Minimum Spanning Rectangle
A complete case:
constructing a solution from theory to implementation

Uriel Gonzalez
Graduate Student
Department of Computer Science
University of Texas Rio Grande Valley

August 4, 2019

1 Preamble

In progress ...

This document is a draft. It is intended to be a blog post. The language is informal and the reader is assumed to be someone with introductory level skills in computer science. The idea is for this paper to be a guide for new programmers on how to make software.

2 The Problem

The problem was presented in the MI Lab at UTRGV as follows:
A file is given with numbers written as triplets. The first two numbres represent a point in a Cartesian coordinate and the third represents a radius. The radius could be any $r \in \mathbb{R}^2$ *such that* $r \geq 0$. *Thus, the file represents an arbitrary(finite) number of circles in the Cartesian plane. Find a rectangle that contains all the circles and that also has the smallest possible area.*

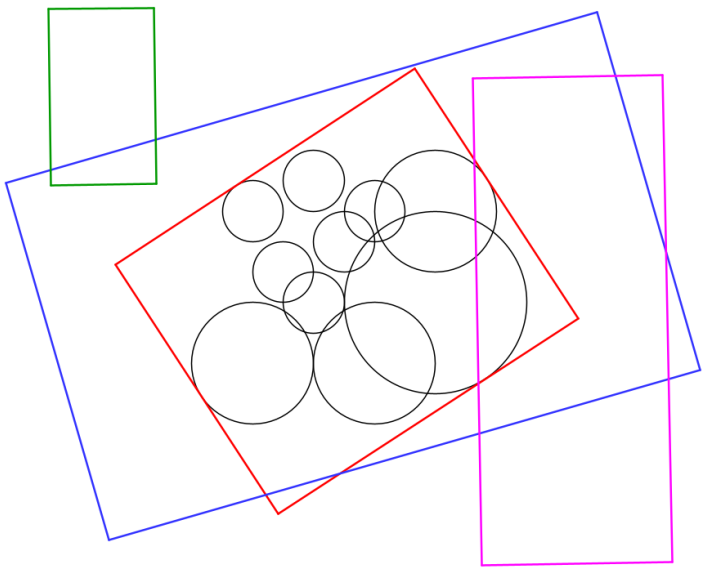

Figure 1: Example of the Minimum Spanning Rectangle Problem

Described in the language of sets, from the set of all rectangles in the Cartesian plane, find a subset of rectangles that have the property of containing all circles. From that set, find the rectangle with the smallest area. From the rectangles in Figure 1, the green and the pink one are examples of rectangles that belongs only to the set of all rectangles in the Cartesian plane. The blue and the red one are examples of rectangles that belong to the set of rectangles containing all the circles. An interesting fact is that the definition of the problem allows for the radius of a circle to be zero. A circle with radius zero is just a point. Thus, this problem allows the use of both circles and points. I decided to call the rectangles that contain all circles Spanning Rectangles (SR). I called the spanning rectangle with the minimum area the Minimum Spanning Rectangle (MSR). From now on an individual circle is represented by C and the set of all circles as S. This is due to the fact that a circle is also a set, so this is more convenient.

Definition 1. *Let S denote the set of all circles in the input file. A spanning rectangle is a rectangle that contains all the circles in S.*

Definition 2. *Let R denote the set of all spanning rectangles. Let A be a function that maps from R to $\mathbb{R}$. A is the function that maps each rectangle to its area. The minimum spanning rectangle is the spanning rectangle $r^* \in R$ such that for every spanning rectangle $r \in R$, $A(r^*) \leq A(r)$.*

Notice how we have only defined the existence, but not the definition of the area function A. This is a paper in computer science. Unlike mathematics, were atomic information is easily accessible - you don't have to bother about arrays, how data is ordered, storing the area into temporary variables, etc. In computer science information is stored as a data structure in RAM. As a designer, we get to decide how to store that information. We cannot describe a function without first deciding what data structure we are going to use to store the information. We are delaying the definition of the computation of the area function until we decide how we are going to represent each rectangle R.

Note that this definition allows for multiple rectangles to be the minimum spanning rectangle.

3 The solution

3.1 Foundations

Creating algorithms is like art: there is no formula to create one. Nevertheless, just like in art, there are useful guidelines - like painting the background first and then painting the foreground. It is my contention that there is a best-approach to algorithm design. What follows is my opinion on the subject, so take it with a grain of salt.

Every algorithm takes an input and produces an output. I call the set of all possible inputs the input space and the set of all possible outputs the output space. Often each input has a unique output, but this is no necessarily the case. It is always a good idea to have a couple of input-output pairs to help you in figuring out how to solve the problem. For any given input the algorithm returns a subset of the output space. I like to call this subset the solution space. It is easy to describe these sets once the algorithm is finished, but during its creation things are more tricky. It is best to explain this with an example: a sorting algorithm. The input space of a sorting algorithm is the set of all lists of real numbers. The output space is the set of all lists of real numbers such that the numbers are listed in ascending order. All of this might sound silly, but having all these structures is going to help us develop algorithms. For a sorting algorithm, an arbitrary input is an unordered list of n elements. During the development phase one should define the solution space so that it excludes anything that is clearly not a solution. There are many such definitions, so it requires experience and experimentation. Lets say that the input is [4,7,5]. Clearly [3,6,4] and [1,2,3] are not in the solution space because both have numbers that are not listed in the input. This is true even though one of them is ordered. Assuming that we did not know how to sort, one good candidate would be [7,5,4] or [5,7,4]. In this instance we could define the solution space as all the permutations of the list given as input, then we could try each possibility until we find the one that satisfies the condition. I call this the trivial algorithm (analogous to the trivial solution of a differential equation) (commonly referred to as brute force). Provided that the solution space is finite, this algorithm always works. It is also the most inefficient algorithm. It might sound silly but there are some problems, like the NP class, that can only be solved this way. As a developer, your task is to find something better than the trivial algorithm. To achieve this, we use properties, operations, and decisions to navigate through the solution space. The idea is that, roughly speaking, after each step of the algorithm, the solution space shrinks. Once you reach the last step of the algorithm you have the solution of the problem. There are two ways of reducing the solution space: construction and search. In a construction approach, you use functions/properties to create a subset from the solution space, with each iteration being a set with fewer elements. Due to the fact that the properties are precise, the solution is guaranteed to be in each successive subset. In a search approach, we start at some point in the solution space and we use an indirect metric to move closer to the solution. This movement is from a region in the solution space with higher error to one with fewer error. Unlike constructive algorithms, this does not guarantee that the solution is in the new subset. Search algorithms generate a sequence of guesses. Each sequence can be parametrized by an initial point and a metric. The perfect search algorithm is one where no matter what the initial point, all sequences converge to the same point, which is the solution. Sometimes, due to the nature of the problem, the sequences are sensitive to the initial point - and perhaps parameters of the metric function. The metric function is a noncharacteristic/nondeterministic property of the solution. It is used as a guide to discriminate how good a solution is from another solution. For constructive solutions, the properties need to be proved useful. For search solutions, the metric needs only guide to the solution, which can be proved loosely/informally/partially. The choice of this metric will also affect the rate of convergence of the sequence.

My original intent was to find geometric theorems that would help me construct a rectangle with the desired property. I began like anything good begins, with pencil and paper. I drew some circles and rectangles that contained them. The first thing that I noticed was that the number of possibilities was huge. There are infinitely

may rectangles in the Cartesian plane that contain all the circles. In algorithms, objects and their properties are described mathematically. It may seem silly, but in mathematics everything needs to be proven. So we cannot simply claim that there are infinitely many rectangles, we need to prove that this is the case. To achieve this, I am going to use the principle of mathematical induction. First I will prove that there is one rectangle that contains all circles. Then, I will prove that I can use that rectangle to create as many rectangles as I wish that also contain all the circles.

Theorem 1. *For any finite set of circles S in $\mathbb{R}^2$, there exists a spanning rectangle*

Before we prove this theorem we need to establish what does it mean for a rectangle to contain all circles. There are many ways of doing this. I think that the most intuitive and simplest way of doing this is as follows: *all rectangles in the Cartesian plane have an area. The area is a real number, but geometrically it can be interpreted as a region of $\mathbb{R}^2$. Every circle represents a set of points from the Cartesian plane. Thus, for each circle $C \in S, C \subset \mathbb{R}^2$.* Since both the circles and the area of the rectangle can be represented as sets, it makes sense to say that a rectangle with area set A contains all circles if for all circles $C \in S, C \subset A$. But how do we define A? Before we can describe the area of the rectangle we have to describe the rectangle itself. Noticed how we already defined how a rectangle can contain all circles before defining the rectangle itself! That is really crazy! That is one of the most beautiful - and also the must puzzling - thing about this language called mathematics: you can start anywhere you want. Some choices are better than others and there is no easy way of knowing how to choose. So how do you describe a rectangle in the Cartesian plane? The easiest way would be to provide its four corners, but there is an issue with that. Take as contrast the circles provided in this problem: they are represented by a center and a radius. From this three values x_c, y_c, and r we can construct the circle $C = \{(x, y) \in \mathbb{R}^2 : (x - x_c)^2 + (y - y_c)^2 = r^2\}$. This formula is the perfect description of a circle. As long as $r \geq 0$ any circle given in this format is a valid circle. There is no way for the input to be invalid. We can use a random number generator of nonnegative real numbers with the confidence of knowing that all inputs are valid. In the case of a rectangle, the four corners we wish to use are not any four random points: they must be at right angles with each other. If we receive four points as input, can we be sure that they represent a rectangle? Should we trust the user to input it right? Would our algorithm work with an invalid input? Would the output be meaningless? Would it lead to a compile error , runtime error, or logic error? Notice how those issues do not arise in the case of the circles. What is the best way of describing a rectangle? The first idea I had was to use two points that form the diagonal and the size of two sides of the rectangle. Notice that with this new description, every input has to be correct. False, some sides are not possible with right angles. This description gives parallelograms, not rectangles. It is hard to describe the right angles of the rectangle; it would require the use of Pythagoras' Theorem. Despite that, not being able to describe the rectangle does not stop us from using it in our theory. To my knowledge, this is the best way of describing a rectangle in the Cartesian plane:

Theorem 2. *The minimum amount of information required to represent a rectangle in the Cartesian plane is a point, a magnitude, a sign, and a vector.*

To prove this theorem, I am going to show that from the given information I can produce the four corners of a parallelogram that meets the criteria of a rectangle. To achieve this I am going to use vectors. Explaining vectors goes beyond the scope of this text.

Remark. Unit vectors have a hat on top. So if $|\vec{a}| = 1$, then it is customary to refer to it as $\hat{a}$

Proof. Let $\vec{p_0} = (x_0, y_0)$ be a point vector representing the point provided. Let $\vec{v}$ represent the vector provided. Let $\vec{u}$ be a vector such that $|\vec{u}| = 1$ and $\vec{v} \cdot \vec{u} = 0$. There are two solutions to this equation. Use the sign to choose one of the two solutions. The vector $\hat{u}$ is the normal unit vector to $\vec{v}$. Scale the vector by the magnitude m given. With these two vectors we proceed to find the other three points. Let $\vec{p_1} = \vec{p_0} + \vec{v}, \vec{p_2} = \vec{p_0} + m\hat{u}$,

and $\vec{p_3} = \vec{p_0} + \vec{v} + m\hat{u}$. Since $\vec{v}$ and $\hat{u}$ are perpendicular, then by the parallelogram rule of vector addition this parallelogram is a rectangle. □

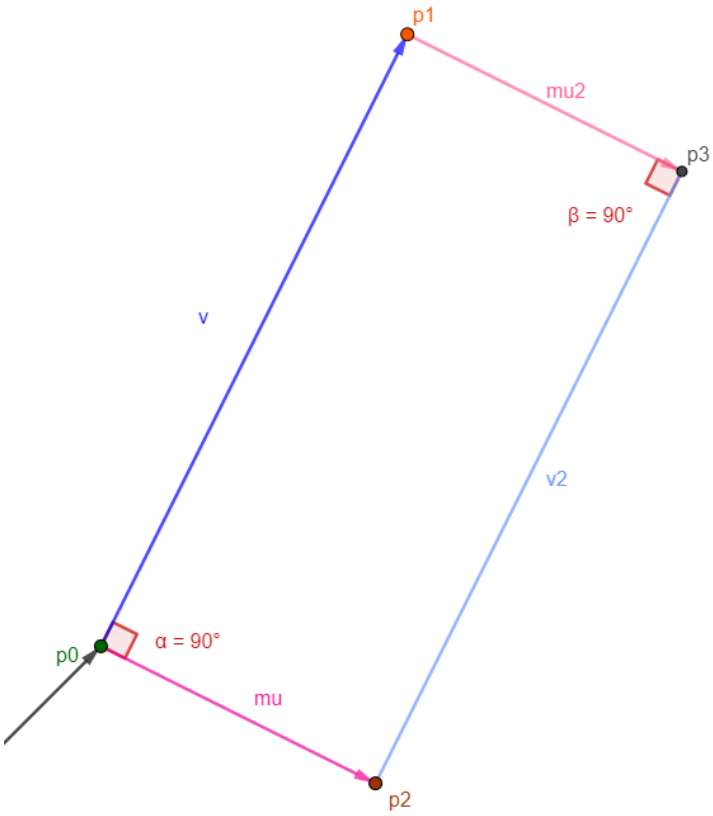

Figure 2: Theorem 2 visual aid

How do you know there exists always a normal vector? Its a thing from solving $\vec{v} \cdot \vec{u} = 0$, where there are always two solutions. How do you prove that you can represent every possible rectangle? This is a tricky one. We know for sure that given this information we can construct a valid rectangle. Is there a rectangle that we cannot represent? If so, our description is incomplete. One way of thinking about this is by looking at all variables. The magnitude and the magnitude of the vector can have any value. Thus, as size is concerned, we can represent a rectangle of any size. This also means that we can represent a rectangle with any area. We can also vary the direction of the vector. Thus we can represent a rectangle with every orientation. Since we can vary the point indefinitely, we can represent a rectangle at any location in the Cartesian plane. Thus, we can represent a rectangle of any size, any orientation, and at any location. So yes, we covered every possibility: *with this description we can represent any rectangle in the Cartesian plane*. Moreover, our data structure can do more. If we let the magnitude be zero, then we can represent a segment. If we let both the vector magnitude and the magnitude be zero we can represent a point. It is an interesting data structure.

All that aside, there is an important critique about this description. Lets refer to the 4-point data structure as A and the point-magnitude-sign-vector data structure as B. For both data structures a point consists of

two floating point values. So does a vector. For A, the description uses 8 values. On the other hand, B uses only 6 values. That means we save 2 numbers for each description. Assuming that each number is a 64 bit floating point and that we are representing 100 rectangles, we save 1600 bytes. As long as the magnitude and the vector are greater than zero and the signs are properly distinguished the structure encodes a valid rectangle. It is relatively easy to check if all these conditions are met. On the other hand, if we receive four points, we need an algorithm to check if they describe a rectangle. A big disadvantage is that B requires computation to construct the rectangle as opposed to the A, which requires no computation. It is as if A were compressed. Both structures require a verification of the data. Which is better? That is a tough choice. These days both memory and computation are relatively cheap, so it should be a matter of convenience rather than efficiency. It might be the case that validating A is cheaper than validating B. A requires an algorithm to construct a data structure. B requires an algorithm to validate the data structure. Both algorithms take inputs of constant size, so it is reasonable to believe that both have constant complexity. I think it is easier to build an algorithm knowing that the data structure you are using has all the characteristics you are assuming it to have. This can be achieved by using a robust data structure, or a weak data structure with an algorithm to check its validity. As a designer, right now I am considering both, even though I have not found an algorithm to check if four points make a rectangle. The main advantage of A over B is that it contains information that can be used for things other than the representation of a rectangle without making further computations. There is an issue with A: do we care just about the four points or do we care also about $\vec{v}$ and $\hat{u}$? From the design of B, any computation that uses the rectangle represented by B would have to compute the four points of the rectangle. This would occur at the start of each function call and it will be lost at the end of the call. If a lot of functions are going to use the points, then the data structure should reflect this. As an alternative, the points can be computed once and stored in local variables so as to optimize their use. I used this strategy in the implementation of the code, which I will explain in the Implementation section. As a designer you can choose which data to store and for how long, so there is no need to limit oneself. You should strive to use the least amount of computation and reuse variables as much as you can.

Now that we have a description of the rectangle we need to define the area set of the rectangle. This one is a little easier. Remember the vectors that we used in the previous proof? We can parametrize them to create lines. We use four vectors to create four lines. We can use these lines to describe a system of inequalities that describe all the points inside and on the rectangle. The difficult part is to account for all possible rectangles. Each of the four lines must be an inequality that tests whether the points are above or on the line or below and on the line. There are two cases for the inequalities: either on or above the lines or on or below the line. If the given vector is perpendicular or horizontal, we get special cases where the inequalities use a value rather than a line - this represent a horizontal or vertical line. It is better to work on a general case first and incorporate the special cases latter. In this case there is no easy way, we have to account for all possibilities. One of the mayor pitfalls that designers fall into is that they try to find a simple, elegant solution to every problem. There are times when this is not possible. This is one of those times. To determine the inequalities we need to consider the quadrant of $\vec{v}$ and the sign given. It is represented in the following figure:

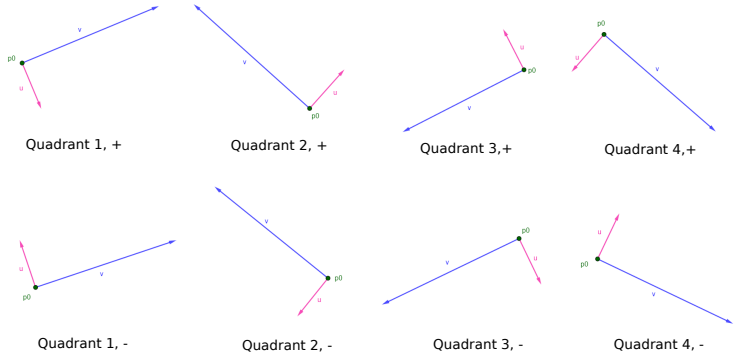

Figure 3: All the cases

Notice how we did not define what it means for a vector to be in a particular quadrant. At his point it really is not that relevant, so long as we know it is definable. You should only define things whenever they are needed. So how do you know when something is needed? We did not need to specify all the details of how to determine if all circles are inside a rectangle. How do we know that we need to define area set A? I know I have to because I wrote all the proofs. This documents is an after-the-facts account. In practice you should define things only when you need them. As a designer it is ok to work with abstract objects, where we loosely describe what objects do but we do not defined how they do it. Sometimes, as one develops software some objects end up not being used. Defining every detail of such object in advance would have been a waste of time. Loose definitions also allow you to contemplate other design options. It gives you the leeway to find other possibilities.

The area of a rectangle can be defined by four linear inequalities. A line can be represented using two points. This is a widely accepted axiom proposed in *The Elements* by Euclid. Let $L(p_i, p_j)$ represent the line crossing through points p_i and p_j. As a researcher, you should not be afraid of creating your own notation, as long as you take into consideration the notation widely used and you describe yours with detail. After drawing some examples I realized that the the inequalities of the lines depended on the orientation of the rectangle; meaning, sometimes the lines were greater than and other times less than. Moreover, note that $L(p_2, p_3)$ always has the opposite relation of $L(p_0, p_1)$ and $L(p_1, p_3)$ has the opposite relation of $L(p_0, p_2)$. Therefore, $\vec{v}$ and the sign are the only variables necessary to figure out how to establish the inequalities. More precisely, the inequalities are determined by the sign and the quadrant in which $\vec{v}$ lies. There is a total of 16 irreducible cases: 8 base cases and 8 special cases. The base cases correspond to the same cases in Figure 3.

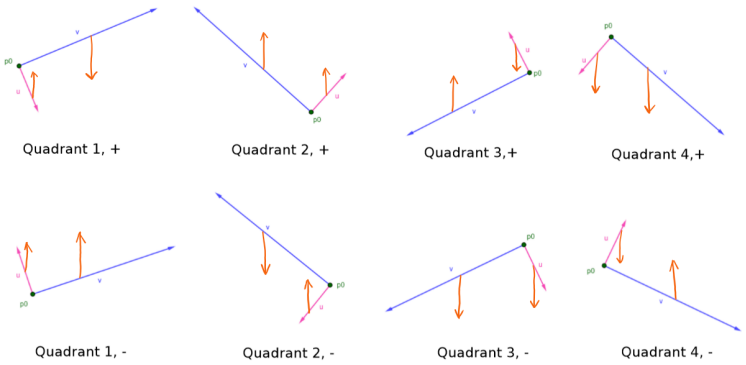

Figure 4: Inequality cases market by orange arrows

1. case 1: $y \leq L(p_0, p_1), y \geq L(p_0, p_2)$
2. case 2: $y \geq L(p_0, p_1), y \geq L(p_0, p_2)$
3. case 3: $y \geq L(p_0, p_1), y \geq L(p_0, p_2)$
4. case 4: $y \leq L(p_0, p_1), y \geq L(p_0, p_2)$
5. case 5: $y \geq L(p_0, p_1), y \leq L(p_0, p_2)$
6. case 6: $y \leq L(p_0, p_1), y \leq L(p_0, p_2)$
7. case 7: $y \leq L(p_0, p_1), y \leq L(p_0, p_2)$
8. case 8: $y \geq L(p_0, p_1), y \leq L(p_0, p_2)$

Notice how there appears to be symmetry among the cases. This has to do with rotations along the x-axis and y-axis. Unfortunately, the logic required to compute this might be as tedious as coding all the cases, not to mention that attempting to do such thing requires careful design. Is best to keep it simple and tedious.

The special cases arise from when the rectangle is aligned with the axes. NOT DONE.

I would like to take the time to mention that, whenever you work on theoretical stuff, it is important always to work and present things as intuitive as possible. It is often the case that intuitive descriptions are long and tedious as opposed to shorter and elegant definitions. The elegance usually arises from some clever manipulation. The issue is that such manipulations obscure the real intent of the objects you are trying to describe. As a rule of thumb, first describe things in the most intuitive way, then show how it simplifies into something nicer.

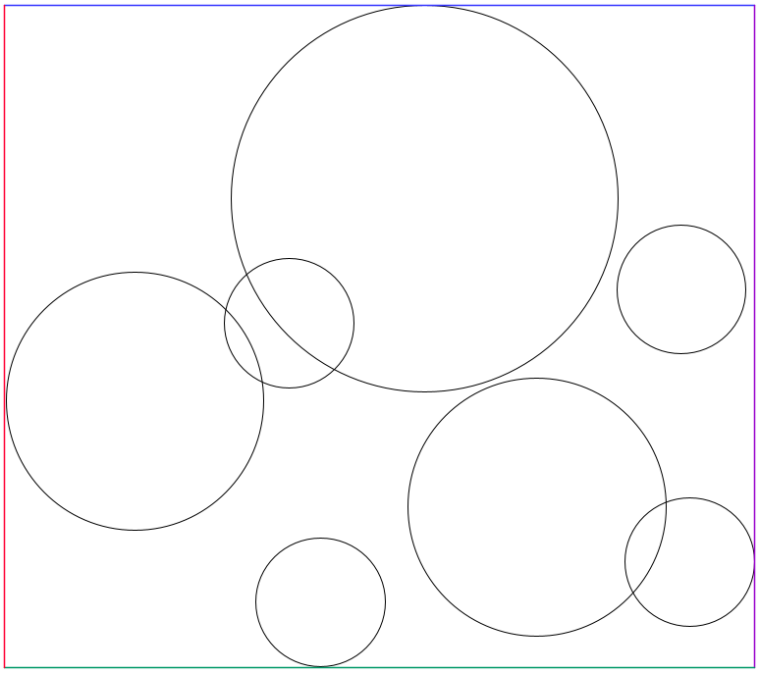

Figure 5: Theorem 1 visual aid

Now we proceed to prove Theorem 1. How would you draw a SR? For any set of circles, the easiest way is to draw a rectangle that has one side parallel to the x-axis - this implies that its opposing side is also parallel to the x-axis and that the other two sides are parallel to the y-axis. This is represented in Figure 5. You can think of taking a ruler, placing it parallel to the x-axis so that every circle is above it, sliding it upwards until it touches a circle, and drawing a line (green segment). You do the same, but this time so that all circles are below the ruler and you slide it downwards instead of upwards. Then you repeat the process but with respect to the y-axis. Rigorously, the argument goes as follows:

Proof. We begin by defining useful subsets of the points in every circle to construct a spanning rectangle.

Lemma. *Every circle is represented by a radius r and a center (x_c, y_c). Every point in the circle must satisfy the condition $(x - x_c)^2 + (y - y_c)^2 = r^2$. Then for every point $(x, y) \in C, x_c - r \leq x, x \leq x_c + r, y_c - r \leq y$, and $y \leq y_c + r$.*

Proof. First we present some useful inequalities. Since $(x-x_c)^2+(y-y_c)^2 = r^2$ it follows that $r^2 \geq (y-y_c)^2$. Therefore, $\sqrt{r^2 - (y - y_c)^2} \geq 0$. It also follows that $r^2 \geq r^2 - (y - y_c)^2$. Therefore, $r - \sqrt{r^2 - (y - y_c)^2} \geq 0$. Since $r \geq 0$ and $\sqrt{r^2 - (y - y_c)^2} \geq 0$ it follows that $r + \sqrt{r^2 - (y - y_c)^2} \geq 0$.

Solving for x_c and subtracting r, we obtain that $x_c - r = x - r - \sqrt{r^2 - (y - y_c)^2}$. We can rewrite this equation as $x_c - r = x - (r + \sqrt{r^2 - (y - y_c)^2})$. Clearly, x is being reduced by a nonnegative number. Therefore, $x - (r + \sqrt{r^2 - (y - y_c)^2}) \leq x$. Therefore, $x_c - r \leq x$ as we seek to prove.

Similarly, solving for x_c and adding r, we obtain that $x_c + r = x + r - \sqrt{r^2 - (y - y_c)^2}$. Since x is being increased by a nonnegative number, it follows that $x \leq x + r - \sqrt{r^2 - (y - y_c)^2}$. Therefore, $x \leq x_c + r$ as we seek to prove.

Without loss of generality, the same two arguments can be applied to prove that $y_c - r \leq y$ and $y \leq y_c + r$. □

From each center extract the x value and subtract the radius. Refer to these values as set L. For each center extract the x value and add the radius. Refer to these values as set R. From each center extract the y value and subtract the radius. Refer to these values as set B. From each center extract the y value and add the radius. Refer to these values as set T.

USE PREV PROPOSITION TO DO A DIRECT PROOF OR CORRECT THE PROOF BY CONTRADICTION

Since C is a finite set, it follows that L,R,B, and T are also finite sets. Therefore, each one has an infimum and a supremum. Let $l = inf(L), r = sup(R), b = inf(B)$, and $t = sup(T)$. A rectangle aligned with the axes can be represented by two points. Let (l, b) and (r, t) be these points. PROOF IN PROGRESS. □

This is not the only way to create a spanning rectangle. Another way would be to find the mean of all centers, find the center that is furthest away from the mean. Add to that distance the radius of that circle. Make a circle from the mean with a radius with this value and inscribe the circle into a square. A square is a rectangle, so the spanning rectangle exists.

Theorem 3. *There are infinitely many spanning rectangles*

Before we continue with this proof we need to define what does it mean for a rectangle to contain another rectangle. It might seem tedious but this is necessary. More than a definition, we need a procedure to determine if a rectangle includes another.

COMPLETE THESE PROCEDURE.

Proof. Assume that a spanning rectangle already exists. For any real number $\alpha > 1$ scale the spanning rectangle by α so that only its size, not its orientation, is changed. The new rectangle is larger than the spanning rectangle. Therefore, this new rectangle contains the spanning rectangle. Since the spanning rectangle contains all circles and the new rectangle contains the spanning rectangle, it follows that the new rectangle is also a spanning rectangle. □

The majority of those choices are clearly not the MSR. As I drew rectangles it became clear that they all shared one thing in common. I modified my definition to be so that *spanning rectangles are all rectangles that contain all circles such that each side touches at least one of the circles*. In Figure 1 this corresponds to the red rectangle. This made more sense. Even with this new definition, there were some SRs that were clearly not the MSR. Others were very close to each other, so any of those could be the MSR. At this point I encountered my first obstacle: I could find no way to discriminate between good SRs from bad SRs. I could find no property that good SRs had that bad SRs didn't(see Figure 6). The only way to tell them apart was to compute the area. Why would finding such property be important? Because those properties can be used to construct a MSR rather than computing the areas of a set of SRs and then choosing the minimum one. Construction is always more efficient than searching, but searching is easier to implement. Some problems do not have a way to construct a solution. A classic example of a problem with such property is the Traveling Salesman problem, were it is simple to search through all possibilities (at the expense of computing time) but there is no way of constructing a path that has the desired properties. Another similar problem is that of prime numbers. The only way to know if a number is prime is to test all other numbers before $\sqrt{n}$ to see if they divide the number. There is no other easy way. An example of a problem were solutions can be constructed is statistical linear regression, were the best fit line can be found using calculus. Another example is the merge sort algorithm, were the merge portion of the algorithm takes advantage of the property that the lists being merged are already sorted. Both linear regression and merge sort use properties of the objects they deal with

to move logically/beneficially towards some objective. The MSR problem I describe here has more in common with the Traveling Salesman problem and primes in the sense that, to this point, I have not found properties that I can use to construct a solution. At this point, the MSR problem gave me the same feeling I get when I am trying to solve a system of linear equations were there are too many free variables: I feel like I do not know where to start, that any starting point is good enough but I know that one is easier than the others. This kind of thought always reminds me of the Knapsack problem. There is a special type of difficulty with problems where there are too many possibilities and there is no way of telling which is better without having to do a large computation. Checking each possibility is tedious and inefficient.

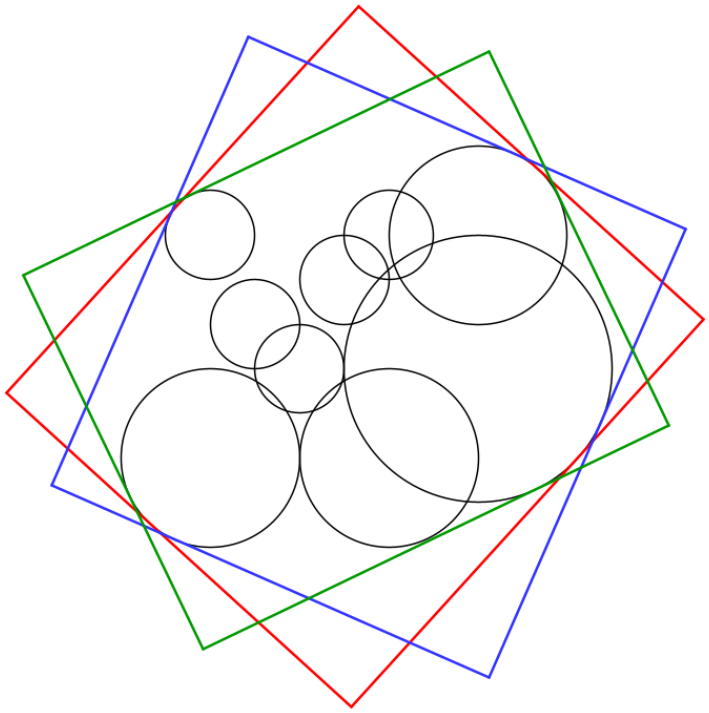

Figure 6: Some Spanning Rectangles

3.2 Relaxation

From Theorem 1 the summing and adding of the radius corresponds to points in the circle. Instead of focusing on the MSR I tried to solve a similar problem: *From a set of points in* $\mathbb{R}^2$ *find the minimum spanning rectangle than contains all the points*. So far, S represents a set of circles and C the set of points a the circle. As part of this relaxation, we will use S to represent the set of all points of the problem. This is to prevent the use of $(x, y) \in C, \forall C \in S$ and instead use $(x, y) \in S$. There is no need to make our lives unnecessarily harder. This problem would prevent me from dealing with the radius. Removing constraints is a technique called relaxation methods. Its jargon from mathematics. To me both problems are just as hard. I drew with pencil and paper points and rectangles to give myself examples. This subsection contains all the results obtained in attemting to solve the relaxed MSR problem. Not everything is used in the final implementation. In fact, most of the ideas never get used. Is it all for nothing? Edison figured out 1000 ways of not making a light bulb before

figuring out how to make one. What I did was to draw the best solution I could think of and assume that it was the MSR. From the examples that I drew, I tried to find similarities betweeen all the examples and attempted to prove that this similarities were true. Consequently, in the following paragraphs I use the term MSR to refer to a candidate MSR rather than the true MSR. One of the first patterns that I noticed was that the longest side of the MSR tended to be the length of the two points furthest apart from each other. This can be stated in tow different ways.

Proposition 1. *The base of the minimum spanning rectangle equals the distance between the two points that are furthest apart.*

Remark. By base I refer to the longest side of the MSR

Proposition 2. *The two points that are the furthest apart are at opposite sides of the minimum spanning rectangle.*

This is important because if we prove that these propositions are true, it is straightforward to compute which two points are furthest apart. So we would know that opposing sides have to contain those points. I tried and tried, but was unable to prove this. I quickly got a concern. To find which points are furthest apart, it would have to compute and compare $\binom{n}{2}$ distances. This complexity grows factorial, making the problem intractable. Two solutions popped into my head:

1. discriminate/prune points that are clearly not relevant to the problem
2. compute distances with respect to the mean. This requires the computation and comparison of n distances as opposed to $\binom{n}{2}$.

The first choice is easy to state, but not easy to solve. The points that matter are the ones that make contact with the MSR. For us it is easy to tell the difference, but for a computer this is an array of numbers. From the array, how do you know which points will touch the MSR? At this points we don't even know how to find the MSR. Thus, we cannot construct a solution to this problem. Often, in the process of solving a problem you usually create more problems that need to be solved first. At this point I decided to relax the problem even further: *what geometric figure spanns the points most efficiently?* an irregular polygon (see Figure 8).

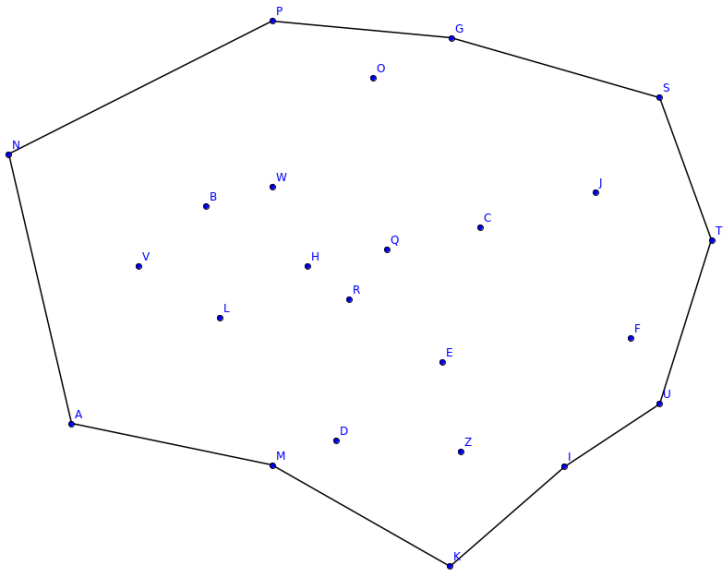

Figure 7: Example of a Minimum Spanning Irregular Polygon

From now on we will refer to the points on the polygon as the boundary points and all other points as interior points. Clearly, the interior points are not going to touch the MSR. Therefore, at least four boundary points are going to be on the MSR, but we do not know which. We can regard the set of boundary points from the MSIP as the solution space for the relaxed MSR problem. Even though it is easy for us to tell which points are boundary points and which are interior points, to a computer these are nothing more than floating point numbers stored in RAM. *How do you know that a point is a boundary point?* This one is a tricky one. I drew a couple of examples. This one is the one that lead me to the result: INSERT IMAGE. The concavity gave it away. I felt I needed to exclude it, but I did not know why right away. Then I noticed: *for every line connecting two neighboring boundary points, all other points are either above the line or below the line*. This is a good example of how properties allows us to create algorithms. This property, in and of itself, represents a simple algorithm that answers the question: *is this pair of points boundary points?*. We can check out two at a time. If this is not the case, then one is a boundary point or none is. In the second case, we are done. In the first case we would need to figure out an algorithm to determine which is a boundary point. This would in a sense require to input this two points into the same algorithm used so far - a recursion. Or we could just ignore it and choose other random points. But that is a bad idea, because there is no way of pruning points for sure. Thus recursion is the way to go. Thus we can delete points one at a time? Perhaps two at a time? Is there really no other way to prune the points? Notice how so far we are building, step by step, properties and definitions that are helping us find the solution to the problem. There is one drawback behind this algorithm: we would need to compute $\binom{n}{2}$ lines and test whether all other points are above or below it. This defeats the purpose of the relaxation: it is just as hard to find the minimum spanning irregular polygon as it is to find the minimum spanning rectangle (under our current theory).

Nevertheless, the idea of the MSIP and boundary points is still useful. Clearly, the points that are furthest apart have to belong to the set of boundary points. We still need to prove this, so for now it is still a proposition.

Proposition 3. *The points in S that are furthest apart are elements of the set of boundary points.*

The second idea is more appealing since it results in a drastic computation reduction. If two points were

the furthest apart from each other, it appears reasonable to think that they would be the furthest apart from the mean

Proposition 4. *The first and second largest distances from the mean are the two points furthest apart.*

I tried it, found a counter example, threw a tantrum, and cried. But the counter example does not take into account the property of mean point. duh! I think I can still construct a counterexample.

After attempting to prove and disprove this proposition I noticed that the most promising rectangles had the points furthest apart in their corners.

Proposition 5. *The minimum spanning rectangle has the points that are furthest apart at two opposing corners.*

This points also form the diagonal of the MSR. This implies that the longer the diagonal the smaller the area.

Proposition 6. *For any two spanning rectangles* A *and* B*,* $Area(A) > Area(B)$ *if* $Diagonal(A) < Diagonal(B)$.

Both turned out to be false. I was able to find a counterexample for each.

Another thing that I noticed is that all SRs had one side with two points.

Proposition 7. *The minimum spanning rectangle has at least one side with two points.*

Whenever you make a proposition it is important to include as many cases as possible. We seek for a side to have 2 points, but why not 3 or 4? It could also be the case that other sides have many points in them. There is no reason to restrict, that is the reason why 'at least' is used.

This one has a lot of variability. To prove it we would have to prove that every SR that has only one point on each side has a larger are than every SR that has at least two points in one of its sides. What makes this so challenging is that the areas of this rectangles are data dependent. It could be the case that for some sets of points the SRs with at least two points in one side is smaller than the other SRs, but perhaps there are sets where this is not true. We would have to consider the set of all possible inputs. This abstract set is hard to handle. If that is the case, then we could construct a counterexample. We only need to find one and attempt to generalize it. If its only one we could treat it as a singularity. Even if its only one, all of this sounds like a lot of work. So I did not bother.

I did noticed that proposition 5 and proposition 7 were related. In the examples I had there was always an SR with the point furthest apart in a corner and one of the sides that contained it had another point.

Proposition 8. *One of the points in proposition 7 is one of the two points that are furthest apart.*

Proposition 5 and 8 are related. What if Proposition 5 is false? Then neither of the furthest points is on the corners or only one is in one corner. Regardless, Proposition 8 could still be true; this proposition is independent from the validity of Proposition 5. Proposition 5 is false; it does not hold always. There are two points that are furthest apart. One of them is at a corner, but there is no way to know which until the area is computed. A corner point is contained in two lines. We need to consider all candidates: each point has two candidates and there are two points. Therefore, there is a total of four candidates.

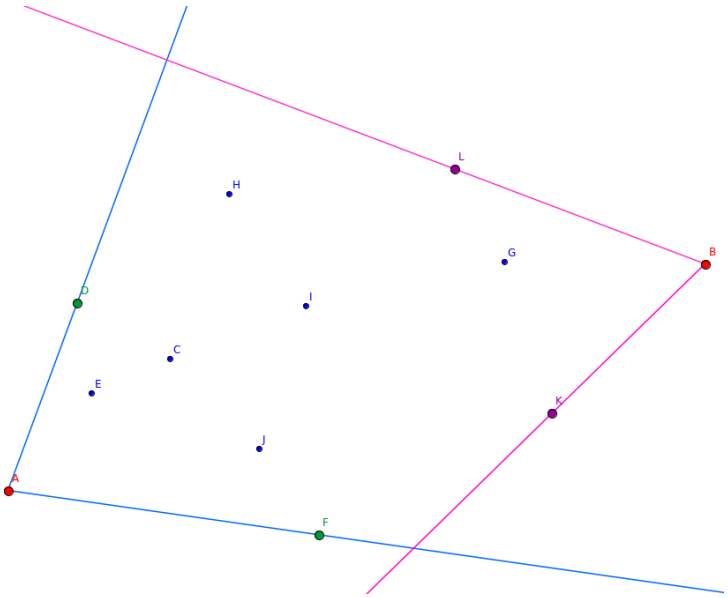

Figure 8: There are 4 candidate lines

This had an immediate consequence: *If we consider only boundary points, for both points that are furthest apart each has one closest neighbor to the left and one to the right*(see Figure 9). If Proposition 8 is true then at least one of this neighbors could be the second point on the side. If we connect these points by lines we see that it is more likely than not that if one line is in the MSR that the other three are not. If Proposition 8 is true, then for any S we would only need to check four cases. This is called constant complexity and it is the best type of complexity one could find. The challenge is to prove it. Nevertheless, this would be a solution to a relaxed problem, not the problem we want to solve. Whenever propositions appear to be true but are too hard to solve some researchers assume that propositions are true and continue working on finding a solution. Someone else might prove or disprove them after you finish your research. The key idea is to document your work in such a way that someone else can understand what you are thinking and what you are assuming and pick up the torch after you stop working on the problem.

Unlike the other propositions, this was not intuitive and I quickly developed properties/situations that could result in a counterexample. To prove Proposition 8 I would have to prove that there exists another point $p \notin S$ such that the area of the rectangle that has that point as a corner is smaller than the area of the rectangle that has one of the furthest points apart as a corner for all possible sets S. Clearly a major challenge. Choosing p to be too far away from the line connecting the furthest point from one of its closes neighbors is a poor choice since it adds more area to the rectangle. On the other hand if I choose p such that the rectangle rotates, the area that is added locally around p could be canceled by the area reduced on other sides of the rectangle(see Figure 10). It became clear to me that rotations of the rectangle around the points is the way to search for a minimum area; the most important feature to search for is the orientation of the rectangle. This got me thinking that there is at least one fixed point for all SRs; meaning, if you find the intersection of all possible SRs, then the intersection must have at least one point - probably a region in $\mathbb{R}^2$. It got me thinking: *if you create a third $p \notin S$ such that p is in between the two points featured in Proposition 8 to represent the corner of the rectangle, then there is this L shape that appears*. To continue the proof I had to compare the area achieved through Proposition 8 and the one to counter it. As I thought about how to compute the area of the counter, I realized that for any choice of L there is a unique complementary L that completes a SR. You can think of

moving the complementary L until it touches a point(see Figure 11). Then I realized that there is no need for an L shape: *this is also true for only one side*. So far the SR is an abstract object; we cannot construct it. Up to now we use a system of inequalities to represent the area and the point-magnitude-sign-vector object. Lets put them to good use now. Lets fix one side of the SR; meaning, we wish to find all the SRs that have that segment as one side. There is no way to tell right away the length of that side - it has undetermined length, namely it's a line(see Figure 12).

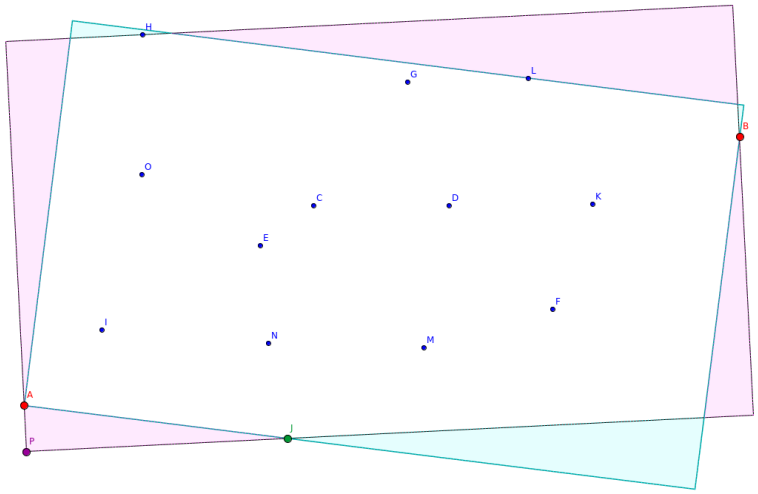

Figure 9: Areas that do not overlap are shaded

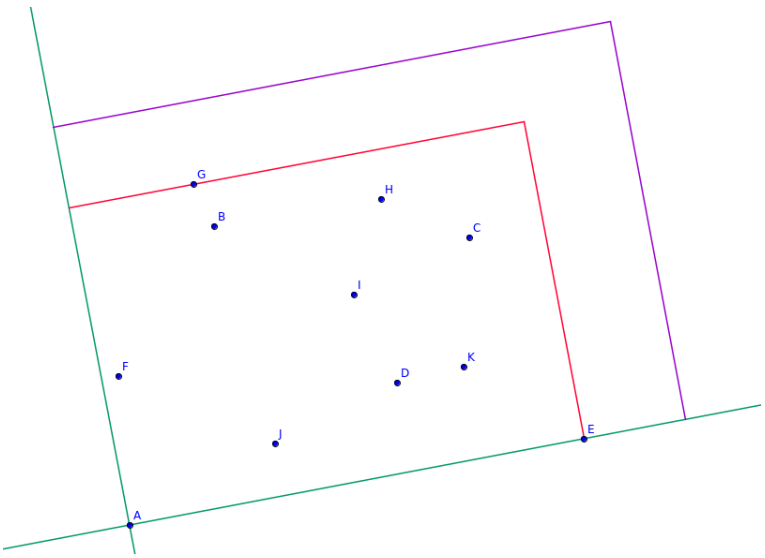

Figure 10: Example of L Parametrization

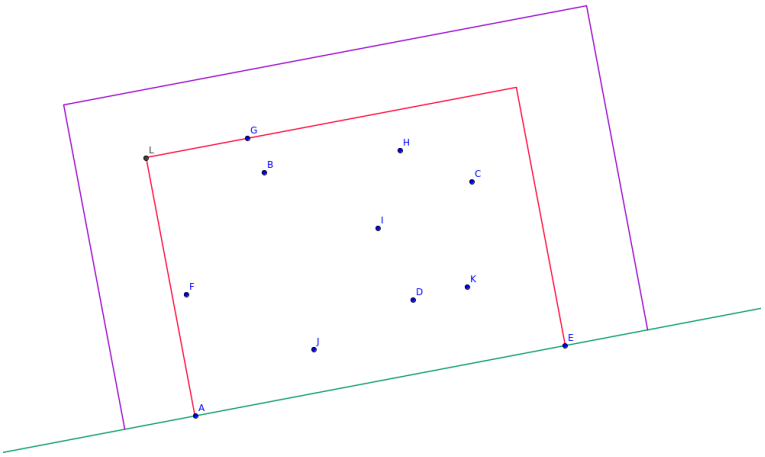

Figure 11: Example of Parametrization Using a Line

Proposition 9. *For any line drawn containing at least one point from S, there is a unique Spanning Rectangle that has the line running through one of its sides.*

Proof. The one that uses vectors and projections. This reduced to ... □

At this point I realized that these theorems could be extended to the problem of MSR with circles.

All of these propositions seem very promising, but they are useless unless they are proven. So far I have not found a polynomial-time algorithm that finds the two points furthest apart, nor have I proven propositions 5, 7, or 8. Still, I had the intuitive feeling that the two points further apart or the point furthest from the mean are at one fo the sides of the MSR for this relaxed problem.

I need to figure out where to place this one:

Proposition 10. *The point furthest away from the mean is a boundary point.*

The reason why it is so hard to construct a solution is that for each rotation of a rectangle or for each choice of line for all rectangles one side get smaller and the other gets larger. The only real way of finding the solutions is to compare the areas of the rectangles. At this point I began to suspect that the problem MSR with points was NP. If true, this means that the MSR with circles is NP as well. To prove this, we would have to reduce an NP problem into the MSR with points, which is no easy task.

Despite the fact that I was not able to solve the MSR problem for points I did get some valuable insight and I did get to prove some propositions. Once a proposition is proven it becomes a theorem.

THEOREMS GO HERE The main results from the investigations so far are:

As you work on theoretical stuff you usually have several ideas. Every time you have to make a choice whether to pursue an idea or to leave it. Some ideas are clearly silly, but it could be possible that they may teach you something important down the line. But time is a valuable resource and you should not squander it. You have to choose between making a depth search or a breadth search. The optimal strategy, as proposed in game theory, would be to use a probabilistic approach: play depth search with probability p and breadth search with probability $1 - p$. What value of p to choose...that is the dilemma.

4 The Solution Without Relaxation

Since I could not find enough theorems to construct a theory to build the spanning rectangle, I decided to pursue a dynamic programming approach. The idea is to find a base case and use theorems to improve a solution from the base case. In a sense, dynamic programming is a combination between a construction and a search approach. You can incorporate both methods into one. Thus, I began with the base case for two circles. What is the MSR for any two circles? As I drew some examples I noticed that there was a possibility for the solutions to exhibit symmetry. This is bad news: if true, the only way for this to be possible is for the mathematical description to have multiple solutions. A problem has multiple solutions whenever there is not enough constrains. In the best case scenario, this results in free variables (as in the case for solution vectors in linear systems). On the other hand, this often leads to situations where no conclusion can be drawn from the information given, so that, unlike a solution vector, there is no clear way of writing the answer. It is not that unusual for problems to have solutions that cannot be found, like the sum of reciprocal cubes.

The first thing to do is figure out how to represent any SR in this system of two circles. The circles could be of any size and be anywhere in the Cartesian plane, so lets think of all possibilities: the circles do not touch, the circles touch but one does not contain another, and one contains another. The last one is easy to solve. The solution is the square that contains the big circle. The second is complicated. I decided to solve the first one before I began attempting to solve the second one.

We start by contemplating all possibilities for the first case. It is of utmost importance to contemplate all possible cases. Sometimes, one algorithm can be used for more than one case. In the worst case scenario each case requires its own algorithm. Therefore, from the algorithm point of view, missing a case is not that bad. On the other hand, data structures are more sensitive. A good data structure is one that has the proper balance between convenience and information - as discussed previously about the best way to represent a rectangle. As a designer, you cannot design an algorithm for a case that you are not aware of. As a consequence, there will be a runtime or logic error when the input meets the missing case. It could be the case that the new case requires extra information, in which case you have to make a new function to compute that information, which is tedious. Having functions that are only for special cases makes your code hard to read. Special cases always make code hard to read, so don't blame yourself for this; instead focus on making good documentation for your code. Perhaps the case requires an extra data structure, in which case you objects now are split. Perhaps you could merge the objects into a new object, in which case you need to update every function that takes that object as input-tedious, prone to error, easiest way to break your code. Take my word for it, its best for you to think carefully about all possible cases before moving on. Abstraction always helps to achieve this. To me, the best way to account for all possible cases is to identify which variables can change. If the variables are discrete, you can group them into subsets with properties of interest. If the variables are continuous you should focus on intervals of interest. In this case, the variables of interest are two points in the Cartesian plane and two radii. All variables are continuous. As you work with algorithms you often generate an example and within the first 15 seconds you already have an idea of how to solve it. Often the first couple of ideas are wrong. With each mistake you gain insight that helps you find a true solution to the problem. Usually, the reason why these initial ideas work most of the time, but there are special cases when it doesn't. After a couple of examples and tries, I realized that there is a special case when both circles have the same radius. We can use this special case as a starting point in our theory. Special cases are a pain for coding, but they are a joy for theory because you can use them as a guide to develop a theory. You can always generalize them. Special cases area provide a minimum description of a problem: they can be regarded as a lower bound to the amount of information needed to model a problem. In the first case, the solution is simple. The MSR has one side the size of the diameter and the other size is the distance between the centers plus twice the radius. From this case we can develop a base case, a precursor formula for the area of an SR for the complex case where the radius are not equal. The formula for the area of the MSR

when there are two circles centered respectively at $C1 = (x_1, y_1)$ and $C2 = (x_2, y_2)$ that do not touch and the radius r of both are equal is: $(2r)(2r + \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2})$. PROOF NEEDED. For the second case, the first thing I noticed is that one side of the MSR is guaranteed to be the size of the largest diameter. PROOF NEEDED. As I tried to find the MSR for this case I realized that it is almost as if you can rotate the rectangle with respect to one circle to find all good candidates. This is achieved by choosing one circle as the parametrization circle and choosing an appropriate point in it as a parametrization point. Figure !!MISSING FIGURE shows this, where pp is the parametrization point. At this point, this became my solution space. Now we must define the area from the data structure we have. Once a pp is chosen, the other points of the SR must be calculated before we can compute the area. We see from the image provided that the SR has side $r_1 + r_2 + \vec{d} \cdot \hat{u}$, where $\hat{u}$ is pointing to the right of pp and $\vec{d}$ is the direction vector from $C1$ to $C2$, where $C1$ is the parametrization circle. The are then $A = (r_1 + r_2 + \vec{d} \cdot \hat{u})(r_1 + r_2 + \vec{d} \cdot \hat{u}_N)$, where $\hat{u}_N$ is perpendicular to $\hat{u}$. $\hat{u}$ is normal to $\vec{r}$.

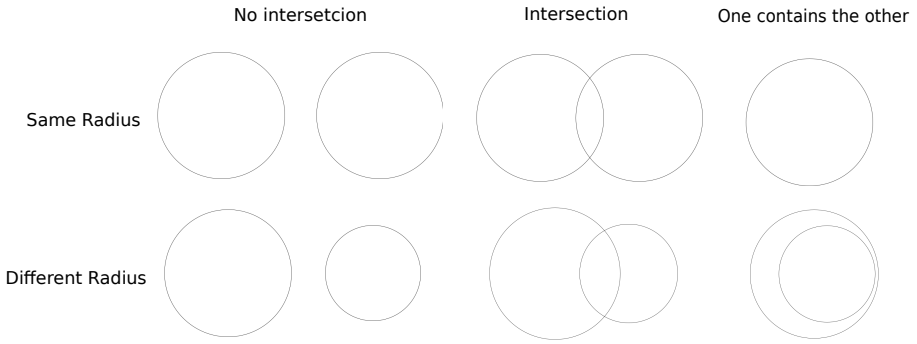

Figure 12: Circle Cases

Its time to wrap up into a formal structure. Any choice of $C1$ or $C2$ would serve as our origin and the line connecting them as the tangential axis. Its a custom I have from when I took a course in physics. Without loss of generality, if we choose a point p in $C1$ at an angle γ from its 'origin' with respect to axis t, then the area of the rectangle is given by $A = (r_1 + r_2 + |\vec{d} \cdot \hat{u}|)(r_1 + r_2 + |\vec{d} \cdot \hat{u}_N|)$, where $\vec{d}$ is the vector difference between $C1$ and $C2$, $\hat{u}$ is the unit vector of the parametrization line at p and $\hat{u}_N$ is the unit normal vector to $\hat{u}$. The absolute values account for the sign of the dot product. We wish to minimize this function. The only variable is $\hat{u}$. Note how this is a consequence of what was stated earlier, that a single line is required to parametrize a SR. Note how now we are representing that line with a single unit vector. The reason for this is that we can construct the line using this vector. We are systematically distilling the problem. We went from rectangles, to rectangles containing all circles, to rectangles containing all circles and having one touch each of its sides, to a line, to a unit vector. This is the power of theory, proper design, and data structures + algorithms. Should I define $\hat{u}$ in terms of t and n or x and y? The choice does not affect the area. What about defining $\hat{u}$ in therms of γ? γ requires t and n.

Any decrease in $\vec{d} \cdot \hat{u}$ results in an increase in $\vec{d} \cdot \hat{u}_N$. Could it be the case that for any two circles any SR has the same area? False by counterexample. In this particular case, for any choice of p, its leftmost orthogonal component is evident.

After trying some drawings, I noticed that it is always the case that the smallest rectangle ends up in one of the corners for the MSR. But this might be just often, rather than always the case.

analytical solutions are a single computation. Constructed solutions are an algorithm. those two are two different things.

THE IDEA OF THE SQUARES - complexity of generalization appears to be untractable.

THE IDEA OF PHYSICS
!!!!!!!!!!!!!!!!!!!!!!!!
So far this is what we know:

1. We assume that the point furthest from the mean is on one side of the MSR.
2. A line can be used to parametrize a SR
3. This is a search problem(as opposed to a construction problem)

Based on these facts we are going to have to compute and test candidates. Since this is a minimization problem, the best approach is to make an initial guess and then somehow improve it iteratively. Thanks to the assumption that the furthest point is on one side of the MSR there is no need to test various random starting points and then having to choose the best. Since the point furthest from the mean is going to be used to parametrize the line used to find a candidate SR, it is fitting to refer to this point as the parametrization point, and the circle containing that point the parametrization circle. From theorem !theorem incomplete-ref missing, we know that, with the exception of the point of contact, all points in the circle have to be above or below the line. This implies that the parametrization line has to be tangent to the parametrization circle at the parametrization point. From geometry, it is known that a line that is tangent to a circle is perpendicular to its radius. Sticking to naming conventions, the radius of the parametrization circle will be referred to as the parametrization radius. Recall that to prove theorem 2 we used the dot product to find a perpendicular vector. Since that description worked really well, I will continue to use vectors.

With this we can construct our first algorithm:

1. Find the parametrization point
2. Describe everything as a vector
3. Find parametrization line
4. Find an initial guess SR
5. Somehow improve the guess

As a brief discussion, it is ok to be informal every now and then for the sake of brevity but it is important to keep the right tools at hand. The sequential list given is a good tool for short algorithms, but as we seek to implement the algorithm more detail is going to be added. This leads to more complexity, at which point a sequential list is not going to be a good resource. I use flow diagrams to build software. When my dad learned programming, everyone in the CS business used flow diagrams. I was shocked when I took a course in algorithms this past fall when no flow diagrams were used. Somehow they fell out of style. I do not like the program-like algorithms. I feel that flow diagrams keep the objects abstract so that no implementation dependent constrains start creeping into the algorithm. I also feel that loose descriptions of the algorithm allow for more leeway for data structure design. It leaves more room for creativity.

5 From Theory to Algorithm

6 Implementation