needed is actually $\Omega(n^v)$ using any Valiant-type argument. (DEK was ecstatic over this part of Boppana's paper, but he said it wasn't required reading for Problem 4.) Our challenge is to find the smallest constant C for which we can say that the number of gates $\leq Cn^v$.

One approach is to write a program to compute

$$\theta_{a,b}(x_1, \ldots, x_n) = \begin{cases} 1, & \text{if } \sum x_i \geq b; \\ 0, & \text{if } \sum x_i < a. \end{cases}$$

This may give some intuition of what the problem is like so that an appropriate theorem may be postulated.

The number that we want to keep below unity in this case, for k even, is

$$\binom{n}{a-1} g^k((a-1)p + p_\infty) + \binom{n}{b}(1 - g^k(bp + p_\infty)).$$

Given n and k, this function is easy to compute; the only thing to watch for is loss of significant digits due to rounding. And we have to choose the probabilities so that $np + p_\infty \leq 1$.

PL then described a general method for building the majority function for n inputs out of the majority functions for fewer inputs. He described his method for $n = 8$.

Let C be the set containing the eight input variables $x_1, \ldots, x_8$. Divide C into two disjoint subsets A and B of four inputs. Then $\theta_4(C)$ is

$$(\theta_0(A) \wedge \theta_4(B)) \vee (\theta_1(A) \wedge \theta_3(B)) \vee (\theta_2(A) \wedge \theta_2(B)) \vee (\theta_3(A) \wedge \theta_1(B)) \vee (\theta_4(A) \wedge \theta_0(B))$$

The function θ_0 is trivial; θ_1 and θ_4 may be calculated using 3 gates each, and θ_2 and θ_3 need 7 gates each. Adding up all the gates needed according to the above expression gives a total of 47 gates, which agrees with DEK's prior calculation (of another circuit).

Using this method, PL was able to construct circuits for $\theta_8(x_1, \ldots, x_{16})$ using only 479 gates,[5] and for $\theta_{16}(x_1, \ldots, x_{32})$ using 8639 gates. In general, the recurrence for circuits of 2^n inputs constructed in this manner is

$$t_n(k) = 2(t_{n-1}(0) + \ldots + t_{n-1}(k)) + 2k - 1$$

for $k \leq 2^{n-1}$. Rewriting this with $t'_n(k) = t_n(k) + 2$ simplifies things slightly, but the recurrence still looks difficult.

DEK remarked that this was a nice, systematic method for small n, but that the recurrence might be exponential, and so could not be as good as possible for large n. PL said he thought he could show a growth rate of at most $n^{\log n}$. If that's the true rate of growth, the circuit cannot be optimum for large n, and it will be interesting to find a crossover point where Valiant's method begins to win.

[4] Valiant's published proof does not rule out the possibility of the complexity being $n^v \log n$, for example.

[5] This convincingly beat DEK's circuit, which needed 767 gates. And it's a good thing too, because DEK said he lost his notes about how to achieve 767.

Tuesday February 28

DEK started by remarking that it might not be too difficult, given a, b, n and k, to determine whether there exist p and p_∞ such that

$$\binom{n}{a-1} g^k((a-1)p + p_\infty) + \binom{n}{b}(1 - g^k(bp + p_\infty)) < 1$$

and $np + p_\infty \leq 1$. Exactly how this would help us in our situation was unclear. DQ suggested taking $b = \lceil \frac{n}{2} \rceil$ and trying to get a as close to b for any given n and k. SJP later described his method of setting $a = b = \lceil \frac{n}{2} \rceil$ and finding the best k for a given n.

DEK then said he wanted to talk a little bit about Schröder functions.[6] Schröder functions are analogous to eigenvectors encountered in linear algebra. For a function f whose power series expansion for small $|x|$ is of the form

$$f(x) = \beta x + f_2 x^2 + f_3 x^3 + \cdots$$

where $|\beta| \neq 1$, we can construct a Schröder function $S(x)$ such that

$$S(f(x)) = \beta S(x) \tag{5.1}$$

for $|x|$ sufficiently small. Furthermore, $S^{-1}(x)$, the functional inverse of $S(x)$, is well defined in this range. The expansion of $S(x)$ looks like

$$S(x) = x + s_2 x^2 + s_3 x^3 + \cdots$$

from which it is not too difficult to show that $S^{-1}(x)$ has the form[7]

$$S^{-1}(x) = x + t_2 x^2 + t_3 x^3 + \cdots.$$

Equation 5.1 implies that $f(x) = S^{-1}(\beta S(x))$, and more generally,

$$f^k(x) = S^{-1}(\beta^k S(x)) \tag{5.2}$$

for all integers k. In fact, Equation 5.2 makes sense even when k is not an integer! Note that we only need to know that a Schröder function exists — we don't need to know one precisely.

In our case, we may write $g(x) = (1-x)^2 = \alpha + f(x - \alpha)$, with $g^k(x) = \alpha + f^k(x - \alpha)$.

$$\begin{aligned} f(x) &= g(x+\alpha) - \alpha \\ &= (1/\phi - x)^2 - 1/\phi^2 \\ &= x^2 - \frac{2x}{\phi} = x(x+\beta) \end{aligned}$$

[6] Schröder was a not-so-famous mathematician who wrote two large books about Boolean algebra, a topic that was not in the mainstream of mathematics at the time. He did not, as far as DEK knows, link Schröder functions with Boolean algebra, so it is ironic that we are using them now on Boolean circuits.

[7] For those wondering about how to manipulate the series expansions, see Knuth's "Seminumerical Algorithms," Section 4.7.

where $\beta = -2/\phi$ is about -1.2. For this $f(x)$, DEK has calculated the first eight terms in the power series for $S(x)$:

$$
\begin{aligned}
s_2 &= \frac{1}{\beta - \beta^2} \\
s_3 &= \frac{2\beta}{(\beta-\beta^2)(\beta-\beta^3)} \\
&\vdots \\
s_8 &= \frac{429\beta^{21} + 165\beta^{19} + \cdots}{(\beta-\beta^2)\ldots(\beta-\beta^8)}
\end{aligned}
$$

He noticed that the exponent of β in the leading term of the numerator for s_k was the $(k-1)^{\text{st}}$ triangular number, and that the coefficient of this term was the $(k-1)^{\text{st}}$ Catalan number.[8]

SJP then returned to the program he wrote to find the best k for a given n where $a = b = \lceil \frac{n}{2} \rceil$. One thing he noticed was that k never seemed to be odd. In order to satisfy the constraints mentioned above, he chose p in the same way as Valiant, i.e., such that

$$(\frac{n}{2} - 1)p < \alpha < \frac{n}{2}p.$$

DEK noted that odd k requires a different formula, so it was no wonder SJP got only even answers. We should set p_∞ to 0 when k is even, and p_0 to 0 when k is odd.

RK observed that Valiant actually tries to achieve an overwhelming probability in his paper, not just a positive one. In other words, he chooses k so that a large portion of the circuits in the class being considered actually compute the majority function.

There followed a discussion of probabilities, with DEK describing probabilistic primality testers that can say that a number p is nonprime with probability of the order of 2^{-100}. Nevertheless, this is not a "proof," although p being nonprime is considerably less likely than anything we could imagine. DEK added that somebody recently compared the risks of smoking and flying: Apparently, flying will become more dangerous than smoking when there are three major plane crashes per day.

RK returned to SJP's calculations, asking whether he had considered accumulated round-off errors. SJP replied that he considered that he should have considered them, but didn't. It turns out that the iteration

$$t_k = (1 - t_{k-1})^2$$

is very bad numerically when t_{k-1} is close to 0 or 1; many significant digits are lost every two steps. However, by composing two iteration steps one gets

$$t_k = t_{k-2}^2(2 - t_{k-2})^2$$

which performs better. The relative error doubles every two iterations, so we still have to be concerned about it, but we are still much better off than before.

[8] DEK joked that since such beautiful relationships held for the first 8 values, God wouldn't want it to be any different for the rest. DS referred to this argument as "proof by appeal to a higher authority."

SJP realized that his results suffered from the rounding error problems; it was not long before his iterates became either 0 or 1. He hoped, however, that his early iterates were still reasonable, since he did use double precision. Here are the initial results:

n	k	$2^k/n^v$
2	1	.05
4	8	.17
6	12	.32
8	14	.28
10	16	.35
12	18	.54
14	20	.95
16	22	1.89
18	22	1.02
20	22	0.58

The row where $n = 16$ indicates that $2^{22}-1$ gates are sufficient, despite PL's construction of a circuit with only 479 gates. One difference is that PL's circuit was not a full tree; it has possibly different numbers of nodes on each branch of a gate. It effectively simplifies gates with inputs of 0 or 1, and makes other simplifications (such as a gate with identical children). AG noted that we could do better by observing that p_0 knocks out a number of the gates. From Valiant's results, $p_0 = 1 - np \approx 1 - 2\alpha \approx 25\%$.

If we could find some bound for the probability that there are more than $c2^k$ gates left after simplification, then we could add this term into our conservative estimate for the failure probability and we might get a sharper inequality.

RK noted that our assumption of the inputs being independent is probably oversimplifying it because one intuitively expects an approximately equal number of each input in a circuit that computes the majority function.

DEK said that another opportunity for improvement might be to find an alternative construction, perhaps using threshold functions, which gives a steeper graph than the one we have now.

SJP wondered how Boppana's trees, which use read-once inputs, related to our trees in which inputs may be repeated. DEK replied that using read-once inputs was just a way of abstracting away independence of the leaves. Effectively, we assign a probability for each read-once input being one of our given inputs. Independence was necessary for Boppana, for instance to substitute whole circuits at the leaves of other circuits without having to worry about similarly substituting for the same input appearing as another leaf.

Thursday March 2

DEK continued from last class by considering the double iteration

$$g^2(x) = x^2(2-x)^2.$$

The right hand side is a quartic, and so the iteration should have four fixpoints. It is easy to see that 0, 1 and α must all be fixpoints. Let r be the fourth fixpoint. Then

$$x(x-1)(x-\alpha)(x-r) \equiv x^4 - 4x^3 + 4x^2 - x$$

and so $-\alpha r = -1$, i.e., $r = 1/\alpha \approx 2.6$.

Previously, we had $g(x) = \alpha + f(x-\alpha)$. For $g^2(x)$ there are four possible arrangements corresponding to the four roots. For example, expanding about the root 1, we get

$$\begin{aligned} g^2(x) &= 1 - f(1-x) \\ \Rightarrow f(x) &= -g^2(x+1) + 1 \\ &= -(x+1)^2(1-x)^2 + 1 \\ &= 2x^2 - x^4 \end{aligned}$$

for sufficiently small x. So, we have

$$f(x) = \begin{cases} 4x^2 - 4x^3 + x^4, & \text{near } 0; \\ 2x^2 - x^4, & \text{near } 1. \end{cases}$$

Note the asymmetry in the two cases: The leading coefficient near 0 is twice as big as the coefficient near 1.

DEK then began talking about "generalized Schröder functions." Suppose we can write

$$f(x) = \beta x^2 + f_3 x^3 + \cdots$$

where $|\beta| \neq 1$ and the leading term's exponent of x is 2. Then a generalized Schröder function will satisfy

$$S(f(x)) = \beta S(x)^2.$$

(We can do similar things if the leading exponent of x is bigger than 2 too.)

For f defined as above, it is not difficult to show that the generalized Schröder function leads to the iteration formula

$$S(f^k(x)) = \beta^{2^k - 1} S(x)^{2^k}$$

for all integers k.

So, near 0, ϵ becomes $\frac{1}{4}(4\epsilon)^{2^k}(1 + O(\epsilon))$. Similarly, ϵ becomes $\frac{1}{2}(2\epsilon)^{2^k}(1 + O(\epsilon))$ near 1. Given this apparent asymmetry, DEK thought that it might gain us something to skew the initial estimates somewhat.

DQ did not agree, saying that he had chosen 32 evenly spaced values for p in the interval $(\frac{2\alpha}{n-2} \,..\, \frac{2\alpha}{n})$, and had found the sharpest decrease at the middle of the interval, at least when n is 20 or so.

DEK then started describing the concept of zones. Suppose we start our iteration at $\alpha - \epsilon$. Our theory around α (call this "zone 1") tells us that we will iterate from $\alpha - \epsilon$ to $\alpha - \beta^2\epsilon$ to $\alpha - \beta^4\epsilon$ up to $\alpha - \beta^{2k_1}\epsilon$ (plus lower order terms). The iteration stops when we are out of zone 1. Zone 1 has a fixed boundary, say at 0.38, so it is possible to test when we are no longer in zone 1.

Let's skip zone 2 for the moment, and consider what happens in zone 3, which is a fixed neighborhood of 0. Once we enter zone 3, say at point δ, then our theory around 0 tells us that we iterate from δ to $(4\delta)^2/4$ to $(4\delta)^4/4$ and so on up to $(4\delta)^{2^{k_3}}/4$. We will talk more about k_3 shortly.

Zone 2 is simply the region between zones 1 and 3. There is some fixed bound (say, k_2 steps) so that we always get through zone 2 within k_2 steps. Our total number of steps, k is then equal to $k_1 + k_2 + k_3$. How large does k_3 have to be? Large enough so that the $(4\delta)^{2^{k_3}}$ is sufficiently small to wipe out the $\binom{n}{n/2-1}$ term in our conservative probability estimate.

Well, $\binom{n}{n/2-1} \approx 2^n/\sqrt{\pi n}$, so we want

$$\begin{aligned} (4\delta)^{2^{k_3}} &\approx 2^{-n} \\ \text{so, } 2^{k_3} &\approx cn \end{aligned}$$

for some constant c.

Now, how about k_1? Well, first remember that $p \approx \frac{2\alpha}{n-1}$ and so $\frac{n}{2}p \approx \frac{n\alpha}{n-1} = \alpha - \frac{\alpha}{n-1}$. Hence $\epsilon \approx \frac{\alpha}{n}$. Suppose the boundary of zone 1 is at 0.380 (recall that $\alpha \approx 0.382$). Then

$$\begin{aligned} \alpha - \beta^{2k_1}\epsilon &\approx 0.38 \\ \text{so, } \beta^{2k_1}\epsilon &\approx 0.002 \\ \text{i.e., } k_1 &\approx c' \log(1/\epsilon) \approx \frac{\log n}{\log \beta^2} \end{aligned}$$

Since k_2 is bounded, we have an expression for k that is $O(\log n)$. Note that k_2 is an upper bound on the number of iterations needed to get through zone 2, so that for some initial zone 2 points $k_2 - 1$ iterations may suffice.

AG said that we still need some rigorous inequality for the Schröder function. With Taylor series, say

$$f(x) = f(a) + (x-a)f'(a) + \cdots + \frac{(x-a)^n}{n!} f^{(n)}(a) + R_n(x)$$

we can calculate the remainder R_n using

$$R_n(x) = \frac{1}{(n)!} \int_a^x t^n f^{(n+1)}(x-t),\, dt.$$

RK pointed out that in the case $\beta = 2$, the Schröder function for $2x + x^2$ is

$$S(x) = \ln(1+x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 + \cdots.$$

Then $S(f(x)) = \ln(1 + 2x + x^2) = 2\ln(1+x)$.

SM wondered what happened on the other side of α, when the iterates approach 1. DEK said that it would have analogous zones near α, near 1, and in between, and that a similar analysis holds there too; some of the minuses become pluses, and the factor near 1 is 2 rather than 4, but the principle is still the same.

DEK asked whether anyone had developed the idea of threshold function modules to use for constructing larger circuits. Nobody had. RK said that he and SJP thought that it may be too restrictive to only use threshold functions as building blocks. Perhaps using asymmetric circuit modules (threshold functions must be symmetric on all inputs, by definition) would yield smaller circuits overall. As RK put it, "There are lots of monotone circuits that are not threshold functions."

Tuesday March 7

PL and UH had plotted a log-log graph of the numbers of gates obtained from the recurrence mentioned last week. It was not a straight line, so the solution was not a polynomial. However, they estimated that Valiant's method becomes superior only when $n > 1600$. To obtain this crossover point, they used empirical observations of the convergence of the iteration (up to $n = 180$) to estimate the asymptotic complexity of Valiant's approach as $0.4n^v$.

There must have been some inter-group collaboration, because the reason that UH stopped at $n = 180$ was "That was as far as SJP went." SJP said that this was the point at which the floating point arithmetic broke, probably due to some very small quantity vanishing below the machine's smallest representable number.

DEK said that in this type of circumstance it makes sense to extend the possible range of numbers by representing numbers explicitly as pairs. The first component is an ordinary floating point number, and the second component is an integer exponent for an additional power of some base. Since machine exponents tend to be limited to of the order of 8 bits, this can gain a lot of expressive power. Of course special routines need to be written to handle addition, multiplication, output, and so on, but these are easy.

DEK went on to describe other problems that plagued floating point arithmetic until only recently. Before an IEEE standard was set in about 1980, computer manufacturers gave no guarantees that their floating point arithmetic was reliable. Fundamental mathematical laws such as

$$\forall t, u, x: \quad t < u \Rightarrow (t + x \leq u + x) \text{ and } (tx \leq ux \text{ when } x > 0).$$

were violated.

SM said that an alternative to maintaining pairs for real numbers was to work in logs.[9]

SJP mentioned that after he fixed his program following our observations on numerical error, the results did not noticeably change. There followed a brief analysis of the single-step iteration, from which we concluded that it probably only made a difference over the final three or four steps.

RK then announced that his group[10] had proved a rigorous (although probably conservative) bound of $256n^v$.

DEK seemed pleased at this, and even hinted at the possibility of publishing their result, given the gap in the literature. He went on to describe how it was becoming unfashionable to maintain high standards of rigor in computer science. (An example from his own experience was deriving a recurrence for the complexity of a program from the program itself; everybody might agree that it's the correct recurrence, but how can you get a machine to check such an assertion?)

"How did you get the rigorous bound?" DEK asked.

"Taylor series," SM answered, as if that was enough to convince everybody. When pressed for further details, he explained as follows.

[9] Why do snakes mate inside tree trunks? Because adders can only multiply in logs.

[10] The "group" actually was formed by a merger of two groups. Since the two groups had worked so closely together, they got KAR out of bed at midnight the previous evening to ask if they could submit a joint report. I said OK, but that I would demand a higher standard than otherwise.

Let $g(x) = x^2(2-x)^2 = \alpha + f(x-\alpha)$. Then the Taylor series expansion for $f(x)$ is

$$\beta x - cx^2 + \cdots$$

where c is positive. Letting C be slightly greater than c, we have

$$\beta x \geq f(x) \geq \beta x - Cx^2$$

within a sufficiently small neighborhood of 0. Using this inequality, we can iterate to get the more general inequality

$$\beta^k x \geq g^k(x) \geq \beta^k x + A_k x^2$$

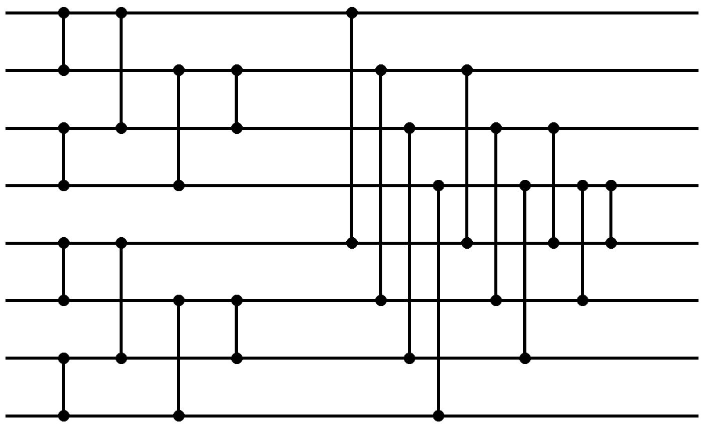
where $A_k = C(\sum_{i=k-1}^{2k-2} \beta^i)$.

AG said he had come up with essentially the same argument to show that such a constant exists, but that due to time constraints he had not actually calculated the constant. (Of course, doing the calculation of the constant is the easy part.)

There followed an energetic discussion, in which the results of the various groups were compared, and discrepancies (such as apparently different observations of the rate of growth of the number of gates) were (hopefully) explained.

PL interjected with the comment, "All this talk is futile because my method is better for up to 10^{16} gates!"

PL's method reminded DEK again of sorting networks. Given two sorting networks of equal size, there is a standard merging operation (due to Batcher) to give an network of double the size. For example, two 4-element networks may be merged as follows:



Constructing networks recursively using this method uses approximately $n(\lg n)^2/2$ comparators.

DEK together with Bob Floyd had tried hard to prove that $\Omega(n(\log n)^2)$ was a lower bound, but couldn't. It was not surprising that they couldn't, given that Ajtaj, Komlós and Szemerédi (AKS) later proved the existence of sorting networks requiring only $O(n \lg n)$ comparators. Nevertheless, this asymptotically better method had a constant of about 100, which meant that it only became better than Batcher's $\frac{1}{2}n(\lg n)^2$ method when $n > 2^{200}$.

AG observed that we could obtain a tree from a sorting network by duplicating inputs to comparators that are used more than once. Thus a method of depth d gives a formula of size $\leq 2^d n$. The AKS network therefore implies a polynomial circuit size for threshold functions, but the exponent is much higher than Valiant's.

Chapter 6

Antomology

With only a few minutes left, DEK switched the discussion to Problem 5. PL wondered what degree of competition was going to be maintained, and what amount of secrecy was appropriate. DEK said that we could discuss general methods of solving ant problems without giving away too much of the algorithms. He said the discussions would raise the level of competition, since he assumes it will be hard for anybody to keep their good ideas to themselves.

"Besides," somebody said, "it may not influence the final runs much anyway, since most of the work seems to get done between the Thursday and the final Tuesday."

Thursday March 9

DEK started the class by discussing Randy Anty, the simple ant program from the Data Sheet. He drew diagrams on the blackboard corresponding to the state of the simulation (on the small board in the Data Sheet) at times 100, 200, and 400; the ants got all the food at time 451.

It was moderately clear that the ants weren't very clever. They would wander around randomly, laying scent, until they found food. At this point they would wander around randomly, erasing scent, until they found the nest. Since ants only came out of the nest onto an unscented square, only a small number of ants (at most 4) initially left the nest.

RK observed that an ant coming home can just wander by the nest, clearing the scent, so that another ant comes out. This can happen many times, leading to more and more ants on the field. Furthermore, an ant will never go into the nest until it has found food. SJP observed that on a slightly larger field there was a minor population explosion, with an ant carrying food blocked in by its neighbours. Eventually, though, it got home. DEK said that it was not hard to imagine a situation in which one or more ants would get hemmed in permanently, never being able to return home.

In order to fix this apparent bug, DEK wrote Randy Anty II. This version attempts to control the population as follows: If an ant sees another ant ahead of it, turns right and sees another ant, then it decides to go home even if it does not have any food. DS said that this may actually be a slight overkill, since such an ant may be observing another "suicidal" (or should we say altruistic?) ant. There was a more obvious flaw than this, though. When an ant returned to the nest for this reason, it erased the scent as it went, thereby releasing another ant (or perhaps several if it wandered around the nest). The result was an even greater population explosion.

Randy Anty III then followed. This fixes the bug in Randy II by not allowing empty-handed ants that return to the nest to erase scent.

There followed a discussion of how the ant programs would be tested in the Great Race. Certainly multiple nests, separated food, and arbitrarily hard fields were fair game for the tie-breaking tests. Other tests, such as random perturbations of scents, would probably be interesting but could not be done by the present version of the simulator. SJP suggested a "spider" that ate unwitting ants stumbling on its square. RC thought that this wouldn't be fair, since the ants can't do much about it.

DEK wondered how many neurons there were in an ant. RC happened to know that there are about 50000 in a grasshopper. DEK mentioned that Tom Binford had recently told him that a neuron was more correctly thought of as a microprocessor than as a gate.

PL had asked the following question before class: "Why not generate a new random bit after each unsuccessful test of a candidate instruction, rather than using the same bit throughout?" AM said that using different random bits would allow many different probability levels, not just probability $\frac{1}{2}$.

DEK's first response was to say that getting a new random bit each time would significantly slow down the simulator. However, there is a more technical reason to use the same bit.

The reason is that you might want consistent values of the random bit in the instruction tests in order to determine whether a previous instruction did not apply because the random number did not match, or for some other reason. It is not difficult to see that each method can be simulated by the other, although probably not in the same number of locations.

DEK asked how an ant might know it has food. PL answered that it can tell it gets food when it moves successfully onto a pile of food. If having food is important to the ant's behaviour, then the ant had better branch to a separate part of its code at that point. (Of course, it is possible to write ant brains in which the ant doesn't know it has food.)

AM conjectured that the type of ant that is good for the single nest case would not do well when there are multiple nests. This was not clear to most of the class, although any species of ant will have its favorite type of terrain.

PL noticed in his early implementation that on the larger board from the Data Sheet, turning right performs better than turning left. More precisely, he had implemented a method that went straight ahead if it could, and then turned one way or another (so all ants turned right or all ants turned left). Using the "right turn" method gets to the food quickly, while the "left turn" method does not.

DEK then brought up the topic of how to solve a maze, given that you can leave a trail of where you've been. He thought it was called the Tremaux procedure for planar mazes.

The Tremaux procedure is designed for a single entity in a maze. One possible implementation is to send out one ant and let the others follow its trail. However, several ants cooperating should be able to do better.

DEK then asked what the best way of solving some simple situations would be. For example, on an empty board apart from a nest and a pile of food, what would be the best mode of attack. AM suggested two parallel paths, one for ants heading to the food, and the other for ants returning from the food.[1] Our ants can't drop food, or pass it to each other,

[1] AM called this a "Side of the road convention."

or climb over each other, so there is no possibility of a "bucket brigade" type approach. It was quickly realized that four parallel paths would do twice as well, although as SM pointed out, this is probably the best you can do given that the nest has at most four openings.

According to the specifications, it is impossible to have a dense trail of ants. More precisely, if all the ants are to move in lockstep, then there must be a vacant square in front of each ant.

Having run out of time, DEK left until next time the topics of how to handle turning corners, and how useful a "way station" might be.

Tuesday March 14

The class opened with a discussion of how the Great Races will be run. The first test of an ant brain is whether it does in fact get all the food for a given field. (The test will be repeated with different values of the random number seed, and with different fields.) Among those that always get all the food, faster is better. We aren't going to wait forever, so there will be some (large) threshold T so that any ant not succeeding by time T gets a score of T.

PL had suggested allowing groups to choose one of several ants to use depending on the particular field. "After all, the ants in Texas are different from those in California, which are in turn different from those in Brazil." This was an unpopular suggestion. AM thought that PL's motive was to be able to choose between "left turning" and "right turning" ants depending upon the board. PL defended his motives, saying that a strategy that works when there is lots of food is going to be inefficient when there is, say, only one unit of food. A lot of the overhead marking trails would be wasted.

DEK gave an assurance that the Great Races will take place on large boards with lots of food.

It seemed that groups were running into space problems for their ant brains. As AM pointed out, every bit of "memory" doubles the amount of code for which that memory item is critical. The main complaint seemed to be that ants can't see the scent on their own square. Remembering the scent on the basis of what they saw on the preceding move to that square necessitates significant duplication.

RK remembered a technique he had seen used on a Z80 processor for dealing with a very limited memory: If one procedure could make use of part of another procedure's code, it would execute a jump to the appropriate place *in the middle of the code.* In principle, we could do this with the ants, too, but DEK did not seem impressed with this technique of software engineering.

Somebody mentioned self-modifying code, and this led DEK to change the subject for a moment. He said that the normal programming environment for software development should have automatic profiling, so that the programmer can see where the machine cycles are being used most. He described how he profiled TeX in this manner, logging all users' statistics for a year on SAIL. The results showed where the program usually spent most of its time, which helped DEK later streamline the code. He also learned which error messages were most commonly generated, etc. Efficient profiling needs modified hardware support where counters are stored in correspondence with each procedure to indicate how often it is called. Unfortunately, such hardware is not available; self-modifying code is not at present approved of.

Returning to the problem at hand, ESC asked whether it is possible to get more than one ant out of the nest every two ticks. SJP said that the following sequence gives two out of every three ticks, assuming that there are no other ants interfering.

Time	Action
1	Top ant moves upwards out of nest.
2	Ant on nest rotates right, while ant above nest moves away.
3	Ant on nest moves rightwards, revealing an up ant on the nest.

From the resulting state, this sequence of moves may be repeated, again assuming that no stray ants block an opening. SJP proved this to be an upper bound on the ant throughput, by justifying that there cannot be any three consecutive ticks at which an ant emerges.[2]

DEK asked how an ant might follow a tortuous path. PL wondered to what extent technology should be proprietary. DEK mused that when he wrote TeX he had no qualms about making it public domain, since "he already had some theorems to his name." He was not sure what he would have done had it been his first contribution. Anyway, he encouraged people to be open about their ant-brained ideas.

To get the discussion going, DEK then brought up the subject of navigating to or from the nest. He asked how an ant might behave if, at every square, it knew from the scent how far it was (by a shortest path, measured in terms of physical distance rather than time) from the nest. If it was trying to get home, then the thing to do would be to look for an adjacent square that is closer to the nest. It is not difficult to convince yourself that the adjacent squares can be at most one unit further or nearer the nest than the current square. Further, since there are no odd-length paths from the nest to itself, it is impossible for two adjacent squares to be at the same distance from the nest.

We can't represent all the integers using scents, but we could represent them mod 3, and still know which squares were getting us closer to and which were getting us further from the nest. As DQ pointed out, getting the ants to construct such a pattern of scents is hard.[3]

DEK asked how else an ant might represent the information, "This way to the nest." RC suggested arrows. We have to distinguish between relative and absolute arrows, since (unless the code is written in such a way that the ant knows which way it's facing) an ant does not know up from down. A problem with this, which motivated the earlier complaint, is that an ant can't sense the scent of its own square. This makes it hard to follow paths, for example, without branching to 16 different parts of the code depending on the scent of the square to which a move is made.

RC also suggested a scheme with six kinds of tiles, one for each pair of distinct directions, telling how to turn; for example, the tile {N,E} means "go East if you come from the North, go North if you come from the East." Such paths can be traversed in both directions. DEK suggested that ants could perhaps come and go on the same path, stepping aside politely to let others get by (as cars do on mountain roads).

[2] After an ant emerges, the new ant faces upwards; if that ant also emerges, he'll be in the way at time 3.

[3] But a single ant could do it, given enough time: First paint "1" on all cells at distance 1, then "2" on cells at distance 2, then "0" on cells at distance 3, etc. On each round, traverse the scented squares, finding all unscented squares adjacent to color x and color them $(x+1) \bmod 3$, for $x = 0, 1, 2, 0, 1, 2, \ldots$ until you don't find any more. This (slow) operation could prepare the field for (fast) subsequent processing.

Thursday March 16

KAR announced Version 2.00 of the simulator. The major embellishment is the addition of an extra bit that determines whether the $abfstuv$ bits are for the square the ant is on, or the square it is facing. This greatly reduces space requirements for the ant brain logic.

A few suggestions from the audience for still more changes were silenced when DEK said that one of the most important general principles of computer science is "fitting within the constraints of the machine you're using."

The other recent development was the arrival of DEK's (real) ants from LA. He had put them into the ant farm the previous evening, and they were beginning to dig tunnels.[4]

DEK started the discussion by describing Tarjan's algorithm for dissecting a graph into biconnected components. A (sub)graph is *biconnected* if between any two vertices there are at least two disjoint paths. This may be relevant in our context, as an ideal method for moving ants between the nest and the food would be along two disjoint paths.

The algorithm uses a form of depth first search of the graph starting at any node. DEK could not remember all the details, but described roughly how it used "back-links" to identify biconnected components.

DEK said that a choice evident in Problem 5, and pertinent to AI in general, is whether to try to emulate nature or to try to find a better way. (Should a chess-playing program emulate Kasparov or should it use new alpha-beta pruning algorithms?) On one hand, it seems that following the patterns observed in nature may be too restrictive; on the other hand there are good evolutionary reasons why these methods have persisted and not others.

PL observed that "pre-empting is hard when we re-use code." DQ said, though, "We can't re-use code."

AM reported an interesting bug in his ant code. When an ant sees too many other ants, or a certain pattern of scents, it may become "comatose" and just stay put. Unfortunately, it was possible for the top ant on the nest to become comatose itself. (It thought it had moved out of the nest, but the move actually failed.) This ant would never leave the nest thus preventing any further ants from emerging. He noted that this could be avoided now that ants could sense the scent on their own squares.

RC reported another bug that occurs when two ants try for the same pile of food. One succeeds, while the other gets pre-empted and does not progress. The ant with food expects the other ant to get out of the way, while the empty handed ant stays put, transfixed by the food under the ant in front of it.

This story reminded DEK of the first machines to share disk drives, which at that time (1958) resembled juke-boxes. Once every few days, the machines would hang, effectively saying to each other, "After you, Sir."

The room for the Great Races has been reserved from 2pm to 6pm next Tuesday. Everybody should show up at 2pm, although the real action will begin at 2.30. (There is an exam in the classroom until 2pm, so don't get there too early.) DEK indicated that the Great Race fields will be biconnected with respect to the ants and the food, at least in the initial trials. (There may be cul-de-sacs.) DQ noted (somewhat ruefully, given the workings of his ant brain) that biconnectedness does not necessarily mean that all paths are two squares wide.

[4]Within a number of days, the ants had built an impressive network of tunnels in the sand. Several of them died, unfortunately, probably due to the poor food they serve in the baggage section of the plane.

DEK wondered what style of ant to expect on Tuesday. “How necessary is the random bit,” he asked. DQ said that his ant was deterministic, and so didn't use the random bit at all. But the class agreed that random bits were often helpful as tie-breakers in parallel algorithms that have a lot of symmetry.

DEK then asked, “How well would the ants do if I perturb the scents a little? After all, error recovery is an important task.” RK thought that such an expectation was not fair, since the machines being used are not fault-toler*ant*. PL was optimistic that his group's ant would be somewhat robust in this fashion. DS and AM thought that it would be more reasonable to expect ants to cope with a few cleared scents rather than replacing scents with other scents having other meanings.

DEK asked whether having all the food together helped. The consensus was “Yes.” DEK wondered how the food remained connected when some of it was taken. Most groups seemed to scent the intervening emptied squares.

DEK reminded the class that reports for this project are still required, and would be due Thursday March 23.

Tuesday March 21

The Great Races

At 2pm, DEK and KAR arrived at Sweet Hall. KAR was wearing a bow-tie and jacket; DEK looked the part in his referee's costume, complete with whistle. While the technical formalities of collecting the groups' brains[5] was going on, DEK played a tape he had procured especially for the Great Races from Michael Mauldin at CMU. The gist of the music was, “Ants, Ooooo. Ants, Ooooo. ...” (*The Ant Song* by Ronnie Wasp, WDVE.)

The races began at 2.30, by which time there was quite a crowd in the room. The CSD had been invited to witness the event, and friends and relatives also showed up. The live ants made it too, but didn't seem too impressed by their electronic counterparts.[6]

By the toss of a coin it was decided which order to present the ants. “Diligant,” a product of UH, DK, and DS, was introduced by DS. The colony of Diligants started by sending out three scouts to look for food. These scouts search in a random fashion, but are biased to walk straight more often than turn. Each scout uses an independent scent and is oblivious to the other two scouts. Once one of the scouts finds food, it returns to the nest along the trail it set up as it searched, and (hopefully) tells lots of other ants to follow that trail to the food.

Diligant was then demonstrated on the first of the initial fields:

```
BBB...................................
BB.................BB.................
B................BBBBB................
....BBB..........BBBBBB...............
....BBB...........BB..B......11111....
....BBB............B.B.BB....11111....
....BBB..............BBBBB...11111....
....BBB..............BBBBB...11111....
....BBB...............BBBB............
....BBB.......N........BBB......BB....
....BBB.................BB.....BBBB...
.............BBBB........B.....BBBB...
............BBBBBB..............BB....
............BBBBBB....................
.............BBBB.....................
......................................
......................................
......................................
```

[5]Ant brains, that is.

[6]You should have seen them dance to the ant music though!

This field is based on an example from the CMU race of 1987. The CMU rules were rather different. For example, they had many nest cells with only one ant per nest.

Using the projector, everybody was able to see what the ants were doing. Diligant seemed to be marching along pretty well, until a bug manifested itself. After about 350 iterations, with only one unit of food to go, the ant carrying that food item became deadlocked. It did not move from its square facing another ant, and this other ant did not move either.[7]

The next ant was "Newanto," written by ESC, SM, and MY. SM gave the introduction. This colony also starts by sending out some scouts but the scouts leave "reference trails" that use three different scents so that the direction of the trail can easily be established.[8] Once an ant discovers food on an adjacent square, it starts laying a new reference trail (using different scents) back towards the nest. Once such a "returning" reference trail hits the nest, ants follow it to pick up the food.[9]

Unfortunately, this ant also displayed some problems. It got all but six units of food back home on the first field, but then deadlocked. SM explained how they had changed their whole approach at 4am that morning, and so it was not surprising that bugs still appeared.[10] MY repeated the simulation with seeds of 19 and 31. The ants got all the food in 1276 and 1062 time units respectively. It was unclear whether he had actually tried all seeds between 0 and 31, though.

RK then introduced "Puissant,"[11] a creation of RC, RK, PL, and SJP. This is a complex ant. At first, many ants leave the nest to go looking for the food. Each leaves a trail that (hopefully) points towards the food.[12] When an ant hits food, it builds a "highway" back to the nest along the path it took initially. This highway is then the main thoroughfare for ants returning to the nest.

There is also a system of "frontage" roads that border the highway. When an ant without food bumps into one with food, it tries to leave the highway for a frontage road (building frontage road if there is none already), thus clearing the way for the food to get back to the nest. When there is space on the highway, an ant at the end of the frontage road turns onto the highway and proceeds towards the food.

Puissant was run on the first field, which it successfully cleared in 763 units of time. (Applause.)

The final entry was "Newdeterminant," written by AG, AM and DQ. DQ explained that their ant was deterministic, not using the random bit except in one case to decide between left and right. He claimed that on any field having a path from the food to the nest, and for which the food is connected, Newdeterminant would eventually find the food and return it

---

[7]When the field was twice repeated with different random number seeds, the ants got all the food back home both times.

[8]These trails made nice-looking plaid patterns on the screen; DEK awarded Newanto the "Beauty Prize." Five scents were actually used (two for corners), in such a way that the path remained traversable in both directions.

[9]The returning ants could not carry food themselves, just in case they found a dead-end and became stuck.

[10]In fact, all of the groups were busy hacking out their ants right up until 2pm. DEK repeated something Geoff Phipps had earlier said that he'd learned the hard way: "Never change the demo code within 24 hours of the demonstration."

[11]French for "powerful." There may also be some alliterative reason for this name.

[12]This group used their own resource file, displaying appropriate arrows for directional scents.

all to the nest.

Newdeterminant initially sends out as many scouts as there are exits from the nest. These proceed straight as far as possible, turning only when reaching a barrier or a square already visited.[13] If a scout reaches a square having no unvisited neighbors, usually by spiralling into a region of the field, it backtracks, marking all such squares as "uninteresting," so that other ants don't bother looking there.

When a scout does find food, it rushes back to the nest along the path it took initially, which may be relatively long (given that the search tends to spiral around). When this ant returns to the nest, an "optimizing ant" emerges. This ant follows the path to the food, but on the way back it takes short cuts, cutting off unnecessary loops. Once this ant reaches the nest, a stream of ants emerges to march along the optimized path towards the food. When they return with food they have priority, so that ants without food will step out of their way. If there is nowhere to step, these ants will head back to the nest too, empty-handed.

When run on the first field, Newdeterminant took a little while to find the food. DQ said that the ants have most trouble on open fields, where the ants tend to search a large portion of the field. However, the search takes linear time. Once the food was found, the troops marched out and brought it home relatively quickly. To get all the food back took 1760 time units on the first field. (More applause.)

The second field F2 was taken from a classic paper in the psychology literature about mazes for mice (C. J. Warden, "Distribution of practise in animal learning," *Comparative Psychology Monographs* **1**, 1922):

```
........B....................
........B....................
..B..B..B..B..B..BBBBBBBBBBBB
..B..B..B..B..B..............
..B..B..B..B..B..............
..B..B..B..B..B..BBBBBBBBBBBB
..B..B..B..B..B..............
..B..B..B..B..B..............
..B..B..B..B..B..BBBBBBBBBBBB
..B..B..B..B..B..............
..B..B..B..B..B..............
..B..B..B..B..BBBBBBBBBBBBB..
..B..B.....B..B..............
..B..B.....B..B..............
..BBBBBBBBBBBBBBBBBBBBBBBBB..
..............B........B..B..
..............B........B..B..
..BBBBBBBBBBBBB..B..B..B..B..
..............B..B..B..B..B..
..............B..B..B..B..B..
..BBBBBBBBBBBBB..B..B..B..B..
..............B..B..B..B..B..
..............B..B..B..B..B..
BBBBBBBBBBBB..B..B..B..B..B..
..............B..B..B..B..B..
..............B..B..B..B..B..
..BBBBBBBBBBBBB..B..B..B..B..
99B............N.B..B........
99B..............B..B........
```

Diligant and Newanto took forever to get into the top left corner, and we feared deadlock after they finally would succeed, so we turned them off. Newdeterminant and Puissant solved it in about 3000 steps — pretty good considering the 36 units of food.

[13]It leaves scents along its trail, with special scents at the corners corresponding to left or right turns.

The third and fourth fields were sort of dual to each other in food layout:

```
.......B111B.......    1......B...B......1
.......B1B1B.......    1......B.B.B......1
.......B1B1B.......    1......B.B.B......1
.......B1B1B.......    1......B.B.B......1
.......B1B1B.......    1......B.B.B......1
.......B1B1B.......    1......B.B.B......1
.......B1B1B.......    1......B.B.B......1
.......B1B1B.......    1......B.B.B......1
.......B.B.B.......    1......B.B.B......1
.........N.........    1........N........1
...................    1.................1
...................    1.................1
...................    1.................1
...................    1.................1
...................    1.................1
...................    1111111111111111111
```

Puissant got deadlocked on the second one, being overwhelmed by the variety of easily reached treats.

AG submitted a larger field, F5, whose solution was especially fun to watch:

```
.......BBBBB.............BBBBB....................
..................................................
..................................................
..................BB..............................
...................BBB............................
.................BBB..............................
..................BBB.............................
...................BBB............................
..................BBB.............................
.................BBB.........BBBBBBBBBBBB.........
..................BBB......BBB...........BBBB.....
...................BBB.......BBB.............BBBB.
..................BBB...........BBB........BBBB...
...................BBB............BBBBBBBBBB......
.....................B............................
...............BB.................................
.................BB...............99..............
....B.........B....BB...........999999999.........
.......B.........B...BB...........9...............
.B...B.....B...........BB........999999...........
B..B...........B..B......BB.........99............
........B...B..............BB........9............
..B.............B............BB......9............
.........B.....................BB.................
...B..............B..............BB...............
...................................BBBBBBBBBBBBBBB
B.....B....BBBBBBBBBBBBBBBBBBBBBBBBBB.B.B.B.B.B.B.
..................................................
BBBBBBBBB.BBBBBBBBBBB.BBBBBBBBBBB.BBBBBBBBBBBBBBBB
..................................................
..................................................
BB.BBBBBBBB.BBBBBBBBBBBB.BBB.BBBBBBBBB.BBBBBBBBBB.
..................................................
..................................................
.BBBB.BBBB.BBBB.BBBB.BBBB.BBBB.BBBB.BBBB.BBBB.BBBB
..................................................
....B.............................................
.....B..........BBBB..............................
......BBBBBBBBBB....BBBB....BBBB....BBBBBB........
......B.................BBBB....BBBB......BBBBB...
...................BBBBB.....BB...................
......B.....BBBBBBB.................BBBBBB........
......B.............................B.............
BBBBB........................BBBBBB.N.............
..........BBBBB...................................
..................B......B.......B.......B...B....
..................B......B.......B.......B...B....
..................B......B.......B.......B...B....
..................B......B.......B.......B...B....
..................B......B.......B.......B...B....
```

The time measurements are not reliable estimates (except in the case of Newdeterminant) since only one run was performed, with a single random number seed. A value of ∞ indicates that the ants didn't get all the food back on the *first* run attempted. A dash indicates that the run was not attempted (probably because it was clear from previous races that the ants wouldn't get all the food back).

The Initial Races

	Diligant	Newanto	Newdeterminant	Puissant
F1	∞	∞	1760	763
F2	∞	∞	2927	3144
F3	∞	∞	938	559
F4	∞	∞	832	∞
F5	–	–	11244	13516

The tie-breaking races were tougher. Field T1 had multiple nests (this was our "authentic" CMU example):

```
BBB...................................
BB.................BB.................
B................BBBBB................
....BBB..........BBBBBB...............
....BBB...........BB..B......11111....
....BBB............B.B.BB....11111....
....BBB..............BBBBB...11111....
....BBB.......NNNN...BBBBB...11111....
....BBB.......NNNN....BBBB............
....BBB.......NNNN.....BBB......BB....
....BBB.......NNNNN.....BB.....BBBB...
.............BBBBNN......B.....BBBB...
............BBBBBBN.............BB....
............BBBBBB....................
.............BBBB.....................
......................................
......................................
......................................
```

T2 had separated food:

```
...B...B................B....................B...
.B..BB.B................B......................B.
..B..B.B................B...................BNB..
B.NB9B.B................B....................B..B
..B....B................B..................B.9.B.
....BBB.................B..................B.BBB.
........................B..................B.....
........................B...................BBBBB
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
.......................B.B.......................
......................B...B......................
.................................................
....................B.......B....................
...................B....9....B...................
BBBBBBBBBBBBBBBBBBB....999....BBBBBBBBBBBBBBBBBBB
...................B....9....B...................
....................B.......B....................
.................................................
......................B...B......................
.......................B.B.......................
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
........................B........................
BBBBB...................B........................
.....B..................B........................
.BBB.B..................B.................BBB....
.B.9.B..................B................B....B..
B..B....................B................B.B9BN.B
..BNB...................B................B.B..B..
.B......................B................B.BB..B.
...B....................B................B...B...
```

T3 was not biconnected:

```
......B..B..B..B......
......B..B..B..B.....9
.........B.....B.....9
......B..B..B..B.....9
N.....B..B..B..B.....9
......B..B..B..B.....9
......B..B..B..B......
......B..B..B..B......
......B..B..B..B......
......B..B..B..B......
......B..B..B..B......
......B..B..B..B......
......B.....B.........
......B..B..B..B......
......B..B..B..B......
BBBBBBBBBBBBBBBBBBBBBB
......B..B..B..B......
......B..B..B..B.....9
.....................9
......B..B..B..B.....9
N.....B..B..B..B.....9
......B..B..B..B.....9
......B..B..B..B......
......B..B..B..B......
......B..B..B..B......
......B..B..B..B......
......B..B..B..B......
......B..B..B..B......
......B.....B.........
......B..B..B..B......
......B..B..B..B......
```

The results were:

The Tie-Breaking Races

	Diligant	Newanto	Newdeterminant	Puissant
T1	∞	639	939	604
T2	∞	∞	∞	2068
T3	–	–	1865	2099

Newdeterminant failed on T2 because it found food in each corner and went to sleep. When the food in the lower corner was removed, Newdeterminant finished in 12572 iterations.

From the results above it is clear that Newdeterminant and Puissant are the winners. I don't think that there is any scientific way to declare one better than another, since there is no objective way to select "typical" fields. The judges declared Newdeterminant the winner because of its elegant simplicity. Puissant received an honorable mention for graphics and originality.

PL described some ants that his group had implemented but not fully debugged, and so did not use in the Great Races. One idea was the "Zipper" ants. These ants proceeded in single file until they reached a barrier. From there, ants in this column alternately turned right or left, exploring the rest of the field.[14] When they found that all the neighbours had been visited (or were barriers) they began to backtrack, marking these squares as "uninteresting." It was quite impressive to see, on a big field, a large number of ants "cordoning off" the uninteresting areas. The path to the food was effectively the only interesting area left. This partial ant brain won the judges' Choreography Prize.

Newanto's designed by the gang
of Mahajan, Young and E. Chang.
But since all our code
went down the commode
the credit should go to ANTS! — MY

[14] It was not clear how they navigated around barriers, but PL assured us that this was not a problem.

The class closed with DEK congratulating the students on making it through the course, and thanking them for making it so enjoyable to teach. The students thanked DEK for teaching such a fun course. RK remarked that he had been worried that as a grad student he was getting too old to stay awake all night on homework; he said CS304 was so interesting that the worry became a myth.

As Teaching Assistant for the course, I must say that I enjoyed it too. Everybody was enthusiastic, and the problems were challenging enough to make for interesting reports.

Appendix A

The Trivia Hunt

Appendix B

Maps from Problem 3

Appendix C

Graphs of $g_k(x)$

Appendix D

The ANTS! Simulator

Version 2.02, April 1989[1]

D.1 Introduction

This writeup explains how to use the ant colony simulation program written for CS304 1989. The code was initially written by Don Knuth in WEB, then extended and modified by Ken Ross to run with color graphics under Lightspeed Pascal[2] on a Macintosh II system. Please notify `kar@cs.stanford.edu` of any bugs. Requests for copies of the software (which is not copyrighted) should also be sent to `kar@cs.stanford.edu`. We assume that the reader is familiar with the "antomata" specifications given in the Data Sheet.

New File Format

At the request of the class, the capabilities of ants were extended beyond those specified in the Data Sheet in one important way.

The brain file has a new format. The first eight hex digits are precisely the same as before, but now there is an extra ninth digit. Let the binary expansion of this digit be $wxyz$. Then the following meanings are attached to these bits:

- $z = 1$ means that the bits $abf\ stuv$ are taken from the ant's current location. $z = 0$ means that these bits come from the square the ant is facing, as before. Note that this allows ant ant to tell whether it is on the nest, whether it has food, and whether it is on a square with food. The status line only reports the bits for the traced ant's adjacent square; perhaps if there is yet another version, the current square will be displayed too.
- $y = 1$ means that the instruction applies only during a solar eclipse.
- $x = 1$ means that the instruction applies only if the power is off.
- $w = 1$ means that the instruction applies only if the ant sneezes.

[1] This manual is synthesized from the manual for version 1.00 and the upgrade notices given thereafter.
[2] Lightspeed Pascal (version 2.0) is necessary only if you wish to change the source code.

As usual, having multiple bits set to 1 corresponds to the logical AND of the above conditions. I reserve the right to change bits w, x and y without notice. (Actually, I think my present implementation is buggy, since it seems to ignore the values of w, x and y.)

D.2 Using the Program

The simulator package contains several items. There is the ANTS! application itself, which is the simulator. The file `pal.rsrc`, whose icon contains a jack-in-the-box, is a resource file that is accessed by the simulator. In addition to containing the information about colors and patterns (which you can change – see below) it also contains other information about windows etc. that are used by the program. (These you should not change.) Finally, there will be several sample ant brains and fields for you to play around with.

To start up the program, click on the ANTS! application icon. The simulator will read the default board from a file called `field.default` and a brain from `brain.default`.[3] Note that both these files (and all other field and brain files) must adhere strictly to the required format as specified in the Data Sheet and Section D.1. The simulator will not recover after reading a badly formatted file.

Once both the files have been read in, the board will be displayed, and there will be five menus from which you can select commands. There are a number of keyboard equivalents to the menu-selectable commands; they appear with the funny clover leaf symbol next to the corresponding item in the menu.

The Load Menu

This menu allows you to load in a new brain or a new field. As mentioned above, the formats of these files must be strictly adhered to.

The Commands Menu

The "go" command starts the simulator running at the current point. The menu bar will then display the word "Pause." Click on this word to pause the program. If you release the mouse button without selecting a menu item, the program will resume. Alternatively, select the "stop" option to stop the simulator. Note that the state will be saved, and that you may resume later even if you stop at a certain point. Typing "g" together with the funny clover leaf key is equivalent to selecting "go" from this menu.

The "single step" command may be also invoked by typing "s" together with the funny clover leaf key. It iterates a single step of the simulator. Note that a single step may be more that one time unit – see the Update Menu description below.

The "restart" command re-initializes the board to run the simulation once more. Note that restart increments the random number seed, so that you will get a different run from the previous one. To manually change the seed see the Options Menu below. The option "repeat" is like restart, except that the random number seed is not incremented, so that the subsequent run will be exactly the same as the previous one.

[3] If these are not present, the simulator will crash.

The "quit" command does the obvious.

The Update Menu

This menu allows you to select the step size for a single-step command. It also sets the frequency of screen updates for the go command; because the simulator can work a lot faster than the I/O it generates, you should increase this number if you are not interested in detailed movements. Setting this to infinity will cause the program to update the screen only when all the food is gone. Note that single stepping when the step-size is infinite does nothing.

The Options Menu

The "scented backgrounds" option allows you to select whether to have black backgrounds (actually, the background color you can change — see below), or backgrounds of the same color as the scent of the square. Try experimenting with this, and see its effect on food and ant symbols. Unless you have defined special symbols for scents, the simulation looks nicer with scented backgrounds, although using black backgrounds may be helpful in locating ants within a multi-colored board.

The "change seed" option allows you to enter an arbitrary seed. By default, the seed starts at zero. Note that changing the seed will reset the board. Warning: the seed should be in the range 0 to 32767.

You may redraw the screen by selecting this item from the menu. Hopefully you won't need to do this too often because the screen should be correct if the simulator is working.

Finally, you can hide the status display. This is useful on large fields for which the status information encroaches onto the field.

The Breakpoints Menu

The ants simulator allows you to set breakpoints. A breakpoint is a memory location within the ants' brain. When the simulator is running, either because "single step" or "go" was selected, the program will pause when an ant reaches a breakpoint. (This means that the instruction at the breakpoint location *applies*, as defined in the Data Sheet.) There are two ways breakpoints can be used:

- If an ant is being traced (see below) then the simulator will break when *this* ant reaches a breakpoint. If the traced ant returns to the nest without ever reaching a breakpoint (and there was in fact some breakpoint set), then the simulator will break when the ant enters the nest.

- If no ant is being traced, then the simulator will break as soon as *any* ant reaches the breakpoint. When this occurs, an ant that reached the breakpoint will be automatically selected as traced, and thus highlighted.

You may set a breakpoint, show breakpoints or clear all breakpoints using this menu.

The State Menu

The simulator allows you to manipulate the state directly. You can save and restore, and also edit the current state. Don't try to edit the state files, since there is only rudimentary error checking built in to the routine that restores state files. (This part of the code was written in a hurry!) Besides, it is probably easier to make changes by using the edit feature.

One important note is that the state file does *not* include the ant brain information. Hence you must load the appropriate ant brain *before* restoring the state. The simulator will refuse to restore a state if the title of the ant brain does not match the title stored in the state file. For this reason, you may want to change the title of the brain every time you make a change, say by incorporating a version number. Of course, you can fool the simulator by keeping the same name. You should do this only if you are 100% sure that the program counters of the saved ants still correspond to the same instructions, or if you plan to change all the program counters manually.

To edit the current state, select "Edit" from the "State" menu, or type "e" together with the funny clover leaf symbol. The menu bar will change to give you the "Edit" menu, and the cursor will change too. There are five different ways that you can edit the state, depending on which item from the edit menu is selected. Clicking outside of the field will do nothing.

- If "Change scent" is selected, you may click on a square to increment its scent mod 16. You can't change the scent of a barrier, which always has scent 0.
- If "Toggle ant" is selected, then clicking on a square will remove an ant if one is present (except at a nest), or create an "up ant" if the square does not contain an ant (and is not a barrier). The PC of a newly created ant is 0.
- If "Rotate ant" is selected, then clicking on a square with an ant will rotate it ninety degrees counter-clockwise.
- If "Set ant PC" is selected, then clicking on a square containing an ant will result in a box appearing on the screen, asking you to supply a PC in hex. The current PC will be displayed as the default.
- If "Toggle ant food" is selected, clicking on a square with an empty handed ant will give it some food. Clicking on an ant with food will take it away.

Selecting "Exit edit" will take you back to the main menu.

In order to place food, barriers and nests, you have to edit the initial field file, as before.

Note that the edit feature allows us to perturb scents. If your ant brain is robust with respect to perturbed scents then say so, and we may incorporate this into the tie-breaking tests.

Tracing Ants

While the simulator is not iterating, you may click the mouse on any ant appearing on the screen. The color of the ant will change, indicating that this ant is being traced. Its program counter (in hex) and the remainder of the comment line in the ant brain file will be displayed at the top left corner of the screen. Also displayed are the input bits (apart from the random bit) corresponding to the square the ant is facing. This display will be updated after every step, until the ant returns to the nest, or until the trace is disabled. Clicking on another ant will trace that ant instead; you may trace only one ant at a time. Clicking any place where there is no ant will turn off the trace.

D.3 Changing the Display

The colors and shapes used by the simulator to draw the various objects may be changed. Please bear in mind the following suggestions:

When Debugging, use a different color for each scent, and don't try to get too fancy with the display. Once you are convinced that your ants are doing what they should, you can then worry about the display.

For the Presentation, use the same color for different scents only if they have the same meaning. Use different shades of the same color only when there is some semantic link between the two scents. The aim here is to give an onlooker the ability to comprehend your colony easily.

In either case, make sure that all symbols are clearly distinguishable from one another, either on the basis of color or pattern. In particular, make sure that ants don't get "lost" in their background, and that ants with food are distinguishable from those without.

Standard colors and patterns are supplied with the program in the resource file `pal.rsrc`. Don't edit this directly; make a copy first. You will need to use `ResEdit` to modify a resource file; such files cannot be changed from within the simulator itself. Double-clicking on the icon for `pal.rsrc` should invoke `ResEdit`.

To modify a pattern, select a particular pattern (of type "PAT") using the mouse. (More on knowing which pattern is which later.) A pattern is an 8 by 8 grid of pixels. Clicking on a particular pixel will toggle its value. You can even paint or erase a region by holding down the mouse button while you move the mouse.

Modifying the colors is more of a hassle. You basically have to do it using a hex editor. Click onto the resource of type "pltt" that has the resource number given in Figure D.1. Each entry in the palette consists of eight 4-hex-digit fields. The first entry contains the number n of colors in its first field, and zeroes elsewhere. Subsequent entries correspond to the colors $0, 1, \ldots, n-1$.

A color entry consists of three values between `0000` and `FFFF` for the red, green, and blue components of the color, respectively. The remaining five values are reserved, and should all be zero. If you want to play around with various colors, select the "monitors" item in the control panel.[4] Make sure your monitor is set to enable color, preferably in the 256-color mode.

What is a resource? Well, one of the things that the Macintosh computers emphasize is the use of resources, i.e., data to be input from a standard library. Resources can have a

[4]You get this by pulling down the menu labelled with the apple symbol. Click on the "change color" space to get the color wheel.

#	Loc.	default	function	#	Loc.	default	function
0	`010`	black	background	16	`110`	white	scent 0
1	`020`	dark grey	barrier	17	`120`	mauve	scent 1
2	`030`	yellow	normal ant	18	`130`	purple	scent 2
3	`040`	orange	ant with food	19	`140`	dark purple	scent 3
4	`050`	fawn	ant on nest	20	`150`	light blue	scent 4
5	`060`	maroon	ant being traced	21	`160`	mid blue	scent 5
6	`070`	mid grey	color of text	22	`170`	*sky blue*	scent 6
7	`080`	pink	one unit of food	23	`180`	navy blue	scent 7
8	`090`	dark pink	two units of food	24	`190`	turquoise	scent 8
9	`0A0`	dark pink	three units of food	25	`1A0`	bright green	scent 9
10	`0B0`	dark pink	four units of food	26	`1B0`	light purple	scent 10
11	`0C0`	red	five units of food	27	`1C0`	khaki	scent 11
12	`0D0`	red	six units of food	28	`1D0`	off green	scent 12
13	`0E0`	red	seven units of food	29	`1E0`	green/grey	scent 13
14	`0F0`	red	eight units of food	30	`1F0`	terracotta	scent 14
15	`100`	red	nine units of food	31	`200`	brown	scent 15

Figure D.1: Object colors: resource id is 27773.

variety of types. For example the type "pltt" is a color palette resource, the type "PAT" is a pattern resource, the type "WIND" is a window resource, and so on. The idea behind resources is to separate most of the user interface from the inner workings of the program, so that it may be changed without affecting the logic of the program.

So the palette and patterns (among other things) used by the simulator are kept in the resource file `pal.rsrc`. How do you know which color or pattern is which? By looking it up in the tables in Figures D.1 and D.2.

id	function	id	function
18937	background	31257	scent 0
10337	barrier	30467	scent 1
32173	ant facing up	5198	scent 2
29469	ant facing right	7026	scent 3
26280	ant facing left	20924	scent 4
31575	ant facing down	31607	scent 5
30620	one unit of food	22445	scent 6
31846	two units of food	17236	scent 7
18503	three units of food	28221	scent 8
9456	four units of food	18666	scent 9
31112	five units of food	19894	scent 10
6694	six units of food	6551	scent 11
10284	seven units of food	30271	scent 12
11876	eight units of food	19154	scent 13
28094	nine units of food	27488	scent 14
		31058	scent 15

Figure D.2: Resource id numbers for various patterns.