On the crossing number of K(9,9)

SURF 2002 Final Report

Virginia Vassilevska
Mentor: Richard Wilson

10/1/02

Abstract

Because of the large success of very large scale integration (VLSI) technology many researchers have focused on optimizing the VLSI circuit layout. One of the major tasks is minimizing the number of wire crossings in a circuit, as this greatly reduces the chance of cross-talk in long crossing wires carrying the same signal and also allows for faster operation and less power dissipation. The question of finding the minimal number of crossing wires can be abstracted to a graph theoretical problem of determining the minimal number of edge crossings in a drawing of a given graph. The crossing number problem is especially interesting for complete bipartite graphs, for which Zarankiewicz conjectured a formula in 1954 that still remains unproven. In 1993 Woodall used a computer program to solve the smallest then unknown case - that of K(7, 7) thus proving the Zarankiewicz conjecture for K(m, n) with $min(m, n) \leq 8$. The smallest now unsolved case is that of K(9, 9). The purpose of this project is to write a program that reproduces Woodall's results and further checks the conjecture for K(9, 9).

1 Introduction

Imagine you are an electrician and have multiple wires to connect but the properties of the device you are building force you to minimize the number of crossings between wires. Or, imagine you are a civil engineer and are planning the construction of highways and you care that they cross and wind about each other at as few places as possible. In both cases we are interested in the minimal number of crossing points. Graph theory is an area of mathematics that provides us with the tools to approach problems such as this one.

1.1 Some Definitions

Graphs are abstract mathematical objects composed of points, called *vertices*, and lines, called *edges*, connecting the vertices. We can represent our wires or bridges as edges in graphs and can ask ourselves: What are the crossing numbers of these graphs. The *crossing number* of a graph G is defined as the minimal number of crossings of edges one gets by drawing G in the plane. It is assumed that the edges in a drawing are nonselfintersecting and that every two edges have at most one point in common: either a common vertex or a crossing.

For many applications (such as in VLSI circuit design) we are particularly interested in the crossing numbers of the so called bipartite graphs. A complete bipartite graph $K(m,n)$ is a graph with two subsets M and N of its vertex set V, so that M and N are disjoint, their union is the whole V, there are no edges among vertices inside any of the two sets, yet each vertex in M is connected by a unique edge to each vertex in N, and vice versa. A bipartite graph is a subgraph of a complete bipartite graph.

1.2 Zarankiewicz' Conjecture

In 1954 Zarankiewicz proposed a formula for the crossing number of a complete bipartite graph:

$$crK(m,n) = \lfloor\frac{m}{2}\rfloor\lfloor\frac{m-1}{2}\rfloor\lfloor\frac{n}{2}\rfloor\lfloor\frac{n-1}{2}\rfloor \tag{1}$$

It can be shown that $crK(m,n) \leq Z(m)Z(n)$, where $Z(m) = \lfloor\frac{m}{2}\rfloor\lfloor\frac{m-1}{2}\rfloor$ is the Zarankiewicz number. To see this, arrange the m and n vertices along the x- and y-axis respectively with half of each set on each side of the origin. Then connect them with straight lines (Fig. 1). This arrangement is due to Zarankiewicz [5]. It gives $Z(n)Z(m)$ as an upper bound for the crossing number of a complete bipartite graph. Except for some special cases, nobody has yet been able to prove that this is also a lower bound. In 1993 it was known that Zarankiewicz' conjecture holds for all $K(m,n)$ with $min(m,n) \leq 6$. In that year D.R. Woodall published his result [1] that the conjecture also holds for $K(7,7)$, thus showing that it holds in fact for all $K(m,n)$ with $min(m,n) \leq 8$. This was one of the greatest achievements since 1969 when Kleitman [6] showed that Zarankiewicz' conjecture applies to $K(5,n)$ for all n.

1.3 What Woodall Used

1.3.1 Some Theorems

In the development of his program Woodall used three theorems that were known about crossing numbers of bipartite graphs:

Theorem 1 *Every $G = K(m,n)$ with $crG = k$ contains a drawing of $K(m-1,n)$ with a number of crossings $cr \leq \frac{k(m-2)}{m}$.*

Theorem 2 *If m is even and the Zarankiewicz' conjecture holds for $K(m-1,n)$, then it holds for $K(m,n)$.*

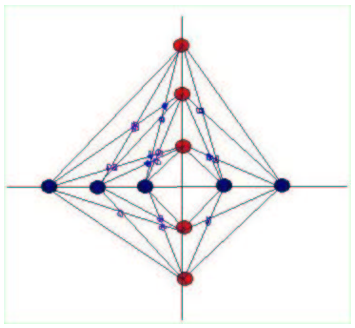


Figure 1: The drawing of $K(5,5)$ that shows that $crK(5,5) \leq Z(5)Z(5) = 16$.

Theorem 3 *If m and n are odd and $m' < m$ is even, such that the Zarankiewicz' conjecture holds for $K(m'+1,n)$ and $K(m-m',n)$, then in any drawing of $K(m,n)$ that includes a drawing of $K(m',n)$ with $Z(m')Z(n)$ or fewer crossings there are at least $Z(m)Z(n)$ crossings.*

The second theorem is particularly useful since it means that one only needs to consider the cases in which m and n are both odd. The third theorem means that if $K(m,n)$ is to be a counterexample to the Zarankiewicz' conjecture, then in some drawing all $K(m',n)$ subgraphs (such that m' is even and the conjecture holds for $K(m'+1,n)$ and $K(m-m',n)$) have more than $Z(m')Z(n)$ crossings. In particular this means that if $K(5,5)$ was to be a counterexample, then, since the conjecture holds for $K(3,5)$, there exists a drawing of the graph so that all $K(2,5)$ subgraphs have at least one crossing.

1.3.2 The cyclic order graph $\mathbf{CO}_n$

Consider a drawing of $K(m,n)$ and take a vertex $v \in M$. Vertex v is adjacent to all vertices in N, and going clockwise one can order them according to which edge leaves v when. Suppose that we have a drawing of $K(2,3)$ such as in (Fig. 2). Then if the vertices in N are numbered one can give the ordering using their numbers: 213 in the case of v in the figure.

Now suppose there is a crossing point P between two edges ax and by in a bipartite graph, where $a,b \in M$ and $x,y \in N$. If there are no crossings on the segments aP and bP, then one can open out the crossing so that there is

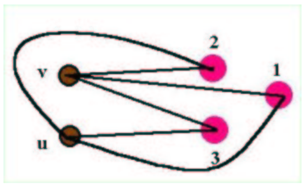


Figure 2: Vertex v here has an ordering of the edges 213 and vertex u has an ordering 231.

no crossing point anymore and no more crossings have been created(Fig. 3). When this is done, the cyclic ordering of one of the vertices a or b is changed by switching x and y. Thus, in attempting to planarize a graph, one switches the places of digits in the orderings corresponding to the elements of M. It is then useful to define the cyclic order graph CO$_n$: this is a graph whose vertices are the $(n-1)!$ cyclic orderings of n elements, and in which two vertices are adjacent if and only if one can be obtained from the other by switching two digits. CO$_n$ is vertex-transitive, n-regular, and when n is odd, bipartite. If $a \in V(CO_n)$ then $\bar{a}$ is the reverse ordering of a, called the *antipode*. The *ab-antipath* is defined as the path from a to $\bar{b}$ in CO$_n$. The distance between two vertices is just the smallest number of edges in a path between them and the *antidistance* $\bar{d}(a,b)$ of a and b is the distance between a and $\bar{b}$ or between b and $\bar{a}$. M is an (m,n) set if $|M| = m$ and if its elements are all in $V(CO_n)$. The sum of the antidistances between all pairs of elements of an (m,n) set M is called the *antisum* $\bar{A}(M)$.

The reason behind the importance of the antidistance function lies in the fact that if one has a drawing of $K(2,n)$ on sets $\{A,B\}$ and V_n, such that the clockwise orders of edges leaving A and B respectively are a and b, then $cr(drawing) \geq \bar{d}(a,b)$ [1]. Now, if one has $K(m,n)$, a lower bound for its crossing number is the sum of the crossing numbers of all $K(2,n)$ subgraphs, which is greater than or equal to the antisum of the corresponding (m,n) set. This antisum is an example for a *lobspacron* function: a function that assigns to each (m,n) set an integer that is a lower bound for and has the same parity as the crossing number of any good drawing corresponding to the set. The fact that the antisum has the same parity as the crossing number of the corresponding $K(m,n)$ is based on the fact that for a $K(2,n)$ we have that $cr(drawing) \equiv \bar{d}(a,b) \pmod 2$ [1].

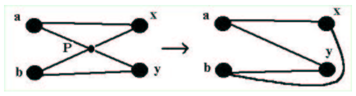


Figure 3: The crossing is opened up and the order of x and y in the ordering of b is changed.

Woodall also proves the following two theorems which complete the theoretical preparation for his program:

Theorem 4 *(a) If $a \in V(CO_n)$, then $d(a, \bar{a}) = Z(n)$.*
(b) Every $(3, n)$ set has antisum at least $Z(n)$.
(c) If m and n are both odd, then the antisum of a (m, n) set is odd when $m \equiv n \equiv 3$ $(mod\ 4)$ and even otherwise.

Theorem 5 *Let m and n be odd integers and $m' < m$ be even so that every $(m' + 1, n)$-set has antisum at least $Z(m' + 1)Z(n)$ and every $(m - m', n)$-set has antisum at least $Z(m - m')Z(n)$. Then if an (m, n)-set contains an (m', n) subset with antisum $Z(m')Z(n)$ or less, it has antisum at least $Z(m)Z(n)$.*

2 The Program

2.1 General Principles

Because of the way it was constructed, Dr.Woodall's program was designed to handle (m, n) sets with $n \leq 7$. In order to make it possible to consider larger sets, we wrote a new program, which is based mostly on Woodall's ideas, yet is quite different in its structure from Woodall's program. The main goal of the program is to check whether there exists a (m, n) set with antisum smaller than $Z(m)Z(n)$. Since considering all $((m - 1)!)^n$ (m, n)-sets is clearly impractical (this number is roughly 3×10^{41} for (9,9)), we want to limit the number of sets we look at. There are several types of isomorphisms between the sets. The antisum of a set is invariant under reordering of the elements, under applying an element of S_n to all of the elements simultaneously, and under taking the inverses of all of its elements. Under those types of isomorphism, the set of (m, n) sets is subdivided into several isomorphism classes. We want to find one representative of each isomorphism class such that its antisum is less than $Z(m)Z(n)$. Let the lobspacron function f we use be the antisum. We pick our representative *abcde...* according to the following two criteria:

1. it minimizes the sequence of lobspacrons $f(abcde...)$, $f(abcd...)$, $f(abc...)$, $\ldots$, $f(ab)$
2. it is the lexicographically smallest among the ones satisfying condition 1.

When we search for class representatives we can take into account theorems 1 and 4. Then we have certain bounds for each set abcd..., $f_{MAX}(k)$ for a set w of size k: if w is a subset of the representative we are looking for, then $f(w) \leq f_{MAX}(k)$. From theorems 1 and 4 we have such a bound for all $2 \leq k \leq m$. The bounds for (9,9) are $\{254, 197, 146, 104, 68, 40, 20, 6\}$ where 254 is the value for $k = 9$ and 6 for $k = 2$.

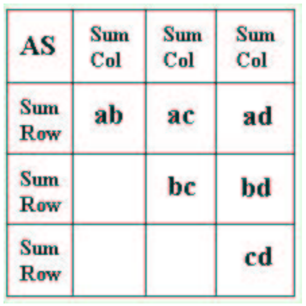
We call each (k, n) set a Word of length k consisting of Letters of length n. The search for class representatives proceeds constructively. We start from the smallest lexicographically Letter 012345... and add a second Letter. At each stage after a Letter is added to the current Word of length l two things are checked: whether $f(\text{new word}) \leq f_{MAX}(l+1)$ (check1) and whether there is no Word isomorphic to it that would be a better representative according to our criteria (check2).

2.2 Some Details

During the course of the project various versions of check1 and check2 were designed. I will describe the current best versions.

2.2.1 check1

The program maintains a matrix of the antidistance values of all pairs of Letters in the current Word. When check1 checks whether the antisum of the current Word does not exceed the bound f_{MAX}, it has to calculate all of the antidistances and when it does it updates the matrix. If the current Word is *abcd...*, the first row starting from the first column consists of all antidistances involving a. The second row starting from the second column contains all antidistances involving b, except for ab. The third row starting from the third column contains all antidistances involving c except for the ones in the second column, and so on. The entries in the zeroth row starting from the first column are the sums of the entries in that column, the entries in the zeroth column are the sums of the entries in the respective rows, and the $(0, 0)$ entry is the sum of all antidistances - the antisum of the current Word. The matrix is further used in check2.



AS	Sum Col	Sum Col	Sum Col
Sum Row	ab	ac	ad
Sum Row		bc	bd
Sum Row			cd

Figure 4: The table used in check1.

Check1 also checks whether the newly added Letter falls within certain bounds we calculated. Suppose the current Word is $abcde$ and the Letter next to be added is f. Then we know that if $abcdef$ is to be a subset of the representative we are looking for, $abcdf$, $abcfe$, $abfde$, $afcde$ and $fbcde$ all have to have antisums greater than or equal to that of $abcde$. If we sum up all of these requirements, except the one for $fbcde$ we get that $4af \geq 4abcde - (abcd + abce + abde + acde) - 3(bf + cf + df + ef)$, where by the Words we mean their antisums. After a little simplification we get that $af \geq AS + \sum_{x \neq a,f} ax - 3\sum_{x \neq f} xf$, where AS is the current antisum, before f is added. In general, $af \geq AS + \sum_{x \neq a,f} ax - (currentLength - 2)\sum_{x \neq f} xf$.

We also calculated an upper bound for af: $af \leq f_{MAX}(currentLength + 1) - AS - \sum_{x \neq a,f} ax$. These two bounds make some difference in the running time since less Words are considered in check2, which is by far the slower function. We have also incorporated Theorem 5 in our program which substantially decreases the running time.

2.2.2 check2

The purpose of check2 is to check whether the given Word has Words isomorphic to it that would be better as representatives. In order to do this, we need to check all possible orderings of the Letters in the Word, out of those to select the ones that have lobspacron sequences smaller than or equal to those of the original ordering, if there are no orderings with smaller lobspacron sequences, but there are some with the same, to check whether there is a way to map these into something lexicographically smaller than the original Word. To do the ordering part at each step we omit a Letter from the Word. For example, if it was originally $abcde$ we may choose to omit d. This means that we are about to check all orderings in which d is the last Letter. After omitting the Letter we check whether the subword has a smaller lobspacron value. If it does, we reject the Word. If it has a greater value, we reject the ordering, return the omitted Letter and omit a new one. If the lobspacron value is the same, we run our mapping function on it to check whether the ordering can be mapped into one lexicographically smaller than the original one.

For an (m, n) Word *abcd...* and a given current ordering $a'b'c'd'$... one has to check $2n$ mappings: the ones that map the first Letter a' into the zeroth Letter a_0 (which by our construction will be the same as a) and the ones that map a' to $\bar{a}'$ and then map that to a_0. This is since the representative should be the lexicographically smallestand thus its first Letter must be a_0. The second n maps are applied to the inverses of the remaining Letters in the Word. It is easy to see that we need not find all such maps. We only need to get one map of each type and afterwards just apply shifts $0 \mapsto 1 \mapsto 2 \mapsto \ldots \mapsto n \mapsto 0$ to the result of the previous mapping. Furthermore, we can save the representatives of the equivalence classes obtained by applying the $2n$ maps to all $(n-1)!$ Letters. We can also save the maps that send a given Letter to the representative of its equivalence class. This does not increase the memory usage significantly (just by about an eighth), yet can decrease the CPU time substantially. The reasoning is as follows: suppose you want to check Word $a'b'c'd'e'$ versus the original Word *abcde*. You first use any map to map a' into a_0 and then apply it to b' to get, say, b''. Then you check in the list *replist* you created to find the representative corresponding to b''. If that representative is lexicographically smaller than b, then *abcde* cannot be the representative of its class, so you reject it. If the representative is larger than b, then you try a new reordering of *abcde*. If the representative is b, then you consider c'. Apply one of the maps in the list of the mappings in *replist* that yield b from b' to c' to get c'' and then check the representative corresponding to c'' in *replist* versus c. If the representative is not c make the same decisions as before, if it is c, then consider d' applying the same map to it as to c'. This procedure continues until either the current Word is rejected, the current ordering is rejected or all Letters in $a'b'c'd'e'$ have been checked, in which case a new ordering is attempted, until all orderings are exhausted. After that, if the Word has not been rejected, it must be a representative.

Further major adjustments to check2 can be made since we do not want to repeat operations unnecessarily. For example, we may want to exclude the possibility that if in the previous level check2 checked *abc* and determined that no reorderings give a better Word, we exhaustively check all orderings of *abcd* in which a, b, c in some order are first in the Word, and thus repeat what was done in the previous stage. In such cases we only need to run the mapping part of check2, not the reordering of *abc*. One problem is that currently the reordering and the mapping part are integrated one into another, and such an adjustment would be a major change possibly involving a lot of bookkeeping. We are also currently working on developing better bounds to incorporate into check1, since that would limit the number of calls to check2 even more and that would be important since check2 is the slowest part of the program.

3 Results and Conclusions

One of the first things we did after the program was first constructed was to run it on the cases Woodall ran his program in order to check that the two programs gave the same results. This is useful in two aspects. Firstly, since no one else has looked at Woodall's program and no one else has so far reproduced his results, it is good to see whether we would get the same things independently. Secondly, if Woodall's results are correct, then we would have a sign that our program is also correct. We ran the program on $(5,5)$, $(5,7)$ and $(7,7)$ and we got the exact same results as Woodall. Unfortunately, since our program was designed to handle more cases than Woodall's, it has a worse time complexity. Woodall was able to save certain data that we are not able to store with a case 5000 times greater. We are currently still working on improving the speed.

After we had reproduced Woodall's results we decided to attempt $(5,9)$, the smallest until now set with unknown minimal antisum. Since the program was going to take substantial amounts of CPU time, we split the job into 167 branches according to which was the second Letter in the Words we looked at. Running the program on $(2,9)$ with the same $f_{MAX}(2)$ as for $(5,9)$ showed us that there are only 167 possible second Letters and we only used those. We ran those branches on 30 computers, 20 of which were 990MHz and 10 of which were 900MHz. We also adapted the program so that it can handle ranges for the third added Letter, in case one of the larger branches took too long to finish and we would have to split that branch onto more computers. Altogether, the program took 123 days 17 hours 35 min and 3.4 sec of CPU time and returned no counterexamples. This was before the adjustments were done to check2 and before the addition of the two bounds to check1. Now the program should run slightly faster. The algorithm is easily parallelizable and this is one aim for the future.

The result of running the program is the following

Theorem 6 *a) Every* $(5,9)$ *set has antisum at least* $Z(5)Z(9) = 64$.
b) No $(7,9)$ *or* $(9,9)$ *set can contain a* $(4,9)$ *subset with antisum* 32 *or less.*

Part a is a direct result of running the program since the program returned no counterexamples. Part b follows from theorem 5 with $m' = 4$ for both cases. In the first case we know that all $(3,9)$ [1] and all $(5,9)$ (from the program) sets have antisums at least $Z(3)Z(9)$ and $Z(5)Z(9)$ respectively, in the second case we only need the result for $(5, 9)$. This theorem would be useful if one wants to attempt $(7, 9)$ and $(9, 9)$. This would reduce the CPU time greatly since more sets would be eliminated from earlier on. To be able to run $(7, 9)$ in reasonable time, however, some major adjustments to the program have to be made.

4 Acknowledgments

I would like to thank

- my mentor and advisor Dr. Richard M. Wilson for his help and support
- Dr. D.R. Woodall for providing us with the source of his 1986 program
- the SURF committee for the opportunity to work on this project
- Mr. Peter Adams for sponsoring me.

References

[1] D.R. Woodall, Cyclic-order graphs and Zarankiewicz's crossing-number conjecture. *J. Graph Theory* 17 (1993) 657-671

[2] J. Pach, G. Toth, Graphs drawn with few crossings per edge. *Combinatorica* 17 (3) (1997) 427-439

[3] R. Guy, The decline and fall of Zarankiewicz's theorem, *Proof techniques in graph theory*, 1969, Academic Press, New York

[4] S.N.Bhatt, F.T. Leighton. A Framework For Solving VLSI Graph Layout Problems. Journal of Computer and System Sciences 28 (1984) 300-343

[5] K. Zarankiewicz, On a problem of P. Turán concerning graphs. *Fund. Math.* 41 (1954) 137-145

[6] D.J. Kleitman, The crossing number of $K(5, n)$, *J.Combinat. Theory* 9 (1970) 315-323