Project Number MA-BZS-022

Winkler Percolations - A Combinatorial Analysis

A Major Qualifying Project report

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Jason A. Gronlund

John W. Hajeski

Date: April 28, 2005

Approved:

Professor Brigitte Servatius, Major Advisor

1. percolations
2. combinatorics
3. renormalization

Abstract

Winkler percolations, also known as coordinate percolations, are digraphs generated by random 0-1 sequences. The percolation's nature is determined by the frequency of 1's in the sequences, governed by a fixed probability p of occurrence. An open question is at what p is the completeness of the percolation no longer ensured. We look into this question using a combinatorial study of small finite examples, and the self-similarity of this model is analyzed using methods of renormalization group theory.

Acknowledgements

The authors wish to thank all who contributed to the realization of this work: our advisors Professor B. Servatius and Professor G. Sarkozy, and Professor Peter Winkler for taking the time to speak with us.

A Note On The Pictures

Many of the images within this report were created using the Visual Basic Winkler Percolation generator written for this project, available for use at:
http://www.wpi.edu/∼johhaj/MQP/percolator.exe
(.NET framework or latest version of Visual Basic required)

Contents

List of Figures

List of Tables

1 Introduction

The terms ‘percolation’ or ‘percolation theory’ both refer to “fluid flow (or any other similar process) in random media [5].” Much like a coffee percolating machine, a trail is being followed from a point of origin (the hot water dripping downward), until some destination is reached (the coffee pot). The random media can be thought of as the positioning of the coffee grounds, which inhibit the flow of water. Percolations are drawn upon lattices or grids; the starting point is defined, and the goal is typically the outermost edges of the lattice (away from the adjacent sides of the starting point), or unto infinity, if no such edges are defined (infinite lattices). Percolations are created as models for problems, and the existence of a trail from beginning to end is relevant (if not equivalent) to that problem’s solution.

Winkler Percolations [3], also known as “**coordinate percolations**”, are built based upon two random infinite strings of 0’s and 1’s. This type of percolation is a new one; the only major work concerning them, Peter Gács’s “Compatible Sequences and a Slow Winkler Percolation”, was published in late 2004. The purpose of this project is to give a thorough yet general synopsis of Gács’s paper, and draw upon the research suggestions that both he and Peter Winkler, the creator of this framework, have offered. This young yet potent subject requires the kind of “mathematical groundwork”, so to speak, that is taken for granted in much older subjects. These tasks will come to light as we begin to unravel the definitions and inherent structures built into coordinate percolation, and from these crude yet effective methods we will devise new observations and conclusions. After our initial fact-finding and background searches, we have found research within percolations at large has not been very extensive. We hope that our contribution to the subject, as tiny as it might be, may inspire others to gain more curiosity about this little known, but very powerful, structure.

1.1 Percolation - an extensive definition

A generic percolation is built upon a two-dimensional lattice (size n x n) and the determination of its structure is based on two arrays (or strings), both of size n. This lattice is itself merely a graph $G(V,E)$, given a specific arrangement on the two-dimensional plane, to represent our percolation. $|V| = n^2$, and each is placed at evenly-spaced locations, in a grid. The number of edges, $|E|$, is not specifically defined. Depending on the placement rules (that is, the determination of how the vertices should be connected), there can be many or few edges on a percolation graph. Very frequently, the graph may be directed; movement would only be restricted in a single direction (towards the goal), much like our trickling coffee that can only go further down toward its destination. These "placement rules" are based upon two arrays (or strings), X and Y, each of size n, of some specified alphabet of elements, Σ. Winkler percolations use 0's and 1's, but any defined alphabet of characters is permitted. For every possible combination of $X[i]$ and $Y[j]$, $1 \leq i \leq n$, $1 \leq j \leq n$, within $\Sigma x \Sigma$, there must be an equivalent rule concerning the placement of edges on the percolation.

For example, suppose we build a percolation of size 2, whose strings X and Y are A's or B's. X = AB and Y = AB. The placement rules are set as follows. Whenever $X[i]$ and $Y[j]$ are the same, we want to draw an edge from that vertex to any vertex adjacent to it (above, below, to the left, and to the right). Whenever they are different, we will draw no new edges. Now, to determine our percolation, we look at all pairings of X and Y:

$X[1] = A, Y[1] = A \rightarrow$ connect $V(1,1)$ to $V(2,1)$ and $V(1,2)$

$X[1] = A, Y[2] = B \rightarrow$ do nothing

$X[2] = B, Y[1] = A \rightarrow$ do nothing

$X[2] = B, Y[2] = B \rightarrow$ connect $V(2,2)$ to $V(1,2)$ and $V(2,1)$

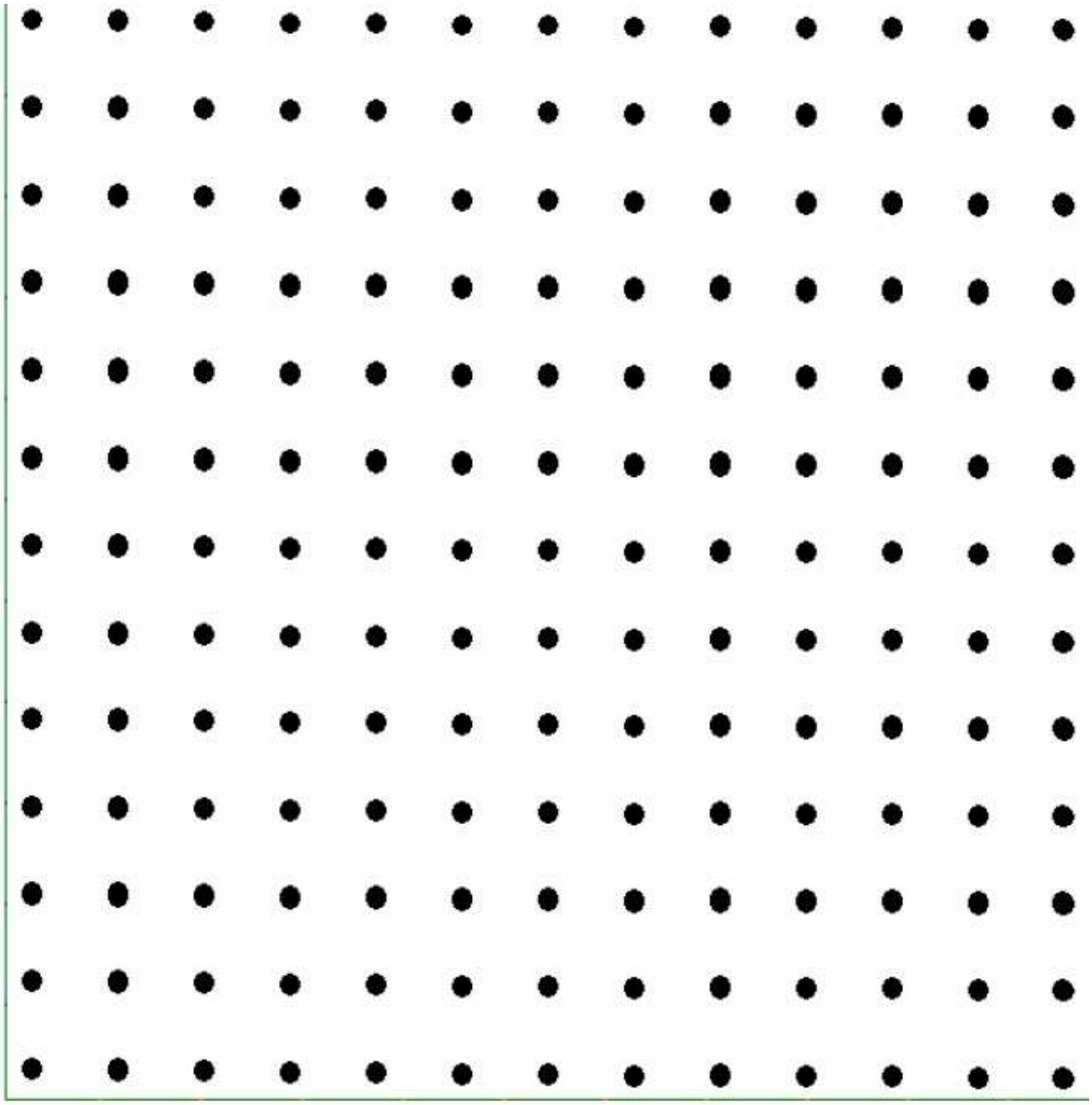

Figure 1: A Blank Generic Percolation

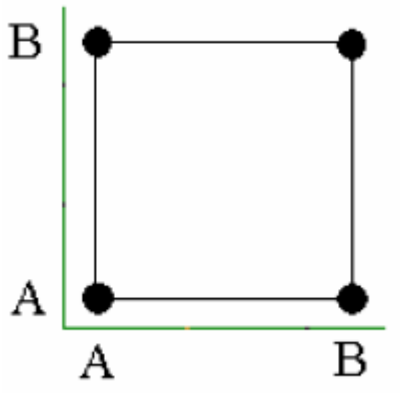

Figure 2: Percolation for $X = \text{AB}$, $Y = \text{AB}$

1.2 Random percolations

These strings X and Y can be pre-determined, or generated at random (choose a random element from Σ, for each element of X and Y). It is important to see the distinction between "random strings" (where every element's generation is unspecified), and these "independently and identically distributed" (i.i.d.) random strings. Every string is completely random because each and every element is randomly determined on an individual basis. Therefore, every pairing of every elements from these strings will occur completely randomly, and each combination is independent of all others (no element or pairing depends on elements created before or after it). It may seem to go against logic, but in this "completely random" scheme, we can make determinations, and predictions, instead of the "quasi-random" scheme where elements are not independent or based in some probability of frequency.

Also, the strings can be infinite in length, meaning that infinite percolations are possible. While both non-random and random strings can be infinite, we are only interested in those that we can make some determination about (we will have some means of predicting or knowing what values will be generated). For instance, using the example above, if $X = AAA...$ and $Y = AAA...$, we know our percolation will be an infinite lattice, with all vertices connected to all of their neighbors. If, however, we say X and Y are taken to be completely random, we cannot determine what is reachable from any vertex on the percolation, whether in a finite sampling of our graph, or at some point in the future (vertices very far from our origin, $V(0,0)$. The graph itself is random, and to observe a single case tells us nothing about the general case. Therefore, it would be best if our strings were not entirely randomized, but restricted by a probability. That is, for any $e \in \Sigma$, there is a probability p_e of occurrence. It goes without saying that $\Sigma_{\forall p_e}$ must be 1. This way, we are able to predict what our infinite strings will look like, and so we can make grounded determinations. Returning once again to our random A,B percolation, let us now set the probability of an A to 75%, and the occurrence of B is specified at 25%. Now, we can at the least know that as the lengths of X and Y get larger and larger, our graph will be very

connected, but still not as connected as when both strings are either all A's or all B's. When seeing "completely random" earlier, it may have been assumed by the reader that this meant all elements of Σ had equal opportunity to occur ($p_e = \frac{1}{||\Sigma||}$), but whatever the chances are, they must be directly specified, as we have shown here.

1.3 A percolation with a purpose

Now that the basics have been put forth, we may define these coordinate percolations in their terms. We will begin by taking the general purpose of this form of percolation, resolving half-duplex communication, and designing a percolation around it. To clarify, what we mean by half-duplex, is any media of transfer that can only send in one direction at once, such as a Citizens Band radio. We will use an alphabet $\Sigma = \{0, 1\}$, and the two strings X and Y will represent two people. A 1 means that the person wishes to send a message over the media, and a 0 means that the person is listening. Since messages cannot be sent by both parties simultaneous, we must avoid two ones from existing within the same locations. These strings will be random, and probabilities set for frequency of 1's and 0's (that is, how more likely these people are sending messages). From now on, we will call p_1 simply p (and consequently, $p_0 = p - 1$). The purpose of the percolation will be to prove whether or not, given a p, the two parties will successfully receive each other's messages. When this is possible, we say the strings are **compatible**. If it is not possible, the strings are **incompatible**.

We will now introduce the concept of deletion, an aspect of Winkler percolations that we will cover extensively later on within this paper, but for now, let us simply define it as a convenient "black box" utility. In general, we want to allow ourselves to remove elements in either string, if they will suit our needs (avoiding two 1's in the same position). Let us specify that the deletion of 1's is forbidden; it is important that every message be sent. Thus, we let 0's be removable from the strings, to represent telling one party to not wait for messages, at that specific point in time. When we delete, all elements in the string after the removed one shift

over one space, to fill this gap. This will create an empty space at the end of the string, and for now we define away this malady, and simply say this blank is irrelevant to our problem.

It may seem counterintuitive that we can prevent two 1's from occupying the same space by making one party "hurry up", but we will prove later that deleting a zero from one of these strings is equivalent to inserting a one into the other (telling the other person to talk more) which aesthetically sounds more productive. Either way, our percolation must now reflect the possibility of deletions.

1.4 Constructing a percolation allowing deletions

If we did not insert the ability to delete 1's, we would not need a percolation to solve this problem. All we would need to do is perform the following $O(n)$ pseudocode (whether by hand or machine):

```
for i = 1 to n
    if X[i] =Y[i] = 1
        FAIL
    end if
end for
SUCCESS
```

In other words, we move along both strings at the same time, and if there is ever any $1 \leq i \leq n$ such that $X[i] = 1$ and $Y[i] = 1$, we instantly fail our objective, since there is no way to shift our strings and try to move these 1's apart. To repeat, deletion is not as straightforward a process as it seems, and later we will demonstrate specifically how our scheme without deletions is **much** more restrictive than with, but again, for now, we simply observe that our problem is rather trivial without deletion, but percolation is **necessary** when including it.

Now, to construct our percolation that has built-in allowance of deletion. For now, we will assume our strings are finite, but extending our constructions onto infinity is not difficult at all. First of all, we restrict movement to either up, right, or diagonally up-right. Therefore, our percolation is a digraph. Second, in justifying this construction, it is best to think of a percolation in this case as moving across both strings simultaneously. For example, if we have an imaginary traveller on our graph, looking to reach the outer edges, and this traveller currently occupies $V(i, j)$, it is as if she were on the ith position of string X, and also on the jth position of string Y. Whenever the traveller moves forward on the graph, she is really moving forward on one of the X, Y strings, or both, whenever moving diagonally. Success means reaching the final element of either string, meaning that the two strings are compatible. Therefore, we now see one of the greatest built-in properties of coordinate percolation: they are meant to represent two separate forces (strings X and Y), who must work in tandem to achieve a common goal.

Third, as stated before, our edge placement have to be analogous to our problem specifications. We have forbidden the existence of two 1's in the same location, so let us define this condition as an absolute block within our graph (no arrows in any directions whatsoever). If there are two 0's in the same position, we can delete either of them without having any effect whatsoever on the problem, so let us employ the up and right arrow for this case. This represents our "traveller" having a choice; either moving forward on the X string, or the Y string.

The remaining two cases are more complex. If the vertical element is 0, and the horizontal is a 1, then we can either leave this pair untouched (move diagonally), or behave as if we had 'deleted' the one (which is basically true, since it has been paired with a zero), by moving upward. The same is done if the vertical element is 1 and the horizontal element is 0: we move either diagonally or to the right. To our traveller, this is the same as either moving forward on both strings at once, or choosing to move on only one. The latter choice is beneficial if the

traveller knows that not using this 0 immediately would be best; if there are more are on the way, this one should be reserved.

1.5 The traveller as clairvoyant

This “knowledge”, or more properly “foresight”, is built into the percolation, as will be seen. Once this percolation is completed, any means of search can be taken, from the exhaustive, or the less expensive or time-consuming reverse traversal. Either way, trails are sought, those going from the origin to the outer borders. If one or many exist, then the two strings **must** be compatible. If none exist whatsoever, then these strings are incompatible. Here is where that foresight comes back: if any path exists, the traveller will automatically do what it takes to reach there, simply by finding that trail. If no path exists, that means the traveller has tried all it can, but cannot complete her task.

It is not hard to see that no person could simply look at two strings of very large size, and be able to see what deletion schemes are needed to determine compatibility, much less tell you every possible manner in which this could be done (and there are often enormous ways, but the naked eye cannot discern them very quickly). However, using a percolation, this question is translated into finding a trail in a digraph. Our imaginary traveller is left with all the brute forcing, performed almost instantaneously when deciding on what move to make. For completeness, we will later demonstrate such a mannerism cannot exist.

Using this percolation model, we can answer the following question: “Given two parties, each with an random plan of communication, based upon some probability p of wishing to talk, and probability $1 - p$ of wishing to listen, will both parties be able to speak and ensure that the other side will be listening?” This is a lengthy question, but reducible to “Is it possible for two parties to utilize a half-duplex communications line?” Such will be possible only if a path exists from the origin to the goal, and it will be impossible if no path exists. Half-duplex lines

are not the only application of this percolation model; this can be utilized to solve the Dining Philosophers problem studied in distributed computing, and queue resolutions within operating systems.

The design we have built here is called a **Winkler percolation**, a schematic that was created by Peter Winkler, and written upon extensively by Peter Gács. Although the subject of percolations is not new, this particular form has not reached massive prominence as of yet; in fact, Gács is the only author of any formal work upon the subject. Before we study it in depth, let us leave this example behind and go through the terminology, both Gács's own, and that which we have devised in our study.

2 Terms and definitions

When we refer to "**a pair of 0-1 sequences**", this is two infinite strings, whose elements are either 0 or 1, chosen randomly based upon some probability p. Each element is independent and identically distributed, meaning there is a probability p that any element is a 1, and a probability $1-p$ that any element is a 0. This pair is called **compatible** if, there exists a method of deleting zeroes from either sequence, such that the resultant sequences will be complementary $(X[i] \wedge Y[i] = 0 \forall i)$. In contrast, a pair is incompatible if no such **deletion scheme** exists. If two ones are occupying the same position, they are said to be **colliding**, and this event is called a **collision**. We define the **k-prefix** of the pair as the first k elements of the infinite sequence. This is a finite sampling, which is quite useful to us in deciding whether the pair is compatible or not.

Theorem 1. *If the k-prefixes of two 0-1 sequences are incompatible, the entire sequence will be incompatible. ($k \in \mathbb{Z}^+$)*

Proof. This follows from the fact that two sequences will not suddenly become compatible if more information is provided about them. If no deletions exist that will align the sequences cor-

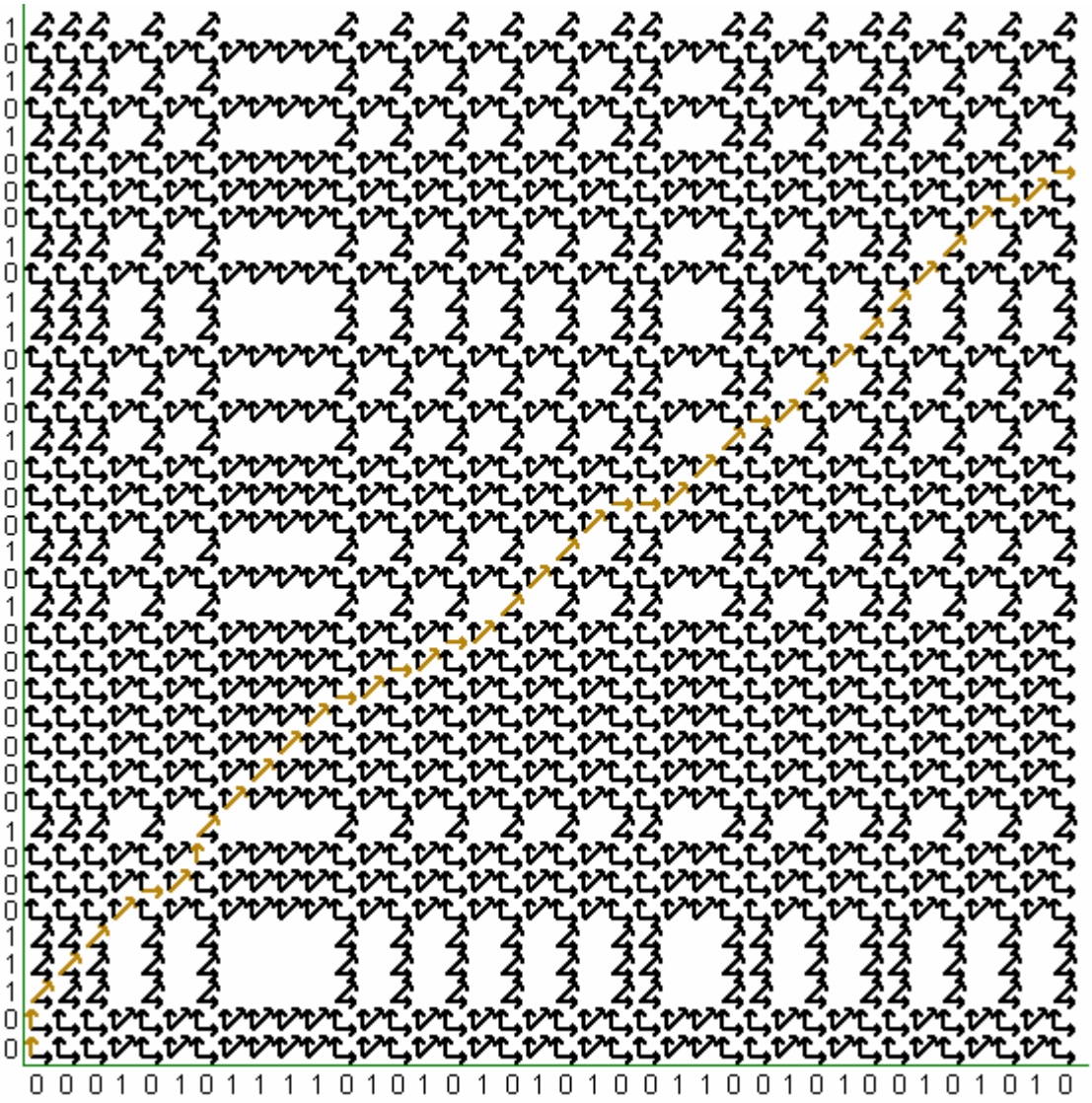

Figure 3: Compatible Winkler Percolation

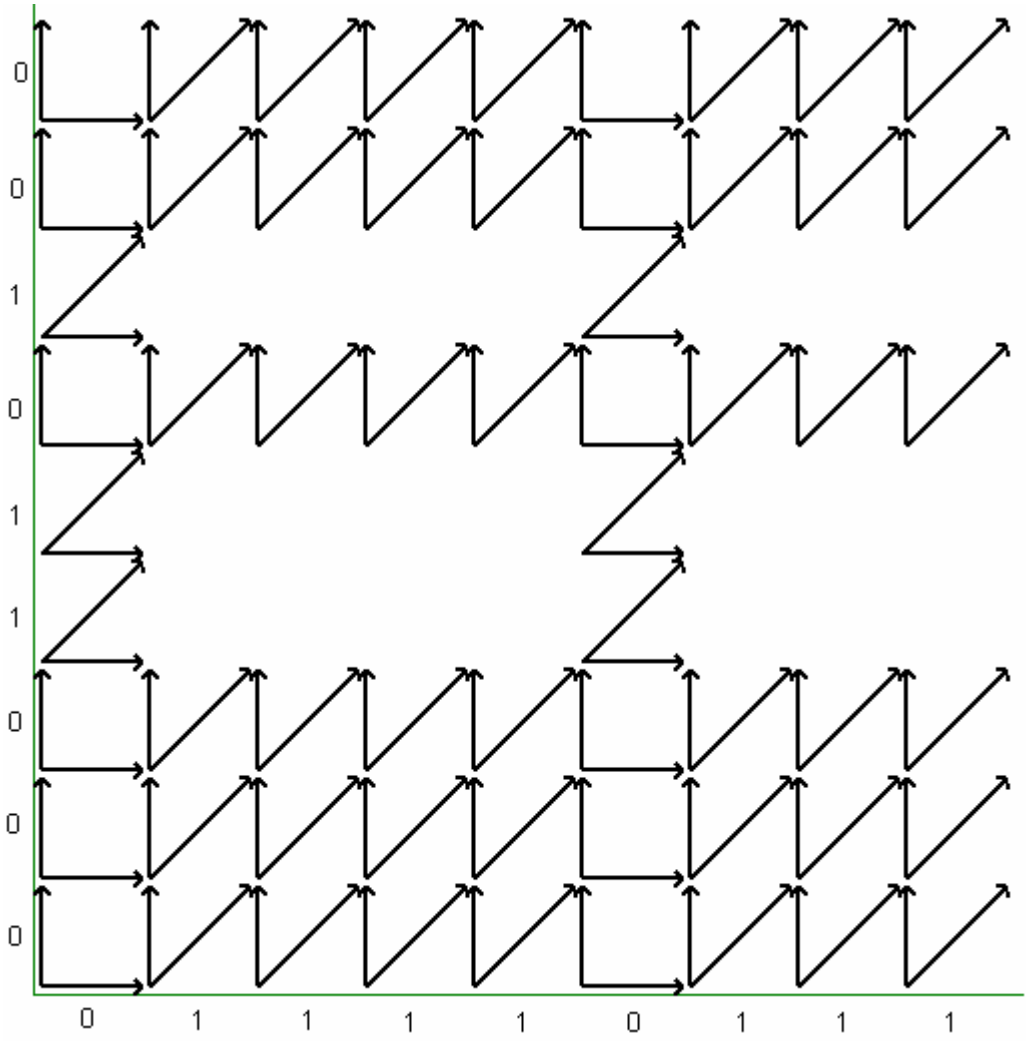

Figure 4: Incompatible Winkler Percolation

rectly, they will not exist after adding more digits to the strings. An incompatibility anywhere in the two strings makes the entire pair incompatible. □

Prefixes also have a feature that infinite sequences do not: a fixed ratio of ones to zeroes. We will call this the **actual probability** of a string (note that a pair of strings have an equal p, but they may have differing actual probabilities). Because of this, we can look at finite sequences, and try to see what probabilities make it so that compatibility is impossible. This will give us a near-bounds for p in the infinite case. At the very least, we should test a piece of intuition: that if there are more ones than zeroes ($p > 1/2$), the sequences are not very likely to be compatible. Winkler uses finite sequences in proving the following theorem:

Theorem 2. *The probability to ensure the compatibility of two finite 0-1 sequences must be less than 1/2.*

We will return to this theorem and prove it, based on Winkler's own technique, once we are armed with more information and background.

2.1 Precise meaning of compatibility

Why we need to be specific = compatibility = morphism of the problem at hand

2.1.1 Percolation without deletions

1.) Compatibility simply means that given any identical location in strings $X[i]$, $Y[i]$, that a 1 does not occur in both of these positions. This implies that two complementary strings are compatible (complementary in the binary digit sense, that given X, define Y to be the reverse of all elements. X and Y are compatible.

2.) Also, this means that given some string X, define Y again to be the complement of X, but also allow any elements of Y to be changed into 0's. The two strings X and Y are still complementary.

3.) A percolation must be constructed differently without deletions. The options our lattice allowed for are not as extensive, and our graph becomes rather simplistic:

If the elements are both 1, draw nothing.

Otherwise, draw a diagonal segment from the current node to that up and to the right ($E((i,i),(i+1,i+1))$).

4.) If two strings are compatible, there will be precisely **one** path from the origin (0,0) to the farthest corner, (*n*,*n*). If they are incompatible, there will be no paths.

We will later show the implications that are apparent here; no deletion schemes means no optional paths to take.

2.1.2 Percolation with deletions

1.) Gács defines this such that there exists a means of deleting 0's such that no 1's will occupy the same space in time, but we will get more extensive than this, only within this section (we will assume the word "compatibility" will entail all of these implications with future uses)

2.) For now, again, we imply deletion is well-defined and permissible. Also let us imply we are able to properly choose on our own what 0's can and cannot be deleted.

3.) Let us call a **legal deletion** one in which we remove a 0 from one string, and the resulting sequences are still compatible. To preserve the sizes of the strings, let us move this 0 to the end of the deleted string.

4.) We want to continue making legal deletions until we reach a point where our strings satisfy

the conditions described above for compatibility without deletions; two 1's do not occupy the same location within X and Y.

5.) This would imply that deletion is merely moving zeroes to the back of the string. It is important to stress we have not yet proven that this is precisely how we want to define deletion, but we will concede at this point that, assuming that **legal deletion** is properly defined (as it soon will be), this is equivalent.

3 Literature review - on Gács's work

What follows is a summation of "Compatible Sequences and a Slow Winkler Percolation", printed in *Combinatorics, Probability, and Computing*. We focus on the topics that will be of greatest use to us later on within our own study, and at least mention those either irrelevant or beyond the scope of giving a general yet thorough introduction. Gács has spent years researching this topic, and his dissertation is of the highest calibre of specificity and extensiveness. Taking many of his critical points and trying to condense them would only serve for our portrayal of percolations to be all the more distorted.

3.1 Introduction

Peter Gács formally defines coordinate percolations as graphs within his paper:

> *"We define a directed graph $G = (V,E)$ as follows. V = $\mathbb{Z}_+^2$ is the set of points (i,j) where i,j are nonnegative integers. When representing the set V of points (i; j) graphically, the right direction is the one of growing i, and the upward direction is the one of growing j. The set E of edges consists of all pairs of the form ((i,j); (i+ 1,j)); ((i,j); (i,j+1)) and ((i,j); (i+1,j+1)) ... In the chat interpretation, when X(i) = 1 then participant 0 wants to speak in the ith turn of his waking time, which is identified with the interval [i, i + 1). In this case, we erase all edges of the form*

((i, j), (i + 1, j)) for all j (this does not allow participant 1 to sleep through this interval). Similarly, when Y (i) = 1 then participant 1 wants to speak in the ith turn, and we erase all edges of the form ((j, i), (j, i + 1)). If X(i) = Y (j) then we also erase edge ((i, j), (i + 1, j + 1)). For X(i) = 1 since we do not allow the two participants to speak simultaneously, and for X(i) = 0 since the edge is not needed anyway, and this will allow a nicer mathematical description. This defines a graph G(X, Y) [3]."

3.2 Using renormalization in proof

The goal of this section is to prove the following: "If (0,0) is not blocked in any finite square, then, by compactness, or König's Lemma, there is an infinite path in the given Winkler percolation starting at (0,0)" (or equivalently, the two strings used to create this percolation are compatible).

König's Lemma: A tree with a finite number of branches at each fork and with a finite number of leaves at the end of each branch is called a finitely branching tree. König's lemma states that a finitely branching tree is infinite iff. it has an infinite path. [6]

Thinking of a Winkler percolation as a tree (each point is a node with zero, one, or two children), we show first that it is a finitely branching tree (trivial), and then that it is infinite (no blocks from $(0,0)$ to infinity), and this will prove there exists an infinite path.

The technique Peter Gács uses for this is renormalization.

renormalization: The idea of renormalization is that, while some continuous physical systems are by necessity described by models with a characteristic smallest length scale (or largest energy scale), the large-scale physical predictions of the theory should not depend on that

characteristic length scale.

This is a simplified definition of a much grander concept, used in statistical mechanics, implementing, as Gács terms it, "messy, laborious, and rather crude" methods of usage. Even the most general of technical definitions had to resort to "automorphisms" (isomorphisms unto itself), "isometrics" (distance bijection preservation), and "Hopf algebras". The general usage (at least, what Gács intends to do) is define a series of sets whose truth holds when certain "bad events" occur, and show that the probability of all of those groups occurring at once is less than one, therefore the percolation/tree is infinite, and there exists an infinite path. The details and sketch of the proof are reworded below:

1.) First, we define a sequence of Δ_k such that $\Delta_1 < \Delta_2 < \Delta_3 < ...$, with the constraint $\Delta_{k+1} > 4\Delta_k$. The reason this constant 4 has been chosen is revealed later in the paper (page 10, corresponds to eventual lookahead calculations).
2.) Using these Δ_k, we define $\mathcal{F}_k$ to be the event that the point $(0,0)$ is blocked in the square that is defined as $[0,\Delta_k]^2$.
3.) The idea is to prove that Prob($\bigcup_k \mathcal{F}_k$) ¡ 1, which proves our finitely branching tree is infinite = not blocked within a finite square, for all possible k (so, any squares that we could make, given these Δ_k) 4.) To do this involves the renormalization groups. First, label some events as being "bad" and other, less common ones "very bad". Later, we will show these "bad" events are the existence of walls, and "very bad" are emerging and compound walls.

emerging wall: a large interval within the percolation without meeting a hole that matches up with a particular wall.
compound wall: two or more walls that are 'very' close to one another, so that they operate combined as an entirely different wall.

5.) Using these events, we define 'models' $\mathcal{M}^1, \mathcal{M}^2, \mathcal{M}^3, ...$, similar to each other, in that the "very bad" events of $\mathcal{M}^k$ are the bad events of $\mathcal{M}^{k+1}$. These will become our "mazeries" eventually; these are not only linked to Δ_k, but also $\mathcal{F}_k$.
6.) Now, let $\mathcal{F}_k'$ hold iff some bad event $\mathcal{M}^k$ happens in $[0, \Delta_{k+1}]$. Note the similarity to $\mathcal{F}_k$: one is bounded in Δ_k, the other in Δ_{k+1}. But, it must be mentioned here that $\mathcal{F}_k$ is considered the "ultimate bad event", but $\mathcal{F}_k'$ is merely a model $\mathcal{M}^k$ with only a "bad event", therefore, its already clear that it will take several $\mathcal{F}_k'$ to be as destructive as only one $\mathcal{F}_k$.
7.) Prove that $\mathcal{F}_k \subset \bigcup_{i \leq k} \mathcal{F}_i'$

Here is the true meat and potatoes of this entire proof: we must now use the properties of these various building blocks to show that, a combination of sufficient amounts of these constructed $\mathcal{F}_k'$'s (the big union) would have to include this major disaster: $\mathcal{F}_k$.

($\mathcal{M}^*$ is now set equivalent to $\mathcal{M}^{k+1}$, $\mathcal{M} = \mathcal{M}^k$
We now call these models $\mathcal{M}$ "mazeries", and look at this problem from the split point of view.

mazery: "a system for creating mazes", a partition of the percolation, with its corner starting at the origin and extending to a square of some defined Δ_k, a property later associated directly with a given mazery (as opposed to this "incursive" creation of mazeries that we are conducting currently, i.e. building one on top of another).

split view: Term used to refer to whenever we look at the two linear mazeries $\mathcal{M}_0$ and $\mathcal{M}_1$, instead of one huge square percolation $\mathcal{M}$. Much easier to see that this is not a problem concerning one structure, but two that must work together somehow.

Any given mazery will have inside of itself some mazery $\mathcal{M}^*$, which corresponds to the typical unlucky events of that mazery (walls, barriers). This is used in transitions: if there is a pair of

close walls in $\mathcal{M}$, then in $\mathcal{M}^*$ there will be a compound wall. The same goes for an interval without a particular hole type in $\mathcal{M}$; in $\mathcal{M}^*$ there will be an emerging wall (both defined as such above).

The reason for this scale-up construction is to hide certain components of $\mathcal{M}$ (not eliminate, but create a system where they will resolve themselves), and to bring attention to two restrictions that will carry over: clean points and slope constraints.

clean point: a point that can be left (departed from). There is way much more to it than this (this is defined in the proximity of walls and barriers to the point, but for now, the idea is that from a clean point, traversal is very easy) Here, its simply defined that a point is clean for $\mathcal{M}_0$ X $\mathcal{M}_1$ if x_d is clean for $\mathcal{M}_d$ $(d = 0,1)$ (if can be departed from in either split mazery).

slope constraint: for some defined constant σ, such that $0 \leq \sigma < 1/2$, two points (x_0, x_1) and (y_0, y_1) satisfy this constraint if the following holds true:

$$\sigma \leq \frac{y_1 - x_1}{y_0 - x_0} \leq 1/\sigma$$

Let Q be the event that the point $(0,0)$ is not clean in $\mathcal{M}_0$ or $\mathcal{M}_1$ (there exists, in a close enough proximity, a wall of some sort (a bad event). "We want to say that in a mazery, if points (x_0, x_1) and (y_0, y_1) are such that for $d = 0,1$ we have $x_d < y_d$ and there are no walls between x_d and y_d, then $(x_0, x_1) \rightsquigarrow (y_0, y_1)$." [3] But theres the slope-constraint; you can only go so far so fast, even with the most optimal of conditions.

(Lemma 2.3) We have $\mathcal{F} \subset \mathcal{F}' \cup Q$. The proof is restated below.

Define a sequence of mazeries $\mathcal{M}^1, \mathcal{M}^2, \ldots$ with $\mathcal{M}^{k+1} = (\mathcal{M}^k)^*$, with $\Delta_k \to \infty$. All these are on common probability space, since $\mathcal{M}^{k+1}$ is a function of $\mathcal{M}^k$ (a reaffirmation for some of what were about to do). The event Q_k, that $(0,0)$ is not upper-right clean in $\mathcal{M}^k$ plays the role of Q in the mazery $\mathcal{M}^k$.

Put this together to find $Q_k \subset \bigcup_{i<k} \mathcal{F}_i'$, and this proves that $\mathcal{F}_k \subset \bigcup_{i<k} \mathcal{F}_i'$.

8.) Prove $\sum_k Prob(\mathcal{F}_k') < 1$.

This is the final step, as Prob$(\bigcup_k \mathcal{F}_k) < \sum_k Prob(\mathcal{F}_k')$, so if $a < b$ and $b < 1$, then $a < 1$, which is what we are ultimately looking for (where $a = Prob(\bigcup_k \mathcal{F}_k)$ and $b = \sum_k Prob(\mathcal{F}_k')$. This step is proven in (Lemma 2.5 Main).

Lemma 2.3: Suppose Q does not hold, then $(0,0)$ would be clean. Suppose that $\mathcal{F}'$ does not hold (there is no bad event of $\mathcal{M}^k$ happening in $[0, \Delta_{k+1}]$), then, since every interval of size 3Δ that does not intersect walls contains a clean point in its middle third, there is some point $x = (x_0, x_1)$ with $x_d \in [\Delta, 2\Delta]$ clean in $\mathcal{M}_d$, for $d = 0, 1$. With $\sigma = 2$, this point satisfies the slope-constraint $1/2 \leq x1/x0 \leq 2$. Now we have a clean point, satisfying the slope constraint, reachable from the origin (the complement of $\mathcal{F}$). If $\overline{\mathcal{F}} \subset \overline{\mathcal{F}'} \cap \overline{Q}$, then $\mathcal{F} \subset \mathcal{F}' \cup \mathcal{F}$ (DeMorgan's Law). $\square$

3.3 Walls, holes, and barriers

Peter Gács defines walls as blocks of 1's, holes as blocks of 0's, and barriers as particular combinations of walls, such as the emerging or compound kind; they are walls that are harder to get around. In this context, it is first observed that it is much harder to resolve 1's that are close together in proximity, such that only finding particular blocks of 0's are the only means of success. Compatibility is not just a matter of hoping that two strings are nearly compatible, but

also, whenever a string pair is full of inconsistencies, being able to apply the proper ‘holes’ to the correct ‘wall’. Gács goes into great detail on this matter, defining our previously mentioned mazeries as structures in terms of the types of walls, holes, and barriers contained within, and using this as the ‘quantitative’ measurement of a mazery; those with a variety of holes will be more sought after than those with few. These calculations are the groundwork for being able to find necessary ‘lookahead’: if mazery characteristics can be predicted, then its resolution can be determined by seeing which walls it contains, and then comparing this to the holes that are available for usage.

4 Other relevant works

4.1 On playing golf with two balls

In 2003, Peter Winkler collaborated with Ioana Dumitriu and Prasad Tetali on a different model that uses two ‘individuals’ working together to reach a particular goal, this time involving Markov systems and game theory. The former is implicated within the goal: to choose a Markov chain that will reach a target state faster than some competing force. The latter is the reminder that staying “loyal” to one particular entity is not always the best game plan; it might become necessary to change from one side to another in order to win [2]. This is representative of a situation within a Winkler percolation in which we cannot hope to move away from the origin in equal directions (up and right), but may come across a situation where it is most beneficial to let a person talk for a very long time, and then let the other person “catch up”. In the best case, both people will talk in equal amounts, but it is not absolutely necessary. While percolating is not a race (there’s no competition), the *resolution* of percolations can be thought of as set against time; we want our deletion scheme to be established in finite steps.

4.2 Percolation beyond $\mathbb{Z}^d$, many questions and a few answers

This paper is one of the earliest works that challenged the traditional thought on general percolations, and has been referenced by many when dealing with irregular percolations. Like Gács's dissertation, the tools that are capable of percolation analysis are defined and justified. For instance, the "clusters" and subgraphs referred to by Benjamini, Lyons, and Schramm are very similar to Gács's "mazeries" [1]. However, unlike Peter Gács, some conjectures and problems are left to the readers, to push this topic further and inspire critical thought. Time has proven that this approach has succeeded: many additional studies were based upon the challenges first raised in "Percolation Beyond $\mathbb{Z}^d$". We hope that "On Compatible Sequences and A Slow Winkler Percolation" is also a catalyst of such curiosity.

5 Our objective - an exhaustive search

As mentioned earlier, a great deal of footwork remains to be done inside of the realm of Winkler percolations, of the sort that exists for many objects similar in design and complexity. When talking about the finite percolation, we can think of this as a changeable object, with a determinate amount of possible states (combinations of X and Y, and their possible elements). Thus, it is possible to enumerate every possible state of a percolation. The best scheme to use would be one based upon something we already know how to enumerate: binary digits. If we think of these 0-1 sequences as boolean numbers, then our ranking would be based on using X and Y to "count" from 0 to 2^n. Since we have two strings, we need to work with both at once, so the following ranking manner is proposed:

1.) Start both X and Y at 0^n. This is the first rank.

2.) Keep "increasing" Y as if it were a binary digit, but leave X alone (at this step).
Each successive increase is a new ranking.

3.) After a step where $Y = 1^n$, increase X by one, as if it were a binary digit. Reset Y to 0^n.

4.) Continue in this manner until rank 2^{2n}, which will be $X = (1^n)$, $Y = (1^n)$.

Whenever we list possibilities for a finite percolation of size n, we will use this ranking method. Now that we have a hierarchy for our objects, we can list them, and see if there are any unifying characteristics, based on certain specifications. By "specifications", we refer to size n, how many 1's X contains, how many 1's Y contains, whether or not deletions are allowed, and so forth. As we said before, deletion requires a great deal of clarification before employing it, so we now take the time to explain what this ability translates to, and how much it complicates our search.

6 On the concept of deletion

Earlier, when defining Winkler percolations, we wished to avoid introducing deletion without a proper backing-up of *why* we are able to perform it, but it might not be obvious why such great care must be taken with this facet of percolation. We said that a "deletion" would force all other elements of the string to shift to the right, to fill in the newly free space. However, what does this mean to our percolation? Assuming we begin with two strings of size n, and we delete an element from one of them, we now have a string of size n, being compared to a string sized $n-1$. How does such a comparison make sense? Our percolation scheme was based on our two parties X and Y performing one of two activities at each "time interval" i (That is, for $X[i]$ and $Y[i]$, both X and Y are either talking or listening, at the same moment. If X has performed all of its actions, and Y has more, how do we resolve those actions? Do we give Y a "free ride" at this point? Fundamentally, the answer to this question should be yes: we do not want to delete from strings, if it will instantaneously make them incompatible. The purpose of deletion should be to seek compatibility, not destroy it. However, just saying this is how our situation **should** be does not make it necessarily true; we should look in-depth and be sure that deleting 0's makes sense logistically.

6.1 A sub-percolation - making non-square lattices

One possible solution to our problem is to introduce the concept of a non-square percolation, one that is *m*x*n* in size, where *m* is not necessarily equal to *n*. This is akin to omitting rows and columns in our original percolation, almost always having our aforementioned imaginary traveller "skip over" that value in the X or Y string, therefore this "free jump" is expressed by truncating our lattice graph. It is obvious that the two percolations are not entirely equal (there dimensions will be different), but now we must prove if they are fundamentally equivalent. Let G' be a percolation with a deletion of a 0 from percolation G. We want to be able to determine facts similar to the following, in order for our transition to work:

1.) The X' and Y' constructing G' are compatible iff. the X and Y used to construct G are compatible.
2.) The X' and Y' constructing G' are incompatible iff. the X and Y used to construct G are incompatible.

For totality, we prove both statements two ways, by looking not only at the newly-scaled graphs, but also the two strings directly. In the latter case, we will be unable to avoid the matter of comparing two strings of unequal size; we will be forced to define away this problem, but once again not without giving warranted justification. We will address this now outside of the proof proper.

The easiest solution would seem to be disregarding any empty spaces that exist within either string. That is, fundamentally, we accept the existence of rectangular percolations. Another approach is, since 0 is a non-colliding element, in order to preserve sizes, we could append that to the end of the string with the deleted element.

One final critical point needs to be addressed: how do we know what can and can not be

deleted? There exist situations where permanently deleting a 0 would cause compatible strings to be incompatible. We want to say "just do not delete these", but we still have not determined how to avoid these situations. For now, we will presume that these 0's are detectable, but this lack of precognition will eventually be our greatest burden in defining deletion. In the theorem below, the underlined proviso entails this foresight. Even though this is not what we want at the moment, we will go through this theorem with our current knowledge base to demonstrate how this "need to know" muddles everything up.

Theorem 3. *Let G be a Winkler percolation of size (n) by (n), constructed by two strings, X and Y. Without loss of generality, delete a 0 from the first string, X, to create a new string X', that will not alter the compatibility status of the two strings. Set $Y' = Y$. Construct a new Winkler percolation of size $(n-1)$ by (n) with X' and Y'. If at least one trail existed in G, then at least one will exist in G'. The converse holds as well; if no trails existed in G, then none will exist in G'.*

Proof. Our new provision makes our proof almost trivial. If X and Y were originally compatible, and we wish to remove a 0 without changing that, then X' and Y' will be compatible also. Compatibility implies a trail will exist within the percolation, therefore both G and G' will contain at least one. □

If the strings were incompatible, by definition, no deletion of 0's will change that. G will contain no complete paths, and neither will G'.

6.2 Another approach: what does the traveller know?

Our current perspective on deletion is lacking something, if we want to continue on to have an airtight concept of what is deleting, and also what can be deleted. What we have tried to do previously is omit sections of the percolation, in order to reduce the problem size, but keep equivalence. We will now show in this section that is only possible under very particular circumstances. To do this, we need to prove that reducing a graph is exactly equivalent to the

event in which our imaginary traveller will **always** choose to delete the 0. Without loss of generality, assume we are dealing with deletion in the X string (since percolations are symmetrical, we could be performing the same steps in the Y string, but when we refer to columns, instead replace with 'rows'). The situation we wish to study is labelled Figure 5.a. We want to delete the circled line, without altering the compatibility of the two strings. The only way we can know this will work is if we construct G' out of X' and Y'. If G' has no trails, we cannot always delete this element. Figure 5.b is G'; unlike G, it has no complete trails. However, if G' does have trails, as in Figures 5.c and 5.d, then this element had no effect on the compatibility of our percolation, and can be removed without altering our problem. At long last, we have a reduction for Winkler percolations, but one only found using an expensive trial-and-error method. Unfortunately, since we can never know as much as our Traveller, this is the only way we will find such elements. Though, it is important to observe this algorithm (design a percolation, and see what can be removed) is a lot easier than the brute force approach: deleting elements, and testing their compatibility.

6.3 Re-approaching Theorem 3 - the clairvoyant's discretion

We have demonstrated that if we leave the daunting task of determining which 0's are irrelevant to the problem up to the human users of coordinate percolation, it is very easy to end up in trouble. Deletion by choice, and when made permanently, can only be checked *a posteriori*; there exists no methodology of determining what zeroes must go. This makes sense fundamentally, as the existence of such a rhetoric would render Winkler percolations obsolete. Compatibility could be left to this greater system, one that perhaps could determine precisely which 0's will always be irrelevant to the problem, thus reducing the problem's size and complexity. However, it is fortunate we have not so easily determined our subject matter to be easily replaced.

Therefore, as earlier suspected, it is up to our "Traveller" to ultimately make our decisions for us. If a particular element fits our previous qualification (that removing it does not change

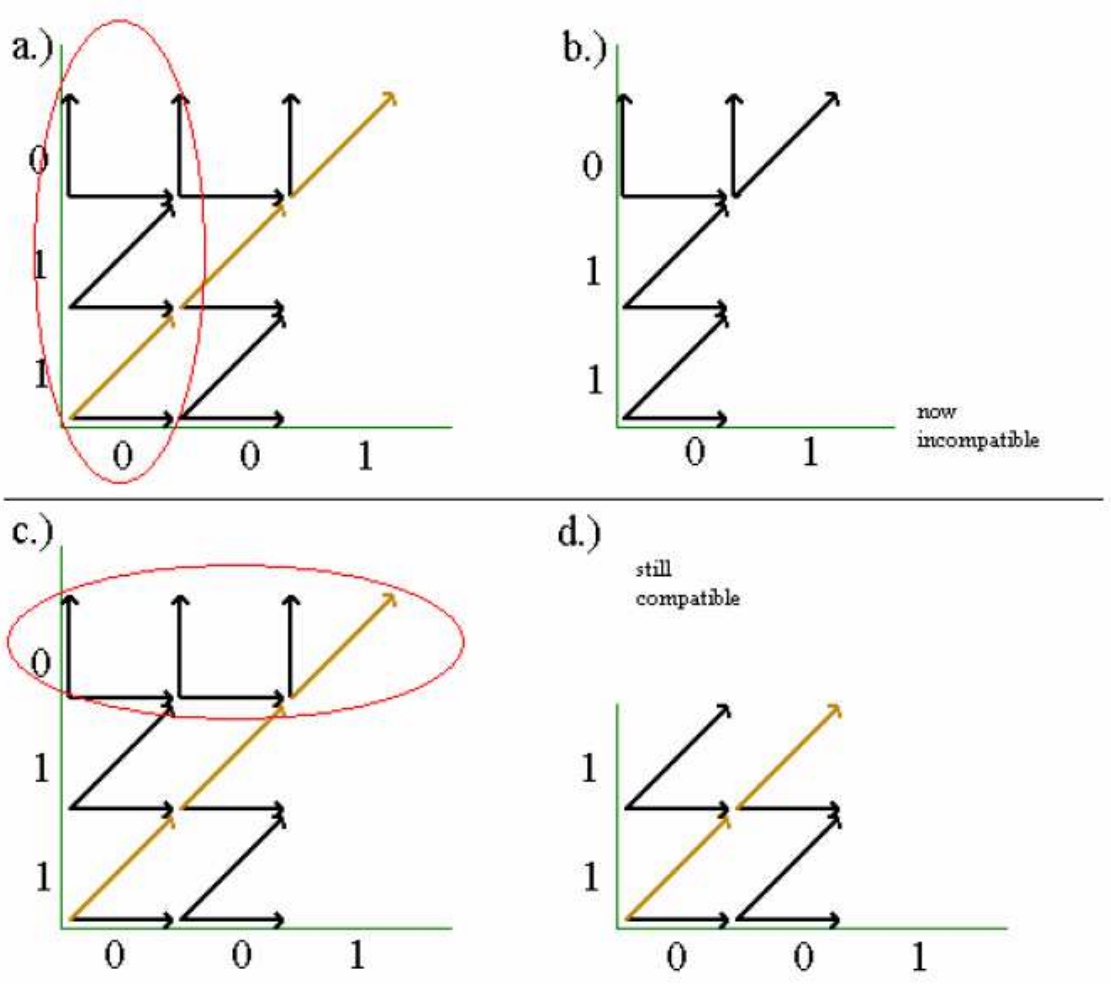

Figure 5: Examples of Deletion

our problem), then our Traveller can choose either to leave it or delete it. This is deletion at large, a device best left to an inherent component of the percolation. If we ourselves want to duplicate this feat, we must be prepared for a great amount of work to reach success. We do not want to let either the Traveller or ourselves delete an element that will create incompatibility, so let us now designate any such removal as a **legal deletion**. In larger percolations, the benefits of "manual deletion" may outweigh the costs. For example, eventually we wish to consider the infinite case, and a currently open-ended question is that of "lookahead"; how much of the infinite percolation must be seen in order to assure compatibility with positive probability? If, inside of our finite samplings, there exist elements that can be removed, the size of the problem is reduced, and, depending on how large this sampling is, the time needed for the traversal of this massive percolation may shrink drastically.

Let us now rephrase and reprove Theorem 3:

Theorem 4. *(Revision of Theorem 3) Let G be a Winkler percolation of size (n) by (n), constructed by two strings, X and Y. Without loss of generality, make a legal deletion a 0 from the first string, X, to create a new string X'. Set $Y' = Y$. Construct a new Winkler percolation of size $(n-1)$ by (n) with X' and Y'. If at least one trail existed in G, than at least one will exist in G'.*

Proof. If X and Y were originally compatible, and a legal deletion was made, then X' and Y' will be compatible also. Compatibility implies a trail will exist within the percolation, therefore both G and G' will both contain at least one. □

Notice we do not state the converse here, because the hypothesis of the proof implies a legal deletion was made. Right now, that is something we can only conduct to preserve compatibility. It would not make sense to delete anything from an incompatible percolation, since no alterations will change its status. Therefore, once we determine incompatibility, there is no reason to want to reduce the problem. One of the purposes of deletion is to reduce the number

of trails that exist in the percolation. Why bother condensing a percolation that has no trails whatsoever?

When all we have seen up to now is considered, Theorem 4 becomes very trivial (because a legal deletion could be made, of course G and G' are compatible), but it was definitely worth "going through the motions", and making sure logic followed suit.

6.4 Conclusion - why can we delete?

The purpose of an in-depth look into deletion was to justify a means of simplification that will come to light later, when we look at the total number of paths that a percolation contains, and this count of paths is proportional to the potential deletion schemes (we will show this later on). Eventually, we will put these two simple facts together, and enrich the existing details that Gács has suggested: that percolations have aspects of "self-similarity", and thus can be re-normalized in the manner summarized above. Showing this fact, and then using it to do a combinatorial analysis of finite percolations, is the absolute goal of this work. It is easy to see that finite percolations are extendible: just add more elements the the end, or take a bigger sampling of the infinite case. Here, we have laid to rest the suspicion that *reducing* these percolations is not entirely easy, but possible, thanks mostly to finiteness. In the end, we can delete **because** we can add. Done 'responsibly', we can use either tool and find "congruent" percolations (G has same compatibility as G').

Armed with new cautions and new techniques, we begin our investigation of the finite examples.

7 Finite cases - using actual probabilities

7.1 Graph setup

The graph that we use to view our pairs of compatible sequences should be set up as follows. The x string is set on the x-axis with each digit representing one unit. The same set-up is used on the y-axis with the y string.

The graph is then drawn as follows. Whenever there is a 0 in both positions on each string, a vertical and a horizontal line are drawn in that spot on the graph. If there is a 0 in the x string and a 1 in the y string, then a horizontal line and a diagonal, bottom-left to top-right line is inserted. If there is a 1 in the x string and a 0 in the y string, then the bottom-left to top-right diagonal line is combined with a vertical line to fill this space. Finally, if there contains a 1 in both positions, that spot is simply left unfilled. This is also a directed graph of sorts, so there should be arrows on all these lines. The graph is directed up and to the right, so the arrows should be on the top and right of the lines.

7.2 Interpreting the graph

Now the graph should look like a connecting series of lines. If the two sequences are compatible then it is possible to get to either the top or the rightmost part of the graph without picking up your pen and without going left or down. If a line can be drawn from the origin straight through to the point (n,n), that means that these two sequences are complimentary. If there is no possible way to traverse the graph, then the sequence is a clashing, non-compatible pair.

When the graph is being traversed, a move in the horizontal direction means that the 0 in the x sequence has been deleted and as such, the next number in the x sequence is compared to the number in the y sequence that was previously compared. Similarly a move in the vertical

direction indicates that a 0 in the y sequence has been deleted. A diagonal move indicates that the 1 in one of the sequences as well as the 0 in the other sequence has been compared and deemed fine to move on, so both sequences advance one term. This explains why only the farthest right edge, or the topmost edge of the graph needs to be reached to ensure that these sequences are complimentary.

1 X 1

The 1 x 1 case is simple enough that it will be discussed in this paragraph just to give a brief starting point to our work. When you have two strings of length one, there are only two choices for each sequence. Either a 1 or a 0. So for every x string, there are two y strings. This means that there are a total of four sequence comparisons.

x = 0, y = 0 – This is the smallest example of a pair of compatible sequences.

x = 0, y = 1 – This is the smallest example of a complimentary sequence. This is a pair of compatible sequences that in any given position of the two strings, one of the strings contains a 0 and the other contains a 1.

x = 1, y = 0 – This may look a lot like the previous example, but it is in fact not the same. This example is obtained by applying a coordinate change to the previous example. The same analysis applies.

x = 1, y = 1 – This is the smallest clashing pair. This pair of sequences is not compatible. It also contains no 0's, so there is no way to obtain compatibility through removal of 0's. **Summary**

Comparisons = 4

Complimentary pairs = 2 – Complimentary pairs can be matched up by coordinate change, since every position in the pair of sequences contains a 0 in one string and a 1 in the other. So there will always be an even number of these. In this case it is x = 1, y = 0 and x = 0, y = 1.

Compatible pairs (without deletions) = 3

Compatible pairs (with deletions) = 3

2 x 2

The 2 x 2 case there are 4 possibilities for each string. 00, 01, 10, or 11. Each of these paired with every possible combination makes 16 possible string pairs. Here it should be interesting to note some growing patterns. These patterns will be able to come into play later as we will notice that any string we can come up with in the future will begin with one of these 16 different combinations.

A. x = 00, y = 00 – This is one of those basic cases. There are 16 different ways to traverse the grid. There is nothing that we have to worry about as these two strings are compatible without any deletions.

B. x = 00, y = 01 – This is almost the same as the previous case. There are still 6 ways to cross the grid path. This is because the 1 occurs at the end of the string, and since the other string is all 0's there is nothing we have to worry about in terms of crossing a wall.

C. x = 00, y = 10 – This is a bit more complicated than the previous case. There is only 4 ways to cross this grid path. Since the 1 occurs in the first position of the y string, there is

no way we can move directly up and reach the leftmost exit of this small graph. Since there is only a single 1 in this pair of sequences, it is easy to spot all the paths that traverse this graph.

D. x = 00, y = 11 – This is another one of those complimentary string pairs. What this means is that it is possible to go from the origin to the point (2,2) which is the upper right hand corner of the graph. This signifies being able to traverse through the graph without deleting any 0's. But since there are 0's and these strings are very small, there are actually 4 ways to cross this grid.

E. x = 01, y = 00 – This example is obtained by applying a coordinate change to B. The same analysis applies.

F. x = 01, y = 01 – This is the first 2 x 2 case where these two strings are not compatible, but through clever deletions of 0's it is possible that these strings can be made compatible. Depending on which 0 you choose to delete it leads you towards one of the only 2 exits of this graph. The uppermost exit can be reached by deleting the 0 in the y string, and likewise the rightmost exit can be reached by deleting the 0 in the x string.

G. x = 01, y = 10 – This is another complimentary pair of strings. This example also has the property that each string has exactly half 1's and half 0's. What this means is that there is only 2 ways to traverse this graph. The first by taking each digit as they come and ending up again at the point (2,2). The other way would be to skip over the 0 in the y string and exit that way.

H. x = 01, y = 11 – This is an incompatible string. There are no deletions you can do to make these strings compatible, and the graph traversable. There are 3/4 ones ... and this fraction is more than half, which would make them incompatible.

I. x = 10, y = 00 – This example is obtained by applying a coordinate change to C. The same analysis applies.

J. x = 10, y = 01 – This example is obtained by applying a coordinate change to G. The same analysis applies.

K. x = 10, y = 10 – This is a pair of incompatible strings. If you notice this starts with a 1 in each position, and as mentioned in the 1 x 1 section, anything that starts like this is destined to be incompatible.

L. x = 10, y = 11 – This is another one of those cases where as we can see there are more than half 1's and thus, no compatibility.

M. x = 11, y = 00 – This example is obtained by applying a coordinate change to D. The same analysis applies.

N. x = 11, y = 01 – This example is obtained by applying a coordinate change to H. The same analysis applies.

O. x = 11, y= 10 – This example is obtained by applying a coordinate change to K. The same analysis applies.

P. x = 11, y = 11 – At this point, when there is not a 0 to be found in any of these strings, then you can stop looking for compatibility and trying to traverse an empty graph.

Summary

Comparisons = 16

Complimentary pairs = 4

Compatible (without deletions) = 9

Compatible (with deletions) = 10 – The sequence that is compatible with deletions only is x = 01, y = 01.

7.3 Banning deletions - a limited scheme

7.3.1 Computational Back-Up

What follows are the results of a computer-generated count of the finite examples, from $n = 2ton = 12$. First, the numbers of successes per *n*, arranged in increasing order of 1's contained within the first string X. The path count average is useless for this case study; every compatible percolation will have a single path, therefore the average paths for any given X string with q 1's would be (number of successes of q)/$\binom{n}{q}$ 2^n, since $\binom{n}{q}$ is the total number of pairings any given X string will be placed with, and each of those will be compared to all other possible strings of size n, 2^n.

For example, for n = 4, there are n+1 = 5 separate entries: 0 through 4. There is one string of size 4 with 0 1's: 0000. It is compared to all other strings of size n, and it is compatible with all of these. The entry for n = 4, q = 0 is 2^n. Next, is X strings with a single 1: 0001, 0010, 0100, and 1000. Each of these are paired with all other strings of size n, yielding $4 * 2^n$ different pairings. Of these, only one in each group will be incompatible: 0001 v. 1111, 0010 v. 1111, 0100 v. 1111, and 1000 v. 1111. The entry for n = 4, q = 1 will be $4 * 2^n - 4$. This continues all the way to n = 12.

Although all results are reported here, the numbers we are most interested in are for $q = \lfloor n/2 \rfloor$, which have an asterisk in these tables.

n	q	successes (s)	number of comparisons $(2^n * \binom{n}{q}) = d$	percent successes $(s/d)100\%$
2	0	4	4	100%
*2	1	4	8	50%
2	2	1	4	25%
3	0	8	8	100%
*3	1	12	24	50%
3	2	6	24	25%
3	3	1	8	12.5%
4	0	16	16	100%
4	1	32	64	50%
*4	2	24	96	25%
4	3	8	64	12.5%
4	4	1	16	6.25%
5	0	32	32	100%
5	1	80	160	50%
*5	2	80	320	25%
5	3	40	320	12.5%
5	4	10	160	6.25%
5	5	1	32	3.125%
6	0	64	64	100%
6	1	192	384	50%
6	2	240	960	25%
*6	3	160	1280	12.5%
6	4	60	960	6.25%
6	5	12	384	3.125%
6	6	1	64	1.563%
7	0	128	128	100%
7	1	448	896	50%
7	2	672	2688	25%
*7	3	560	4480	12.5%
7	4	280	4480	6.25%
7	5	84	2688	3.125%
7	6	14	896	1.563%
7	7	1	128	0.781%
8	0	256	256	100%
8	1	1024	2048	50%
8	2	1792	7168	25%
8	3	1792	14336	12.5%
*8	4	1120	17920	6.25%
8	5	448	14336	3.125%
8	6	112	7168	1.563%
8	7	16	2048	0.781%
8	8	1	256	0.391%

n	q	successes (s)	number of comparisons $(2^n * \binom{n}{q}) = d$	percent successes $(s/d)100$
9	0	512	512	100%
9	1	2304	4608	50%
9	2	4608	18432	25%
9	3	5376	43008	12.5%
*9	4	4032	64512	6.25%
9	5	2016	64512	3.125%
9	6	672	43008	1.563%
9	7	144	18432	0.781%
9	8	18	4608	0.391%
9	9	1	512	0.195%
10	0	1024	1024	100%
10	1	5120	10240	50%
10	2	11520	46080	25%
10	3	15360	122880	12.5%
10	4	13440	215040	6.25%
*10	5	8064	258048	3.125%
10	6	3360	215040	1.563%
10	7	960	122880	0.781%
10	8	180	46080	0.391%
10	9	20	10240	0.195%
10	10	1	1024	0.098%
11	0	2048	2048	100%
11	1	11264	22528	50%
11	2	28160	112640	25%
11	3	42240	337920	12.5%
11	4	42240	675840	6.25%
*11	5	29568	946176	3.125%
11	6	14784	946176	1.563%
11	7	5280	675840	0.781%
11	8	1320	337920	0.391%
11	9	220	112640	0.195%
11	10	22	22528	0.098%
11	11	1	2048	0.049%

n	q	successes (s)	number of comparisons $(2^n * \binom{n}{q}) = d$	percent successes $(s/d)100$
12	0	4096	4096	100%
12	1	24576	49152	50%
12	2	67584	270336	25%
12	3	112640	901120	12.5%
12	4	126720	2027520	6.25%
12	5	101376	3244032	3.125%
*12	6	59136	3784704	1.563%
12	7	25344	3244032	0.781%
12	8	7920	2027520	0.391%
12	9	1760	901120	0.195%
12	10	264	270336	0.098%
12	11	24	49152	0.049%
12	12	1	4096	0.024%

Table 1: Total Successes Per n (Deletions Banned)

The pattern of decreasing compatibilities is very distinct; whenever a 0 is changed into a 1 in our base X string, the percent compatibilities is halved. The formula for percent compatibilities, given any n and q, will be $P(q) = \frac{1}{2q}$, where $0 \leq q \leq n$. If n is allowed to approach ∞, then q will have to be as well, since our ideal q is $\lfloor q/2 \rfloor$. Therefore, $\lim_{q \to n} P(q) = \lim_{q \to n} P(q) = 0$. Given no deletions allowed, any percolation of large enough length will eventually run into a wall, and become incompatible. This follows along with our findings: if we take two random elements, even with the tiniest of probability p, theory says that *eventually* we will have two 1's occurring at the same time, and our percolation is doomed. We have once again shown that the utility of deletion is not obvious until it is taken away. Without deletion, percolations would be impossible to traverse, in the infinite case, and their usefulness would be compromised.

In other words, we cannot make any least upper bound on the probability p for compatibility with **positive probability** except for $p = 0$, when deletions are banned. Only by being assured **each and every** element is a 0 will a percolation be compatible. All other cases will deteriorate.

7.4 Allowing deletions - creating options

7.4.1 Computational Back-Up

Again, we follow the same procedure from Section 7.1.3, except this time, we will include the average path count, to try and discern the relationship between success and deletion options. The two charts with these results follow. The average paths, which now vary with every case, are also now included.

n	q	successes (s)	number of comparisons $(2^n * \binom{n}{q}) = d$	percent successes $(s/d)100\%$	average paths
2	0	4	4	100%	5
*2	1	5	8	62.5%	4
2	2	1	4	25%	1
3	0	8	8	100%	13.25
*3	1	17	24	70.83%	14.25
3	2	9	24	37.5%	6
3	3	1	8	12.5%	1
4	0	16	16	100%	36
4	1	49	64	76.5625%	48.5
*4	2	44	96	45.83%	27.5
4	3	14	64	21.875%	8
4	4	1	16	6.25%	1
5	0	32	32	100%	99.1875
5	1	129	160	80.625%	160.625
*5	2	170	320	53.125%	113.125
5	3	92	320	28.75%	44.6875
5	4	20	160	12.5%	10
5	5	1	32	3.125%	1
6	0	64	64	100%	275.75
6	1	321	384	83.59375%	522
6	2	566	960	58.9583%	437.812
*6	3	446	1280	34.84375%	214.375
6	4	167	960	17.39583%	65.8125
6	5	27	384	7.03125%	12
6	6	1	64	1.5625%	1
7	0	128	128	100%	771.453
7	1	769	896	85.82589%	1672.67
7	2	1718	2688	63.91369%	1626.84
*7	3	1816	4480	40.53571%	943.469
7	4	986	4480	22.00893%	359.734
7	5	278	2688	10.34226%	90.8906
7	6	35	896	3.90625%	14
7	7	1	128	0.78125%	1
8	0	256	256	100%	2168.38
8	1	1793	2048	87.54883%	5301.44
8	2	4878	7168	68.05246%	5867.75
8	3	6557	14336	45.73800%	3922.19
*8	4	4764	17920	26.58482%	1767.5
8	5	1945	14336	13.56724%	556.938
8	6	433	7168	6.04074%	119.938
8	7	44	2048	2.14844%	16
8	8	1	256	0.390625%	1

n	q	successes (s)	number of comparisons $(2^n * \binom{n}{q}) = d$	percent successes $(s/d)100\%$	average paths
9	0	512	512	100%	6116.9
9	1	4097	4608	88.91059%	16655.8
9	2	13185	18432	71.53320%	20681.9
9	3	21720	43008	50.50223%	15643.5
*9	4	20063	64512	31.09964%	8094.02
9	5	10918	64512	16.92398%	3015.42
9	6	3529	43008	8.20545%	813.891
9	7	643	18432	3.488498%	152.965
9	8	54	4608	1.171875%	18
9	9	1	512	0.1953125%	1
10	0	1024	1024	100%	17305.5
10	1	9217	10240	90.009766%	51953.2
10	2	34309	46080	74.4552951%	71560.7
10	3	67360	122880	54.8177083%	60428
10	4	76316	215040	35.489211%	35226
*10	5	52649	258048	20.402793%	15040.6
10	6	22649	215040	10.532459%	4809.2
10	7	6013	122880	4.8933919%	1138.59
10	8	918	46080	1.9921875%	189.98
10	9	65	10240	0.634765625%	20
10	10	1	1024	0.09765625%	1
11	0	2048	2048	100%	49076
11	1	20481	22528	90.913530%	161085
11	2	86633	112640	76.911399%	243861
11	3	198354	337920	58.698508%	227486
11	4	268446	675840	39.720348%	147419
*11	5	226764	946176	23.966366%	70713.1
11	6	123335	946176	13.035101%	25909.9
11	7	43573	675840	6.447236%	7287.5
11	8	9748	337920	2.884706%	1539.09
11	9	1271	112640	1.128373%	230.989
11	10	77	22528	0.341796875%	22
11	11	1	2048	0.048828125%	1

n	q	successes (s)	number of comparisons $(2^n * \binom{n}{q}) = d$	percent successes $(s/d)100\%$	average paths
12	0	4096	4096	100%	139448
12	1	45057	49152	91.668701%	496926
12	2	213640	270336	79.027580%	820460
12	3	560201	901120	62.167192%	838325
12	4	887074	2027520	43.751677%	597828
12	5	894364	3244032	27.569518%	317734
*12	6	594230	3784704	15.700831%	130597
12	7	264938	3244032	8.166935%	42136
12	8	78958	2027520	3.894314%	10605.6
12	9	15175	901120	1.684015%	2023.42
12	10	1714	270336	0.6340258%	275.994
12	11	90	49152	0.1831055%	24
12	12	1	4096	0.0244140625%	1

Table 2: Total Successes Per n (Deletions Allowed)

Here the patterns in decrease in percentage compatibilities are less clear, but, as opposed to the deletion-less case, we will demonstrate that this number will always be non-zero, and that compatibility can be assured as n approaches ∞.... Now, the average path counts, sorted again by the size n of string X and the number of 1's it contained. The incompatible trials **are** included in these counts; we want to see, given n, the expectation of paths that will be in **any** resultant percolation.

A few observations:

1.) There is always one path for $q = n$, since there is always only one compatible pair: $X = 1^n$ and $Y = 0^n$. This percolation looks like a perfect lattice: The greyed out section of this percolation can be disregarded; it is entirely unreachable. The total paths in this percolation are 2^n, which is equivalent to how many different strings were compared to $X = 1^n$. Therefore, the average is $2n/2n = 1$.

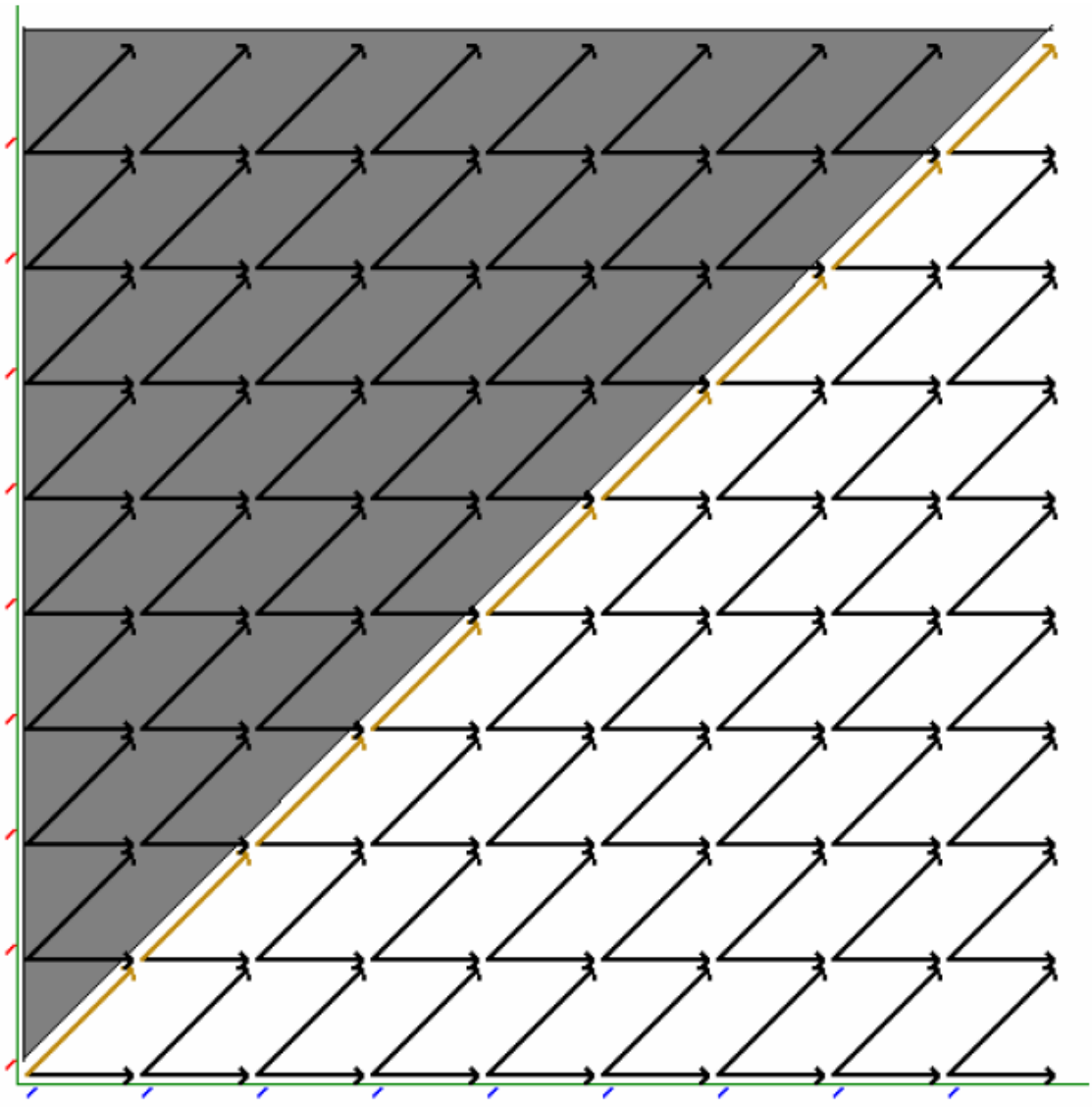

Figure 6: A Percolation of All 1’s and All 0’s

2.) The numbers for $q = 0$ vary greatly; it would be anticipated they would follow some standardization (i.e. be very perfect). This phenomena is worthy of explanation. For $q = 0$, every other string is compatible with it, and these will all have non-zero paths. Each of these percolations will have various path counts, since their construction will be altered by whatever Y string is being used, and there can be massive variations in regards to the paths that will be created. In other words, the $q = 0$ case offers the most variety in percolation graphs, but at the same time, not so varied that compatibility will be impossible. Again, this is directly related to the fact that there will be massive amounts of deletion schemes that can be taken when our traveller moves across these two strings. Therefore, our average path counts will be the most unpredictable.

8 Winkler's proof for p $<$ 1/2

8.1 Introduction

This section is an attempt to answer the big question posed earlier: what is the greatest probability p that still ensures that with **positive probability**, two sequences will be compatible? We say positive probability, because the probability space is endless, since we are dealing with discrete, infinite objects (these 0-1 sequence pairs). A probability space is defined as a triple: $(S, \mathbb{S}, P)$, where $(S, \mathbb{S})$ is measurable space (S is domain, $\mathbb{S}$ is measurable subsets), and P is the probability measure ($P(S) = 1$) [7]. For our percolations, S would be an infinite set of all 0-1 sequences, $\mathbb{S}$ is all 0-1 sequences that can be generated with fixed probability p (also infinite), and $P(\mathbb{S})$ would determine whether any given two strings from $\mathbb{S}$ would be compatible. Since both S and $\mathbb{S}$ are infinite, this P must be constructed in a special manner, such as the following.

Let $f_p(n)$ be defined as the probability that two substrings, composed of the first n elements of two 0-1 infinite sequences, will be compatible. This is a finite probability space: we can list all possible combinations, and find the ratio of compatible sequences to all sequences. We can,

and must, consider sequences "not likely" to be produced by the probability p. One example of this would be the event of $p = .10$ generating two strings size n of all 1's. The probability of this event is nonzero, and we must include it within $f_p(n)$. We now have a function dealing 41 with finite cases, which we will now extend. Now define a function $\ell(p)$ as

$$\ell(p) = \lim_{n \to \infty} f_p(n) = P(S)$$

This is what the infinite case is concerned with: what happens to our $f_p(n)$ as n gets larger? What happens now that our domain S and subsets $\mathbb{S}$ approach infinite cardinality? $\ell(p)$ cannot simply list all compatible pairs and divide by the total possible pairs; continuous probability methods must be used.

Therefore, we can only say that it is with a "positive likelihood" that we will generate infinite compatible sequences with this probability p. Various methods have been undertaken by several individuals to find a greatest lower bound for this p. Theoretical results by Peter Winkler, the creator of this percolation, and Harry Kesten, have both concluded the upper bound $p < 1/2$ [3]. Peter Gács have provided the lower bound $p \geq 10^{-400}$. John Tromp, with computer simulations, but without formal proof, has suggested a lower bound $p > 0.3$. Thus, there is plenty of room for improvement between these bounds.

Throughout this proof, special care is taken at each step to show that it is possible to conduct, as this technique hinges on precise ratios of compatible strings, and that a small, non-zero amount of possibilities will still exist once the smoke clears.

8.2 Two carefully specified prefixes

We begin with two strings, of length $2n+1$. We specify that these two strings will have $n+1$ 0's, and n 1's, and also that they are compatible. We now have two prefixes that have just less

than half 1's, and just over half 0's. Since we are making special demands of our infinite strings (i.e. that they would produce such "choice" prefixes), we must demonstrate that there exists possibility that, with sufficient *n*, our scenario is possible.

1.) **As *n* approaches ∞, the "actual probability" of our prefixes will converge to the set probability of 1's occurrence.**

This is not hard to demonstrate: statistics tells us that if we make more and more trials on a population, in search of a particular result, and this population has a set likelihood of our target, our study's findings will be closer to reality as we poll more and more people. In our case, we generate more and more elements within our strings, and the frequency of 1's should approach our set probability *p*. Therefore, with a large enough *n*, the actual probabilities (ratio of 1's to 0's) will be nearly equivalent to *p*.

2.) **If $p < 1/2$, with a sufficient *n*, then our prefixes' actual probabilities will be less than 1/2.**

This is a furthering of the idea presented in (1): if this set *p* is less than 1/2, then with a large enough *n*, our fixed probability will be less than 1/2 as well. This is sufficient in showing we can "ask for" a string prefix of some size, that will have $n+1$ 0's and *n* 1's, for some huge *n*.

3.) ASSUMPTION: **These prefixes will be compatible.**

Here is the hypothetical step of our setup. We must assume at this point that these sequences will be compatible. It has been shown that when prefixes have this ratio of 1's to 0's, there exist compatible sequences, so it is **possible** to generate them on demand. Thus, depending on one's perspective, we either generate pairs until we get to this compatible sequence, or we skip

ahead to the point where it has been generated.

8.3 Delete a zero from each

If we make a legal deletion within each string, not only will compatibility be preserved, but also we will now have exactly 50% 0's and 50% 1's.

1.) Result: **Our two sequences are now complimentary.**

Since our two strings have half 1's and half 0's, they must now be complimentary to each other. That is, if $X[i] = 0$, then $Y[i] = 1$, or vice versa. This can be proven by contradiction: if two 0's were in equivalent spaces, then two 1's would be as well, and compatibility would be impossible. In short, we would need to make more deletions to our strings to change this alignment, and any deletion would serve to create another pair of 1's in the same position, therefore, these strings are not compatible, and could not result from our original setup.

8.4 Reinsert a 0 into each string

We now wish to replace our deleted 0's, but this time, we can put them anywhere within our strings, in an attempt to create strings that could have possibly been used to create the current n 0's, n 1's situation.

1.) **Since our strings are compatible, when we reinsert the 0's, the new pair will also be compatible.**

Compatibility, by definition, means we can cast out 0's to create complimentary pairs. If we insert 0's into **any** complimentary string pair, we are creating two complimentary string pairs.

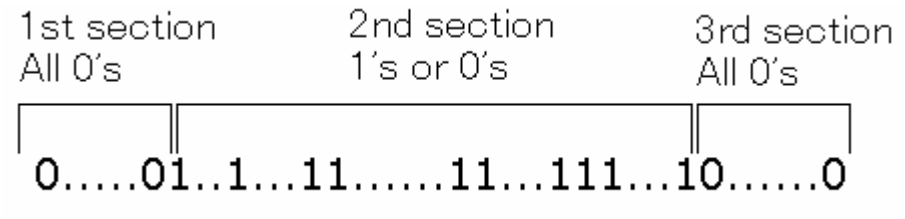

Figure 7: Three regions of our chosen prefix

2.) **Certain reinsertions of 0's will not change our situation.**

Let us divide our prefixes into three parts: the region from the beginning to the first 1, the region from the first 1 to the last 1, and the region from the last 1 to the end of the prefix. Inserting 0's in either the first or third region will not greatly change the different types of strings that will be compatible with the one we are altering. This is because the greatest changes in possible compatible partners come when we divide up existing walls (groups of 1's), or spread existing 1's apart (make the holes in between these walls even larger). 3.) **Insertions between 1's will change our situation.** As a result of (2), we will only insert our 0's into positions that lie in between our 1's. Since there are n 1's in our string prefix, we have *at least* $n-1$ potential places to insert our 0's. There will be more than this, if there are 0's already lying in between existing 1's, but for the general case, we will assume that the entire second region (as defined in (2)) is composed of 1's.

8.5 Interpreting what we have done

In these steps, we have gone from $p < 1/2$, to $p = 1/2$, using our carefully selected prefixes. Then, we showed that, if we begin with a pair of strings, with equal amounts of 1's and 0's, then we can "step back" to $p > 1/2$, and demonstrate that there must be compatible sequences in this situation. This is the positive probability: that we can be sure there will exist at least one (and in fact, there are $n+1$ of them) string that will be compatible with whatever our resultant string is. As $n \to \infty$, $n+1$ is still above zero; this possibility still exists for very large n. We

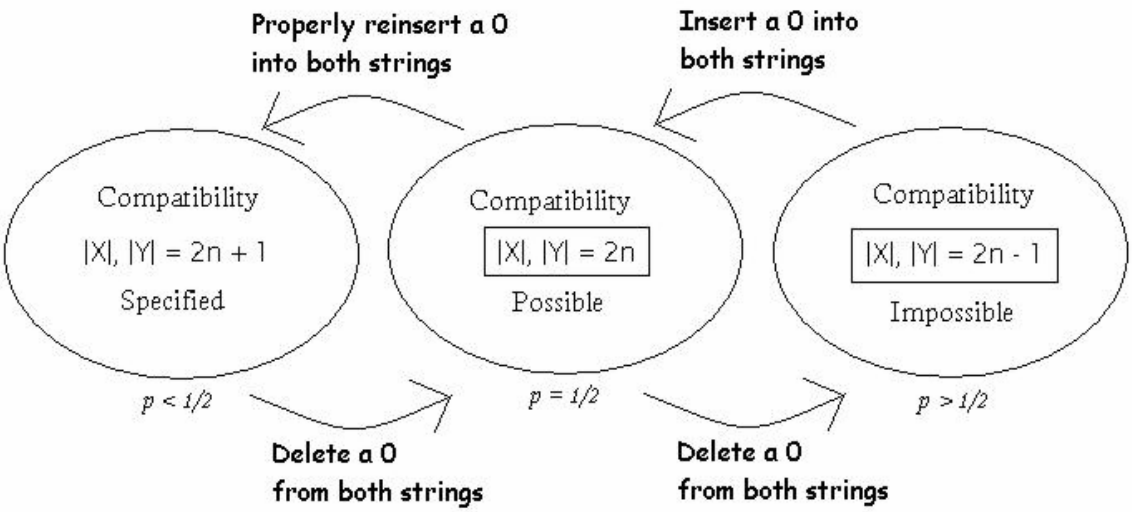

Figure 8: Diagram of Winkler's Method for $p < 1/2$

may not necessarily be fortunate enough to have these pairings come together at the right time, but the chance of it is non-zero, which is what Winkler and Kesten have shown. In contrast, this is not possible for the other two possibilities: $p = 1/2$ and $p > 1/2$.

The following picture summarizes the findings of this section.

9 Conclusions

9.1 From small to large - a compacted problem

Through this investigation, we have demonstrated that percolations are problems that are scalable, through two differing means. First, through Gács and his use of re-normalization, turned a percolation into a system of "mazeries", each with properties that could be used to answer the lookahead and compatibility problems. Also, using deletion and reinsertion, we were able to define non-square percolations, and demonstrate that every percolation is built upon one smaller than it. This was essential in our analysis of the infinite case; we no longer have to generate massive percolations to discover what will happen as n becomes very large, we can make

determinations with smaller examples that are easy to enumerate and classify thoroughly. This all eventually led up to the proof of $p < 1/2$ using Winkler's method: that there was enough "self-similarity" and decomposition technique to build a compatible percolation, remove parts of it, and then add those parts back in different ways. Our reconstructions 'can' be compatible, therefore those generated at random 'can' also be compatible.

9.2 The next step

It was stated earlier there now exist several assertions as to what the least upper bound of p are. For example, simulations by John Tromp, which were reaffirmed by another source, put the value at less than .3, yet we have demonstrated that this value is less than 1/2, around .438184657762795248814 3, according to Harry Kesten [4]. A possible continuation of the current progress on percolations would be finding a more precise determination about which of these numbers is closer to the truth (the truth being either percolations built using more improved computational trials or combinatorial techniques).

There are many other resultant research opportunities from the current status of Winkler percolation development. The three dimensional case (three strings instead of two) may have dramatic effects on our bounds for probability p, and the issue of phase transitions (just how suddenly does the rate of compatibilities drop when p is increased) has not been fully answered. The computational findings suggest it is not as sharp as previously thought, but it must be remembered these findings have inherent error, and thus are imperfect [4].

References

[1] I. Benjamini, R. Lyons, and O. Schramm. “Percolation beyond $\mathbb{Z}^d$, Many Questions and A Few Answers”. *Electronic Communications in Probability*. No. 1, pp. 71 – 82. 1996.

[2] Ioana Dumitriu; Prasad Tetali; and Peter Winkler. “On Playing Golf with Two Balls.” *SIAM Journal of Discrete Math.* Vol. 16 , No. 4, pp. 604 – 615. 2003.

[3] Peter Gács. “Compatible Sequences and a Slow Winkler Percolation”. *Combinatorics, Probability, and Computing*. Vol.13, No.6, pp.815-856. 2004.

[4] John W. Hajeski. “Winkler Percolations: A Computational Analysis”. Worcester Polytechnic Institute Major Qualifying Project. 2005.

[5] Eric W. Weisstein. “Percolation Theory.” From *MathWorld*–A Wolfram Web Resource. http://mathworld.wolfram.com/PercolationTheory.html

[6] Eric W. Weisstein, et al. “König’s Lemma.” From MathWorldA Wolfram Web Resource. http://mathworld.wolfram.com/KoenigsLemma.html

[7] Eric W. Weisstein. “Probability Space.” From MathWorld-A Wolfram Web Resource. http://mathworld.wolfram.com/ProbabilitySpace.html