Tlaquepaque, 1 de diciembre de 2016

Instituto Tecnológico y de Estudios Superiores de Occidente.

Departamento del Hábitat y Desarrollo Urbano.

Proyecto de aplicación profesional.

Algoritmos genéticos para la generación de conceptos de solución de refugios.

Ernesto Rizo Olvera

Javier Petersen Fuentes

Nayar Cuitláhuac Gutiérrez Astudillo

NOMBRE DEL PAP Y CLAVE:
Tecnología apropiada para la generación de sistemas constructivos.
1F04

ÍNDICE

Abstract

Uno de los principales problemas en zonas de desastres (como sismos, inundaciones, etc.) en los que se ven afectadas las viviendas es la selección y construcción de refugios. Debido a que muchos de los desastres naturales afectan las viviendas es necesario construir refugios de forma rápida para que los afectados tengan una vivienda temporal en lo que se reparan o reconstruyen las viviendas. Uno de los principales problemas de los refugios es la selección del refugio, ya que esto depende de varios factores como los materiales disponibles, la velocidad de construcción, el tipo de desastre, la mano de obra disponible, entre otros.

Hasta ahora, una de las maneras de realizar la selección de un refugio era consultar una base de datos, por ejemplo, la de la Cruz Roja. Para esto se realizaba la selección de manera manual, tomando en cuenta solo los criterios que se tengan a la mano o considere la persona encargada de la selección. Uno de los principales problemas con esta manera de selección es que es posible que la selección del refugio esté condicionada por la experiencia, gustos o conocimientos del encargado, lo que no garantiza que se consideren muchos de los factores importantes, comprometiendo la calidad de vida en el refugio. En este proyecto se buscó desarrollar una herramienta que ayude a proponer y seleccionar diferentes conceptos de soluciones para refugios que considere diferentes factores que pueden influenciar en la selección por medio de algoritmos evolutivos.

Palabras clave: refugios, algoritmos evolutivos.

Introducción

"La pérdida de las viviendas en un desastre impacta en muchas otras áreas. Por ejemplo, significa una pérdida súbita de la seguridad y la privacidad. Significa que la salud de la familia que vivía en ella se ve comprometida; y significa que la educación se puede posponer (o al menos interrumpir) y que la economía del vecindario puede colapsar. Además, la reconstrucción informal puede tener impactos negativos en el ambiente y los recursos, puede incrementar el costo de los materiales y la construcción y reducir los estándares de calidad" (Potangaroa, 2014).

Capítulo I. IDENTIFICACIÓN DEL ORIGEN DEL PROYECTO Y DE LOS INVOLUCRADOS

1.1 Antecedentes del proyecto

Actualmente en México, la selección de refiguios para zonas de desastre se realiza por medio de bases de datos, las cuales contienen diferentes propuestas de refugio y se selecciona en función de las variables conocidas (como los materiales disponibles, mano de obra, sistemas constructivos de la zona) y los conocimientos de la persona que los selecciona.

Este método de selección presenta algunos aspectos que pueden resultar en una selección poco óptima y afectar la calidad de vida en los refugios, ya que está condicionado a la información que tenga la persona encargada de la selección.

El hecho de realizar una mala selección del tipo de refugio puede presentar diferentes problemas como:

- Calidad de vida reducida por áreas insuficientes.
- Dificultad de construcción debido al desconocimiento del sistema constructivo.
- Alto costo energético por el transporte de materiales.
- Rechazo del refugio por parte de los habitantes.

Estos, entre otros problemas, son los que se pueden presentar debido a una mala selección o propuesta del sistema.

Hasta ahora, la Cruz Roja se ha encargado de recopilar información sobre diferentes propuestas para refugio, los materiales utilizados y las ventajas y desventajas que presenta cada uno de ellos. Con esto se ha ido generando una base de datos que se puede consultar y se utiliza para buscar el refugio o vivienda que más se adecue a las necesidades del momento.

En esta base de datos de pueden encontrar diversas propuestas y las desventajas que presentan, sus dimensiones y el sistema constructivo de cada una de ellas. Con esta base de datos de ha podido tener una mejor selección debido a que ya se tiene una lista con diferentes propuestas y la persona encargada de la selección puede revisarlas, por lo que ya no está condicionado a los conocimientos del encargado.

1.2 Identificación del problema. Problemática atendida

Como se mencionaba anteriormente, el problema de la selección de refugios y la generación de nuevas propuestas radica en la capacidad que tenga el encargado de realizarlo para poder conocer y evaluar todos los parámetros que pueden afectar a la construcción y uso de cada refugio en cada uno de los casos.

Algunos de los factores importantes que se deben tomar en cuenta son:

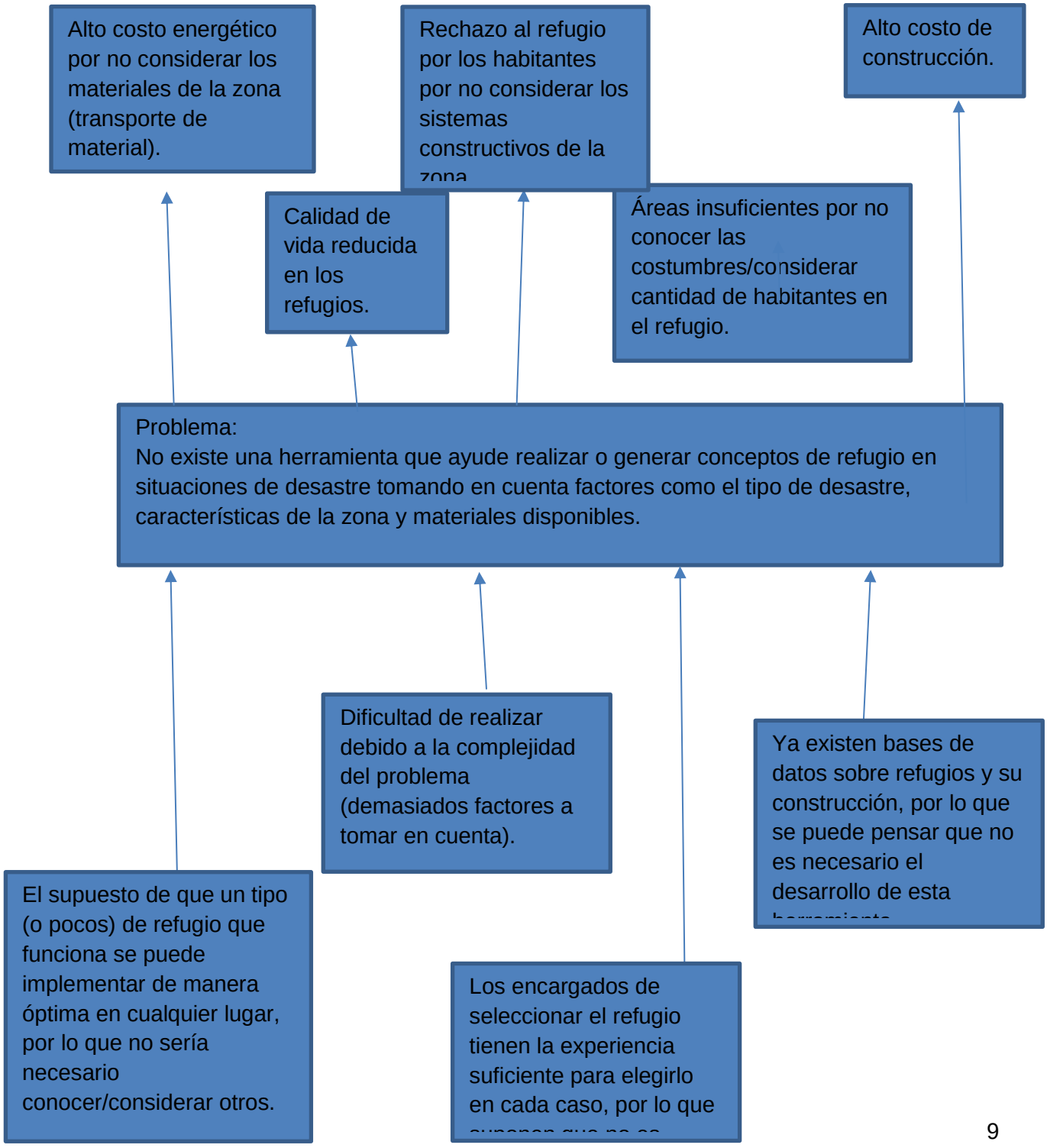
- Materiales disponibles en el momento: este es uno de los factores que suele influir de manera directa en la selección del refugio y sistema constructivo, ya que es necesario que los refugios en zonas de desastre se construyan de manera rápida para iniciar la recuperación de las viviendas y el hecho de traer materiales de otras zonas impacta de manera directa en el tiempo de construcción, por lo que se recomienda utilizar lo que se tenga disponible.
- Mano de obra disponible en la zona: la mano de obra disponible también determina el (o los) sistema constructivo que se puede utilizar, ya que suele ser muy tardado capacitar a las personas a construir con diferentes sistemas en poco tiempo.

- Tipo de constricciones en la zona: aunque suele ser un factor que no se suela considerar como importante, se ha dado el caso de que personas rechacen casas y refugios por no ser el sistema constructivo al que están acostumbrados. Este suele ser el caso de las viviendas de madera o materiales livianos en Jalisco, las cuales son vistas con “desconfianza” por las personas por no ser de mampostería, suponiendo que tienen una menor resistencia.
- Tipo de desastre: este es un factor que, a pesar de ser determinante en la selección del refugio, no suele ser un problema, ya que si suele ser considerado. Un ejemplo de esto es en una zona de inundaciones, donde se deben construir refugios suficientemente altos (o separados del suelo) para evitar inundaciones o humedades en el refugio.

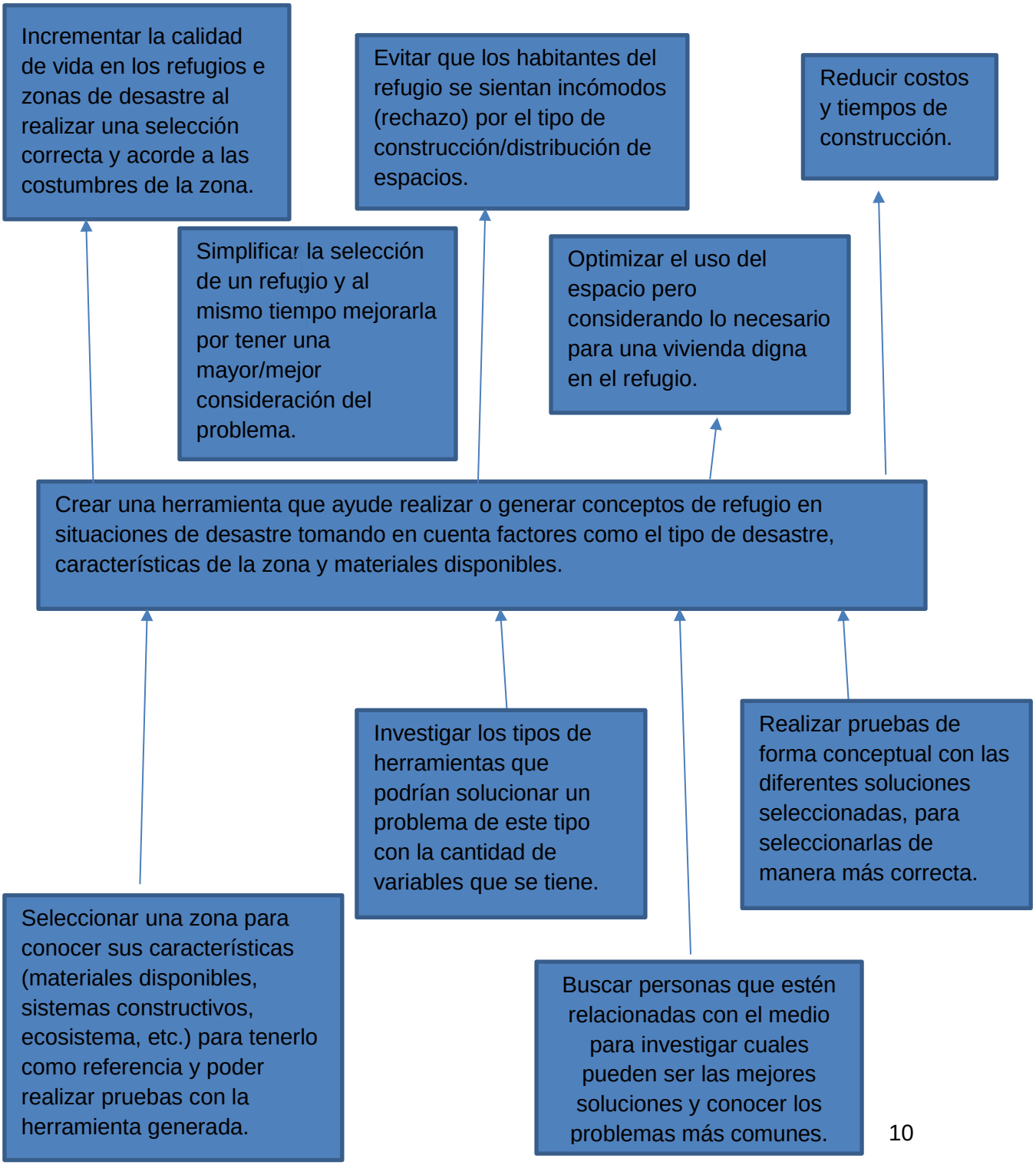
Por estos factores es necesario tener una herramienta que pueda ayudar a generar conceptos de solución y que pueda ayudar a seleccionar de manera conveniente y tomando en cuenta otros factores que pueden ser olvidados por las personas. Esta herramienta debería ser una herramienta que no sea determinista y que permita búsquedas de conceptos de solución que pudieran ser más óptimos que los que se puedan tener en una base de datos.

Con los problemas anteriores de relizó el enunciado del problema y los árboles de problemas y de medios y fines; esto fue con el fin de ayudar a tener una solución bien acotada y definida y conocer los medios para realizarla. Con estos, se puede tener una descripción gráfica del mismo.

Árbol de problemas.



Árbol de medios y fines.



1.3 Identificación de la(s) organización(es) o actores que influyen o son beneficiarios del proyecto

Dado que el proyecto pretende revisar, diseñar y proponer una solución al problema del refugio planteado en la base de datos de la Cruz Roja con el marco recíproco como cubierta, la misma organización es la que se vería beneficiada de manera directa ya que se podría incluir de manera actualizada una nueva propuesta con el mismo sistema constructivo pero que cumpla con la resistencia solicitante del refugio.

De manera indirecta, se podrían ver influenciadas y afectadas las personas que utilicen e implementen el refugio en caso de que se incluya en la base de datos y se llegue a requerir su implementación en algún caso.

Capítulo II. MARCO CONCEPTUAL O TEÓRICO DEL PROYECTO

Dentro de las ciencias computacionales, un algoritmo evolutivo es un tipo de heurística que intenta emular las propiedades de la evolución en un algoritmo cuyo fin es la optimización de una función. Es un subconjunto de los algoritmos de inteligencia artificial.

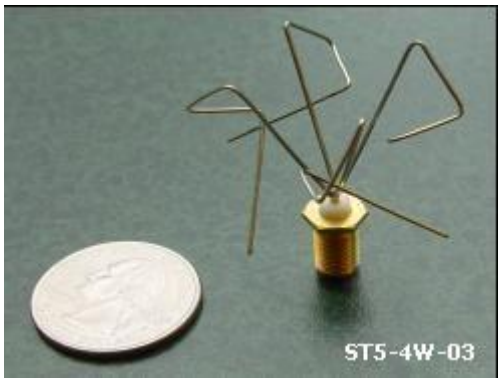
Dentro de los algoritmos evolutivos existen diferentes formas de implementación que buscan emular diferentes cualidades de la evolución. Un tipo de estos algoritmos son los algoritmos genéticos; los cuales intentan utilizar las características de un elemento (genes) y realizar la cruza de elementos, reproducción y mutación de los mismos para llegar a una solución que se aproxime a la óptima. El nombre de estos algoritmos proviene de la similitud del comportamiento de los genes en la evolución de una especie.

Este tipo de heurísticas presenta ciertas ventajas ya que permite optimizar funciones de problemas que pueden tener una solución analítica bastante complicada. Esto se debe a que la manera de evaluación del problema depende de una función (función objetivo) que mapea entre las propiedades de los elementos (población) y un valor que puede ser evaluado para fines comparativos entre los elementos; la función objetivo debe representar la (o las) cualidades de el elemento que se desea optimizar en un valor numérico.

Este tipo de algoritmos son particularmente útiles en problemas cuya solución analítica es muy costosa en cuestión computacional o es muy difícil de implementar. Muchos problemas caen en este caso y en muchos de ellos la solución analítica no es necesaria y una aproximación es más que suficiente (Daniel Dyer, 2010).

La principal ventaja de los algoritmos evolutivos radica en la facilidad de realizar una implementación que pueda iniciar con condiciones arbitrarias, lo que permite realizar búsquedas de soluciones óptimas en espacios muy amplios, además de presentar soluciones que (si el algoritmo está bien planteado) no dependan de

condiciones humanas. Un ejemplo de estos diseños es el de la antena desarrollada por la NASA en" Evolvable *Systems Group"*.



NASA *Evolvable Antenna*

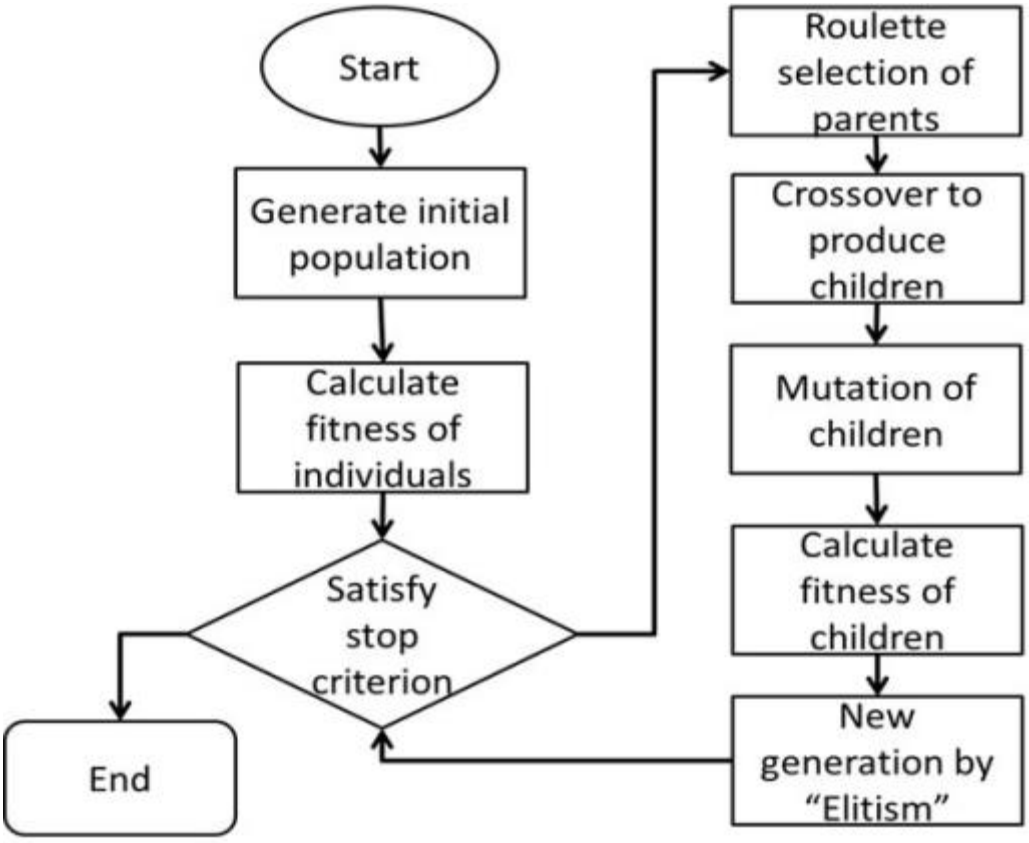
Historia de los algoritmos genéticos.

Los algoritmos genéticos comenzaron a programarse a finales de los años 1950 por biólogos evolutivos que trataban de realizar un modelo de la evolución (Marczyk, 2004). En estos primeros programas se buscaba emular la evolución, más que realizar la optimización de una función. Uno de los primeros cambios importantes se dio en 1966, cuando se propuso la técnica de programación evolutiva. Este método utilizaba máquinas de estado finito para representar a cada uno de los candidatos y relizaba mutaciones sobre los mismos, guardando los mejores candidatos.

Con estas bases, se fueron implementado diferentes tipos de algoritmos que poco a poco fueron implementando más funciones que se asemejan al comportamiento de la evolución. Algunas de estas funciones son la cruza, la mutación y la selección por elitismo (la que trata de emular la selección natural en función de lo apto del elemento).

Ventajas de los algoritmos genéticos.

Una de las principales ventajas de estos algoritmos es que su implementación es relativamente sencilla si se tiene una forma de evaluar, entre dos candidatos, cual es el mejor. Esto se debe a que la impementación de estos es relativamente simple y, a grandes razgos, se sigue la misma lógica para diferentes problemas. A continuación, se muestra un diagrama de flujo de un algoritmo genético.



Ejemplo de diagrama de flujo (Kang, Chuang, 2011).

En el diagrama anterior se pueden observar los pasos que se siguen en un algoritmo evolutivo, en este caso para la solución de un modelo intracelular, pero los

pasos a seguir son similares, con una función objetivo diferente y con penalizaciones y elitismo que se pueden implementar para cada problema que se desee solucionar.

Marcos recíprocos.

Un marco recíproco es una estructura tridimensional que consiste en elementos que se soportan de manera mutua en un circuito cerrado; la parte interna de una barra es soportada por la barra adyacente a ella (Popovic, 2008).

Los marcos recíprocos son estructuras que por sus características geométricas son buenos candidatos para la implementación de un algoritmo evolutivo para generar geometrías válidas. Esto de debe a que en algún punto todas las barras terminan cargándose entre sí. Por este motivo la posición de una barra depende de la posición del resto de las barras las cuales, a su vez, dependen de las demás. Esta condición provoca que la solución analítica a la generación de este tipo de geometrías no exista.

A pesar de que la solución analítica no se pueda encontrar, una ventaja de estas estructuras es que la forma de evaluación para conocer si un elemento es mejor candidato que otro no es difícil y se realiza considerando la suma de todos lo errores (error en la distancia) que existen en la posición de una barra respecto a la barra que la soporta.

Por este motivo se decidió implementar un algoritmo genético que pudiera generar configuraciones válidas que eventualmente pudieran ser implementadas como soluciones reales para cubiertas ligeras.

Capítulo III. DISEÑO DE PROPUESTA DE MEJORA

3.1 Enunciado del proyecto

Este proyecto pretende generar una herramienta que sirva de apoyo en la selección de refugios y que sea capaz de generar conceptos de refugio. Un concepto de refugio es una solución que puede ser implementada y que funciona como una idea que se pueda desarrollar. Con esta herramienta se podrían generar soluciones que consideren más factores que influyen en la construcción del refugio y que después puedan ser revisadas por alguna persona con conocimiento de la problemática para que el concepto de solución de lleve a la implementación y ejecución del mismo.

3.2 Objetivos y alcances del Proyecto

El objetivo principal del proyecto es llegar a implementar un algoritmo genético que pueda revisar diferentes parámetros de un refugio, tomando en cuenta los materiales disponibles, características culturales de la zona y características climáticas.

Para llegar a este objetivo se busca realizar en diferentes partes. La parte inicial es implementar un algoritmo genético para la búsqueda de geometrías válidas para la generación de marcos recíprocos. Con este algoritmo se busca diseñar un marco recíproco que se pueda implementar como cubierta ligera para un refugio y revisar el motivo de falla de la cubierta del refugio propuesto por la Cruz Roja.

Con la revisión del marco propuesto por la Cruz Roja se busca mejorar el refugio propuesto y tener una solución válida para las características, dimensiones y materiales necesarios para implementar ese refugio, utilizando como herramienta para generar y obtener la geometría con el algoritmo genético propuesto.

A pesar de no ser parte del alcance de este proyecto, se busca que este algoritmo se pueda utilizar e implementar para un algoritmo más completo, que

pueda considerar cuestiones arquitectónicas y el comportamiento estructural de los elementos, por lo que es necesario que la programación de realice de la manera más abierta posible y que el código se implemente de manera que se pueda reutilizar el código en un algoritmo más completo.

Además del algoritmo, el equipo se inscribió al concurso “Bright Optimizer”, en el que se publica un problema de una armadura y se debe implementar un método para la optimización estructural de la misma, utilizando MatLab y cumpliendo con las condiciones del programa.

3.3 Metodología

Antes de iniciar a realizar la implementación, se trató de buscar la mejor manera de realizar un programa que solucione este tipo de problemas de manera óptima. Para esto se investigaron diferentes metodologías con algoritmos evolutivos y sus cualidades y desventajas para cada tipo de problema. Se optó por implementar el método propuesto en el artículo “Formfinding of Nexorades Using Genetic Algorithms” por O. Baverel y H. Nooshin.

Para poder generar un programa que sea posible ampliar y reutilizar de forma simple y eficaz es necesario elegir de manera correcta el lenguaje de programación, ya que eso determinará algunos factores como:

- La velocidad de ejecución.
- El tiempo de implementación.
- La facilidad de reutilización de código.
- La cantidad de librerías o funciones a desarrollar.

Para esto es necesario evaluar el tipo de algoritmo a implementar (en este caso el algoritmo propuesto en el artículo) y las funciones que será necesario implementar para su correcta ejecución y funcionamiento.

3.4 Cronograma o plan de trabajo

Tareas a realizar y fechas límite:

- Seleccionar el algoritmo a implementar: 2/10/2016
- Seleccionar el lenguaje de programación: 10/10/2016
- Hacer un bosquejo de funciones más importantes: 12/10/2016
- Implementar el algoritmo:12/11/2016
- Revisar problemas (si es que existen): 18/11/2016
- Desarrollar una forma de visualización de los elementos: 20/11/2016
- Proponer una solución geométrica para el problema: 21/11/2016
- Hacer un modelo a escala de la configuración propuesta 25/11/2016
- Probar el elemento hasta la falla: 28/11/2016
- Presentación del proyecto: 01/12/2016

Capítulo IV. DESARROLLO DE PROPUESTA DE MEJORA

4.1 Memoria Descriptiva del Proyecto

El algoritmo consta de varias partes principales, estas son:

- Generación de población inicial.
- Selección de padres.
- Cruza.
- Mutación.
- Evaluación de la función objetivo.
- Selección por eltismo.

Generación de población inicial.

La población inicial se genera de manera aleatoria en un espacio de búsqueda determinado. Esto se realiza a partir de una configuración básica (una malla en tres dimensiones) que delimita la configuración que puede tener el algoritmo. El motivo de esto es que es necesario conocer las coordenadas iniciales de las barras en la inicialización del algoritmo y de esta manera se acota el espacio de búsqueda. De otra manera, sería necesario revisar muchas configuraciones iniciales de manera aleatoria, lo que haría que el algoritmo fuera extremadamente lento.

Para la generación inicial se crea un cubo (a partir de la coordenada central de la configuración básica) en donde se generan el resto de las coordenadas

requeridas por en número de barras de la configuración básica. Este cubo acota aún más el espacio de búsqueda, permitiendo una rápida convergencia del algoritmo.

Selección de padres y cruza.

La selección de los elementos para la reproducción se realiza de manera arbitraria con todos los elementos de la generación en la que se encuentre. Para esto, se seleccionan 4 indices aleatorios y se toman lo 4 núcleos que se encuentren en esos índices. La selección de padres de realiza evaluando las funciones objetivo de estos cuatro elementos y tomando las dos mejores.

Teniendo seleccionados a los elementos padres, se copian en la siguiente generación y se realiza la cruza de ambos elementos, resultando en otros dos elementos.

La cruza se realiza tomando los genes de los padres e intercambiándolos en los hijos en función de un valor aleatorio con el cual se decide si de intercambian o no. En este caso, los genes son las coordenadas de cada una de las barras, entonces al cambiar de forma aleatoria algunos de los puntos finales de cada una de las barras entre los dos elementos se tienen dos nuevos elementos cuyas propiedades son una mezcla de las propiedades geométricas de los padres. Esto da lugar a nuevos elementos que pueden o no ser mejores candidatos de solución.

Mutación.

Teniendo la copia de los padres en la nueva generación y habiendo realizado la cruza de los elementos, se procede a realizar la mutación de cada uno de los elementos de la nueva generación. Para esto, se itera sobre cada uno de los elementos de la última generación y en función de una tasa de mutación se selecciona si el elemento se muta o no.

La tasa de mutación en este caso se tomó de 7%, es decir, cada núcleo tiene una probabilidad de 0.07 de mutar. En caso de que el elemento pase por esta tasa de mutación, se pasa la geometría del elemento por un proceso de mutación en el cual se vuelve a limitar la posibilidad de cambio para cada una de las barras. Esta segunda parte limita a que cada una de las barras mute; de esta manera se limita la posibilidad de que muten todos los elementos de un nucleo al mismo tiempo.

La función de mutación, en particular la forma de limitar la tasa interna de mutación, fue uno de los cambios que se propusieron a loas funciones del algoritmo original. Realizando la mutación de esta manera se puede incrementar la posibilidad de convergencia del algoritmo. Esto se debe a que, al incrementar el número de barras, si se modifican todas las coordenadas de los elementos, la propabilidad de que todas las mutaciones sean favorables se reducen.

A continuación, se muestran dos elementos que se generaron con la misma configuración básica, mismo número de generaciones y mismo valor de lambda, pero realizando la mutación con la nueva propuesta de función de mutación. La primera imagen muestra el elemento con la función previa.

En los elementos anteriores se puede observar que, al modificar la función de mutación con esta propuesta, se puede incrementar la velocidad de convergencia de manera notoria y la calidad de los elementos generados. Para comprobar lo anterior, se dejó correr el algoritmo hasta 100,000 generaciones con la primera función y el resultado fue mucho peor que utilizando la nueva función con 2000 iteraciones.

Evaluación de la función objetivo y elitismo.

Una vez realizada la mutación, se procede a iterar sobre el arreglo de la nueva generación y a calcular la función objetivo de cada uno de los elementos (ver siguiente sección para explicación de dicha función).

Después de evaluar la función objetivo de los elementos de procede a concatenar en un arreglo la generación previa y la actual, generando un arreglo de 200 elementos. Esto se realiza para garantizar que se evalúen las copias originales de los padres y permite mutar a los padres y los hijos, generando una mayor diversidad en los elementos evaluados.

El elitismo es un porcentaje que se toma para la selección de la nueva generación. Se selecciona un porgentaje de los elementos totales del arreglo ordenado de la generación actual y se copian de manera directa a la siguiente generación. El resto de los elementos se toman de manera arbitraria hasta llegar a la cantidad de elementos necesarios para la siguiente generación. Esto permite tener una convergencia más rápida que si se tomaran todos los elementos de manera arbitraria. Aunque el hecho de tomar los elementos de manera arbitraria permitiría tener un método que en teoría podría estar menos condicionado, el tomar un porcentaje de manera arbitraria (en el caso de este algoritmo se tomó un valor de 0.4, pero se puede modificar) permite jugar con lo arbitratrio de la selección, mientras se garantiza que el altoritmo si tenga cierta velocidad de convergencia al seleccionar algunos de los mejores candidatos.

4.3 Descripción de la implementación

Para la implementación del algoritmo fue necesario implementar las siguientes clases:

- VentanaInicial.
- AlgoritmoGen.
- Nucleo.
- Geometria.
- CargarArchivos.
- GenerarArchivos.
- Objetivo.
- Clone.

Debido a que el código de cada clase resultó ser bastante largo, en este reporte se muestran tres de las clases más importantes (en la implementación de los marcos recíprocos, ya que las otras se encargan de manejar archivos, crearlos, almacenas arreglos para la generación de la geometría, etc.). A continuación, se muestra la clase AlgoritmoGen; esta es la clase que contiene el método principal, el encargado de llamar las funciones de mutación y reproducción y realizar el ciclo hasta que se cumpla el número de generaciones máximo.

```
package al_gen;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class algoritmoGen {
        private Generacion[] gen = new Generacion[2];
```

```
public algoritmoGen() {
}



public void al_gen() throws Exception {
        List<Nucleo> listaNucleos = new ArrayList<Nucleo>();
        int[] indicesPadres = new int[4];
        gen[0] = new Generacion(1);
        gen[1] = new Generacion(1);
        for (int i = 0; i < Constantes.NUM_GENERACIONES; i++) {
                gen[0].actualizarNodos();
                gen[0].calcularObjetivo();
                gen[0].zeroIndicesL();
                gen[1].zeroIndicesL();
                gen[1].nucleosZero();

                for (int j = 0; j < 25; j++) {
                        indicesPadres = gen[0].emparejamiento();
                        gen[0].cruza(gen[1], indicesPadres);
                }
                for (int k = 0; k < gen[0].arregloNucleos.length; k++) {
                        gen[1].arregloNucleos[k].calcularLongitudElem();
                        gen[1].arregloNucleos[k].calcularLongProm();
                        gen[1].arregloNucleos[k].normalizarLongitudes();
                        gen[1].arregloNucleos[k].actualizarNodo();
                }
                gen[1].calcularObjetivo();
                gen[0].calcularObjetivo();
                gen[1].mutacion();
                // Crear una lista que en la que se pueda ordenar
                // y ordenar los elementos. La selección de los candidatos
                // está determinada por el porcentaje de elitismo.
                for (int j = 0; j < Constantes.NUM_CANDIDATOS; j++) {
                        listaNucleos.add(gen[0].arregloNucleos[j]);
                        listaNucleos.add(gen[1].arregloNucleos[j]);
                }
                Collections.sort(listaNucleos, new Comparator<Nucleo>() {
```

```
                    @Override
                    public int compare(Nucleo n1, Nucleo n2) {
                        return n2.compareTo(n1);
                    }
                });
                for (int j = 0; j < gen[0].arregloNucleos.length; j++) {
                    if (j < (gen[0].arregloNucleos.length) * Constantes.ELITISMO) {
                        gen[0].arregloNucleos[j] = (Nucleo)
                        (Clone.copiarObjeto(listaNucleos.get(j)));
                    } else {
                        gen[0].arregloNucleos[j] = (Nucleo) (Clone
                                .copiarObjeto(listaNucleos.get((int)
                        (Constantes.NUM_CANDIDATOS * Math.random())));
                    }
                }
                listaNucleos.removeAll(listaNucleos);
            }
            gen[0].arregloNucleos[0].guardarArchivo("final");
            gen[0].arregloNucleos[0].mostrarInfoNucleo();
            gen[0].arregloNucleos[0].mostrarNucleo();
        }
}
```

En un algoritmo genético, la función objetivo es la encargada de mapear la geometría de la estructura (en este caso un marco) a un valor numérico que se pueda comparar para saber cuál de dos opciones de geometrías es mejor que la otra. En este caso, la función objetivo calcula la suma de las distancias del punto

actual de la barra al punto en donde debería estar respecto a la barra que la soporta. La siguiente imagen muestra un diagrama de la función.

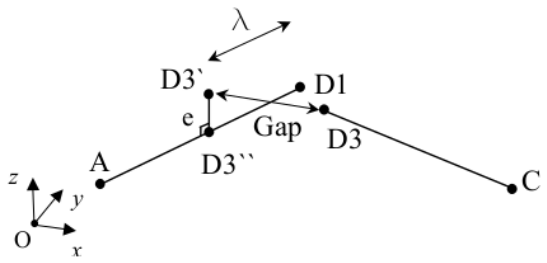


Figure 13. Positions of a pair of nexors

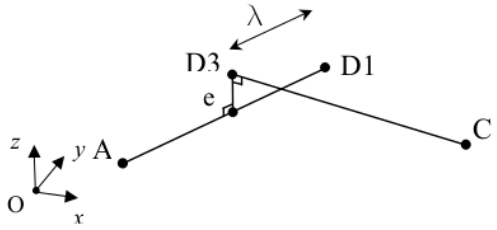


Figure 14. Acceptable positions of nexors

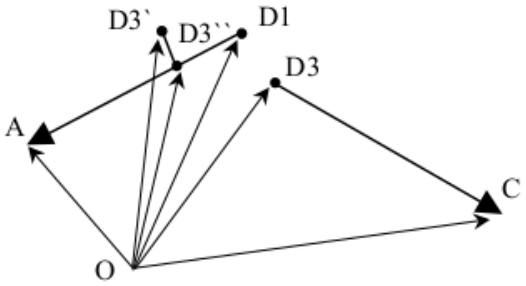


Figure 15. Vectors associated with nexors

La clase Objetivo es la encargada de crear, almacenar y operar los vectores necesarios para conocer el valor de la función objetivo. Debido a que el método requería el manejo de vectores para la función objetivo, también se implementó una clase Vector, que se encarga del manejo y operaciones (como producto cruz,

producto punto, etc.) y la clase Objetivo hace el uso de la misma para calcular la función objetivo.

```
package al_gen;

import java.io.Serializable;

public class Objetivo implements Serializable {
	private static final long serialVersionUID = 1L;
	public double fObjetivo;
	protected Vector[] arregloVectores;
	protected double[] distancias;

	public Objetivo(int numBarras) {
		this.distancias = new double[numBarras];
		// Se requieren 9 vectores para calcular la distancia
		// entre ambas barras. Se calcula para dos barras de
		// forma simultanea, luego se elimina el arreglo y se
		// utiliza para las siguientes dos.
		this.arregloVectores = new Vector[9];
		for (int i = 0; i < arregloVectores.length; i++) {
			arregloVectores[i] = new Vector();
		}
	}

	public Objetivo clone(int numBarras) {
		Objetivo nvoObj = new Objetivo(numBarras);
		nvoObj.fObjetivo = this.fObjetivo;
		for (int i = 0; i < this.distancias.length; i++) {
			nvoObj.distancias[i] = this.distancias[i];
		}
		for (int i = 0; i < 9; i++) {
			nvoObj.arregloVectores[i] = this.arregloVectores[i].clone();
		}
		return nvoObj;
	}
```

```
public void calcularObjetivo(Nucleo nucleo) {
        // Calcula la función objetivo en función de las distancias que existen
        // entre cada par de barras (al punto de la excentricidad de la barra
        // superior).
        // El orden del arregloVectores es: vOC, vOD3, vD3C, vOA, vOD1, vD1A,
        // vOD3pp, vOD3p, vU.
        for (int i = 0; i < nucleo.numeroBarras; i++) {
                // Se debe considerar el caso especial cuando
                i==nucleo.numeroBarras
                // ya que es necesario calcular la separación entre todas las
                // barras; si no se considera, faltará la separación entre la primera y
                // última barra.
                if (i != (nucleo.numeroBarras - 1)) {

                        // 0C, Punto externo de la barra 1
                        arregloVectores[0] = new Vector(nucleo.coordX[2 * i],
                        nucleo.coordY[2 * i], nucleo.coordZ[2 * i]);
                        // OD3, punto interno de la barra 1
                        arregloVectores[1] = new Vector(nucleo.coordX[(2 * i) + 1],
                        nucleo.coordY[(2 * i) + 1],
                                        nucleo.coordZ[(2 * i) + 1]);
                        // D3C 0c-0d3
                        arregloVectores[2] = Vector.restaVectores(arregloVectores[0],
                        arregloVectores[1]);
                        // OA, Punto externo de la barra 2
                        arregloVectores[3] = new Vector(nucleo.coordX[(2 * i) + 2],
                        nucleo.coordY[(2 * i) + 2],
                                        nucleo.coordZ[(2 * i) + 2]);
                        // OD1 punto interno de la barra 2
                        arregloVectores[4] = new Vector(nucleo.coordX[(2 * i) + 3],
                        nucleo.coordY[(2 * i) + 3],
                                        nucleo.coordZ[(2 * i) + 3]);
                        // D1A
                        arregloVectores[5] = Vector.restaVectores(arregloVectores[3],
                        arregloVectores[4]);
                        // OD3pp=OD1+l/l*D1A
```

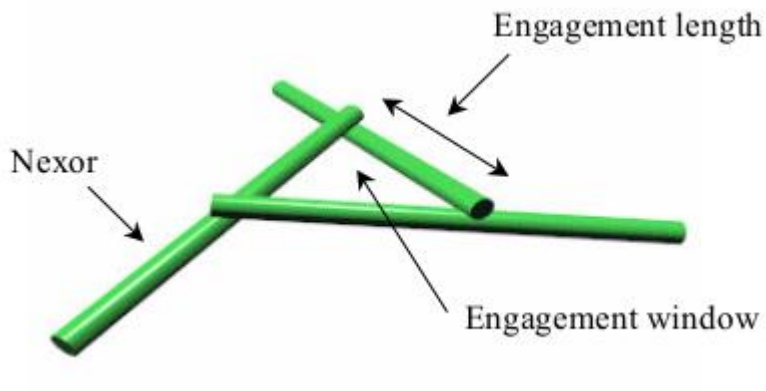
```
                arregloVectores[6] =
Vector.sumarVectores(arregloVectores[4],
                        Vector.multEscalar(arregloVectores[5],
                    Constantes.LAMBDA_L));
                // u=D1AXD3C/MODULO()
                arregloVectores[8] =
                Vector.multEscalar(Vector.productoCruz(arregloVectores[5],
                arregloVectores[2]),
                        1 /
                Vector.modulo(Vector.productoCruz(arregloVectores[5],
                arregloVectores[2])));
                // OD3P
                arregloVectores[7] =
Vector.sumarVectores(arregloVectores[6],
                        Vector.multEscalar(arregloVectores[8],
                Constantes.EXENTRICIDAD));
                // Gap
                                                distancias[i] =
                Vector.modulo(Vector.restaVectores(arregloVectores[1],
                arregloVectores[7]));
        } else {
                arregloVectores[0] = new Vector(nucleo.coordX[2 * i],
                nucleo.coordY[2 * i], nucleo.coordZ[2 * i]);
                arregloVectores[1] = new Vector(nucleo.coordX[(2 * i) + 1],
                nucleo.coordY[(2 * i) + 1],
                        nucleo.coordZ[(2 * i) + 1]);
                arregloVectores[2] = Vector.restaVectores(arregloVectores[0],
                arregloVectores[1]);
                arregloVectores[3] = new Vector(nucleo.coordX[0],
                nucleo.coordY[0], nucleo.coordZ[0]);
                arregloVectores[4] = new Vector(nucleo.coordX[1],
                nucleo.coordY[1], nucleo.coordZ[1]);
                // D1A
                arregloVectores[5] = Vector.restaVectores(arregloVectores[3],
                arregloVectores[4]);
                // OD3pp=OD1+l/l*D1A
                arregloVectores[6] =
Vector.sumarVectores(arregloVectores[4],
```

```
                                        Vector.multEscalar(arregloVectores[5],
                        Constantes.LAMBDA_L));
                        // u=D1AXD3C/MODULO()
                        arregloVectores[8] =
                        Vector.multEscalar(Vector.productoCruz(arregloVectores[5],
                        arregloVectores[2]),
                                        1 /
                        Vector.modulo(Vector.productoCruz(arregloVectores[5],
                        arregloVectores[2])));
                        // OD3P
                        arregloVectores[7] =
                Vector.sumarVectores(arregloVectores[6],
                                        Vector.multEscalar(arregloVectores[8],
                        Constantes.EXENTRICIDAD));
                        // Gap
                        distancias[i] =
                        Vector.modulo(Vector.restaVectores(arregloVectores[1],
                        arregloVectores[7]));
                }
        }

        double suma = 0;
        for (int i = 0; i < nucleo.numeroBarras; i++) {
                suma += distancias[i] * distancias[i];
        }
        this.fObjetivo = Math.sqrt(suma) / (nucleo.numeroBarras *
        Constantes.EXENTRICIDAD);
    }

    public void genVectores(Nucleo nucleo, int numeroBarra) {
        for (int i = 0; i < nucleo.numeroBarras; i++) {
                        this.arregloVectores[i] = new Vector(nucleo.coordX[2 * i] -
                        nucleo.coordX[(2 * i) + 1],
                                nucleo.coordY[2 * i] - nucleo.coordY[(2 * i) + 1],
                                nucleo.coordZ[2 * i] - nucleo.coordZ[(2 * i) + 1]);
        }
    }
}
```

La clase Nucleo es la que modela el punto en el que convergen varias barras en la parte interna de un marco recíproco; la siguiente imagen muestra un núcleo de tres barras.



Esta clase modela desde la geometría del núcleo hasta la generación arbitraria de elementos iniciales de la población. También tiene rutinas para normalizar longitudes de cada elemento (lo requiere el método), generar cadenas de texto con la información del elemento y diferentes constructores para apoyar a otras clases a manejar los elementos de la clase.

```
package al_gen;

import java.io.IOException;
import java.io.PrintWriter;
import java.io.Serializable;
import javax.swing.JOptionPane;

public class Nucleo implements Serializable, Comparable<Nucleo> {
```

```
    private static final long serialVersionUID = 1L;
    public int numeroBarras;
    protected Geometria geometria;
    protected double[] coordX;
    protected double[] coordY;
    protected double[] coordZ;
    public double[] longitudElementos;
    public double longitudProm;
    public Objetivo objetivo;
    public double rango;


public Nucleo(int numBarras, Geometria geom, double[] coordX, double[] coordY, double[]
                    coordZ, double[] lenElem,
                    double lenProm, Objetivo obj, double rango) {
            this.numeroBarras = numBarras;
            this.geometria = geom;
            this.coordX = coordX;
            this.coordY = coordY;
            this.coordZ = coordZ;
            this.longitudElementos = lenElem;
            this.longitudProm = lenProm;
            this.objetivo = obj;
            this.rango = rango;
    }

    public Nucleo(String a) throws IOException {
            this.geometria = new Geometria();
            this.setNumBarras(this.geometria.arregloBarras.length);
            this.objetivo = new Objetivo(this.numeroBarras);
            this.cargarCoordenadas();
            this.actualizarNodo();
            this.calcularRango();
            this.actualizarNodo();
            this.calcularLongProm();
            this.normalizarLongitudes();
    }
```

```
public Nucleo() throws IOException {
        this.geometria = new Geometria();
        this.setNumBarras(this.geometria.arregloBarras.length);
        this.objetivo = new Objetivo(this.numeroBarras);
        this.cargarCoordenadas();
        this.actualizarNodo();
        this.calcularRango();
        this.generarCoordAleatorias();
        this.actualizarNodo();
        // Es necesario volver a calcular la longitud promedio ya que se
        // modifican las longitudes de los elementos al cambiar las coordenadas
        // finales de los nodos.
        this.calcularLongProm();
        this.normalizarLongitudes();
        this.actualizarNodo();
}

public void cargarCoordenadas() {
        // Inicia los arreglos de coordenadas con el tamaño necesario
        // (2*numBarras) y genera las coordenadas de cada barra en los arreglos
        // a partir de los cargados en geometria.
        this.coordX = new double[2 * this.numeroBarras];
        this.coordY = new double[2 * this.numeroBarras];
        this.coordZ = new double[2 * this.numeroBarras];
        for (int i = 0; i < this.numeroBarras; i++) {
                int nodoI = geometria.arregloBarras[i][0];
                int nodoF = geometria.arregloBarras[i][1];
                this.coordX[2 * nodoI] = geometria.arregloNodos[nodoI][0];
                this.coordX[(2 * nodoI) + 1] = geometria.arregloNodos[nodoF][0];
                this.coordY[2 * nodoI] = geometria.arregloNodos[nodoI][1];
                this.coordY[(2 * nodoI) + 1] = geometria.arregloNodos[nodoF][1];
                this.coordZ[2 * nodoI] = geometria.arregloNodos[nodoI][2];
                this.coordZ[(2 * nodoI) + 1] = geometria.arregloNodos[nodoF][2];
        }
}

public void calcularLongProm() {
        double suma = 0;
```

```
            for (int i = 0; i < this.numeroBarras; i++) {
                suma += this.longitudElementos[i];
            }
            this.longitudProm = suma / this.numeroBarras;
        }

        public void nucleoZero() {
            for (int i = 0; i < this.coordX.length; i++) {
                this.coordX[i] = 0;
                this.coordY[i] = 0;
                this.coordZ[i] = 0;
            }
        }

        public void calcularRango() {
            // Calcula el valor del rango para generar el cubo donde se
            // crearán las coordenadas aleatorias de las barras. El tamaño
            // del arreglo siempre será igual al número de dimensiones
            // que se consideren (siempre 3). No es cierto, es la misma
            // para las 3 dimensiones
            this.rango = this.longitudProm * (Constantes.LAMBDA_L);
        }

        public void generarCoordAleatorias() {
            // Genera coordenadas aleatorias dentro del rango del cubo generado por
            // las barras del nucleo.
            // Valores mínimo y máximo en cada eje. Se toma a partir del centro del
nucleo.
            // Asumir que el punto del centro es el último en el arreglo, aunque esto
            // no sea cierto para n Nucleos en la malla.
            double[] rangoX = new double[2];
            double[] rangoY = new double[2];
            double[] rangoZ = new double[2];
            rangoX[0] = this.geometria.arregloNodos[numeroBarras][0] - (this.rango /
            2);
            rangoX[1] = this.geometria.arregloNodos[numeroBarras][0] + (this.rango /
            2);
```

```
            rangoY[0] = this.geometria.arregloNodos[numeroBarras][1] - (this.rango /
            2);
            rangoY[1] = this.geometria.arregloNodos[numeroBarras][1] + (this.rango /
            2);
            rangoZ[0] = this.geometria.arregloNodos[numeroBarras][2] - (this.rango /
            2);
            rangoZ[1] = this.geometria.arregloNodos[numeroBarras][2] + (this.rango /
            2);
            for (int i = 0; i < 2 * this.numeroBarras; i++) {
                  // Revisar si i es par o impar; los nodos finales se encuentran en
                  // los índices nones de los arreglos de coordenadas, son los que se
                  // tienen que modificar.
                  if ((i & 1) != 0) {
                        this.coordX[i] = Math.random() * (rangoX[1] - rangoX[0]) +
                  rangoX[0];
                        this.coordY[i] = Math.random() * (rangoY[1] - rangoY[0]) +
                  rangoY[0];
                        this.coordZ[i] = Math.random() * (rangoZ[1] - rangoZ[0]) +
                  rangoZ[0];
                  }
            }
}

public void normalizarLongitudes() {
            // El algoritmo requiere que las longitudes de las barras sean iguales para
            // cada nodo. Esta función modifica las coordenadas dentro del cubo (las
            // coordenadas finales de cada barra) para que todas sean iguales a la
            // longitud promedio.
            for (int i = 0; i < 2 * this.numeroBarras; i++) {
                  if ((i & 1) != 0) {
                        double factorEscala = this.longitudProm /
                        this.longitudElementos[((i + 1) / 2) - 1];
                        this.coordX[i] = this.coordX[i - 1] + (this.coordX[i] -
                        this.coordX[i - 1]) * factorEscala;
                        this.coordY[i] = this.coordY[i - 1] + (this.coordY[i] -
                        this.coordY[i - 1]) * factorEscala;
                        this.coordZ[i] = this.coordZ[i - 1] + (this.coordZ[i] -
                        this.coordZ[i - 1]) * factorEscala;
```

```
                }
        }
}

public void actualizarNodo() {
        this.calcularLongitudElem();
        this.calcularLongProm();
        this.normalizarLongitudes();
        this.calcularLongitudElem();
        this.calcularLongitudElem();
}

public void calcularLongitudElem() {
        this.longitudElementos = new double[numeroBarras];
        for (int i = 0; i < this.numeroBarras; i++) {
                double x = this.coordX[(2 * i) + 1] - this.coordX[(2 * i)];
                double y = this.coordY[(2 * i) + 1] - this.coordY[(2 * i)];
                double z = this.coordZ[(2 * i) + 1] - this.coordZ[(2 * i)];
                this.longitudElementos[i] = Math.sqrt((x * x) + (y * y) + (z * z));
        }
}

public String toString() {
        return "Numero de barras: " + this.numeroBarras + "\nCoordX: " +
        generarCadenaCoord('X') + "\nCoordY: "
                + generarCadenaCoord('Y') + "\nCoordZ: " + generarCadenaCoord('Z')
                + "\nLongitud elementos: "
                        + generarCadenaCoord('L') + "\nLongitud promedio: " +
                this.longitudProm + "\nObjetivo: "
                        + this.objetivo.fObjetivo;
}

public void setNumBarras(int numBarras) {
        this.numeroBarras = numBarras;
}

public String generarCadenaCoord(char coordenada) {
        String coordenadas = "";
```

```
        if (coordenada == 'X') {
                for (int i = 0; i < this.coordX.length; i++) {
                        coordenadas = coordenadas + String.format("%.4s   ",
        this.coordX[i]);
                }
        }
        if (coordenada == 'Y') {
                for (int i = 0; i < this.coordY.length; i++) {
                        coordenadas = coordenadas + String.format("%.4s   ",
        this.coordY[i]);
                }
        }
        if (coordenada == 'Z') {
                for (int i = 0; i < this.coordY.length; i++) {
                        coordenadas = coordenadas + String.format("%.4s   ", -
                this.coordZ[i]);
                }
        }
        if (coordenada == 'L') {
                for (int i = 0; i < this.longitudElementos.length; i++) {
                        coordenadas = coordenadas + String.format("%.4s   ",
                this.longitudElementos[i]);
                }
        }
        return coordenadas;
}

public void mostrarInfoNucleo() {
        JOptionPane.showMessageDialog(null, this.toString(), "Info",
        JOptionPane.INFORMATION_MESSAGE);
}

public Nucleo clone() {
        Nucleo nvoNucleo = new Nucleo(numeroBarras, geometria, coordX, coordY,
        coordZ, longitudElementos, longitudProm,
                        objetivo, rango);
        nvoNucleo.objetivo = this.objetivo.clone(this.numeroBarras);
        nvoNucleo.geometria = geometria;
```

```
        return nvoNucleo;
}

public void mutacion() {
        for (int i = 0; i < this.numeroBarras; i++) {
                if (Math.random() < Constantes.TASA_MUTACION) {
                        if (Math.random() < 0.5) {
                                this.coordX[2 * i + 1] = coordX[2 * i + 1]
                                                + coordX[2 * i + 1] * (Math.random() *
                Constantes.DISTANCIA_MUTACION);
                        } else {
                                this.coordX[2 * i + 1] = coordX[2 * i + 1]
                                                - coordX[2 * i + 1] * (Math.random() *
                Constantes.DISTANCIA_MUTACION);
                        }
                        if (Math.random() < 0.5) {
                                this.coordY[2 * i + 1] = coordY[2 * i + 1]
                                                + coordY[2 * i + 1] * (Math.random() *
                Constantes.DISTANCIA_MUTACION);
                        } else {
                                this.coordY[2 * i + 1] = coordY[2 * i + 1]
                                                - coordY[2 * i + 1] * (Math.random() *
                Constantes.DISTANCIA_MUTACION);
                        }
                        if (Math.random() < 0.5) {
                                this.coordZ[2 * i + 1] = coordZ[2 * i + 1]
                                                + coordX[2 * i + 1] * (Math.random() *
                Constantes.DISTANCIA_MUTACION);
                        } else {
                                this.coordZ[2 * i + 1] = coordZ[2 * i + 1]
                                                - coordX[2 * i + 1] * (Math.random() *
                Constantes.DISTANCIA_MUTACION);
                        }
                }
        }
}

public void guardarArchivo(String directorio) throws IOException {
```

```
                // Cada vez que se corre el algoritmo se sobreescriben los archivos
                // de corridas anteriores para evitar guardar demasiados arreglos.
                // Ambos archivos se crean en el directorio base y después
                // se mueven a la carpeta correspondiente.
                String comando = "mkdir " + directorio;
                Runtime.getRuntime().exec(comando);
                PrintWriter guardarCoordenadas = new PrintWriter("Coordenadas.csv");
                String linea;
                for (int i = 0; i < 3; i++) {
                        linea = "";
                        for (int j = 0; j < this.coordX.length; j++) {
                                if (i == 0) {
                                        linea += coordX[j] + ",";
                                } else if (i == 1) {
                                        linea += coordY[j] + ",";
                                } else if (i == 2) {
                                        linea += -coordZ[j] + ",";
                                }
                        }
                        StringBuilder sb = new StringBuilder(linea);
                        sb.deleteCharAt(linea.length() - 1); // Borrar la última coma
                        linea = sb.toString();
                        guardarCoordenadas.println(linea);
                }
                guardarCoordenadas.close();
                PrintWriter guardarIndices = new PrintWriter("Barras.csv");
                for (int i = 0; i < this.geometria.arregloBarras.length; i++) {
                        linea = "";
                        linea += geometria.arregloBarras[i][0] + "," +
geometria.arregloBarras[i][1];
                        guardarIndices.println(linea);
                }
                guardarIndices.close();
                comando = "mv Coordenadas.csv " + directorio + "/Coordenadas.csv";
                Runtime.getRuntime().exec(comando);
                comando = "mv Barras.csv " + directorio + "/Barras.csv";
                Runtime.getRuntime().exec(comando);
        }
```

```
public void mostrarNucleo(){
}

@Override
public int compareTo(Nucleo nucleo) {
        return Double.compare(nucleo.objetivo.fObjetivo, objetivo.fObjetivo);
        // if (this.objetivo.fObjetivo <= nucleo.objetivo.fObjetivo) {
        // return -1;
        // } else if (this.objetivo.fObjetivo == nucleo.objetivo.fObjetivo) {
        // return 0;
        // } else {
        // return 1;
        // }
}
```

Capítulo V. PRODUCTOS, RESULTADOS E IMPACTOS GENERADOS

5.1 Productos obtenidos

Los productos obtenidos fueron las funciones generadas para los programas y los diagramas finales de la estructura propuesta con la revisión de los perfiles der madera.

La revisión del maro recíproco se realizó en StaadPro para poder conocer los esfuerzos que se generan en las barras que lo componen. Dado que el algoritmo evolutivo genera los conceptos de solución de manera aproximada, fue necesario tomar las coordenadas de cada barra y ajustarlas a los valores finales para el análisis en el programa.

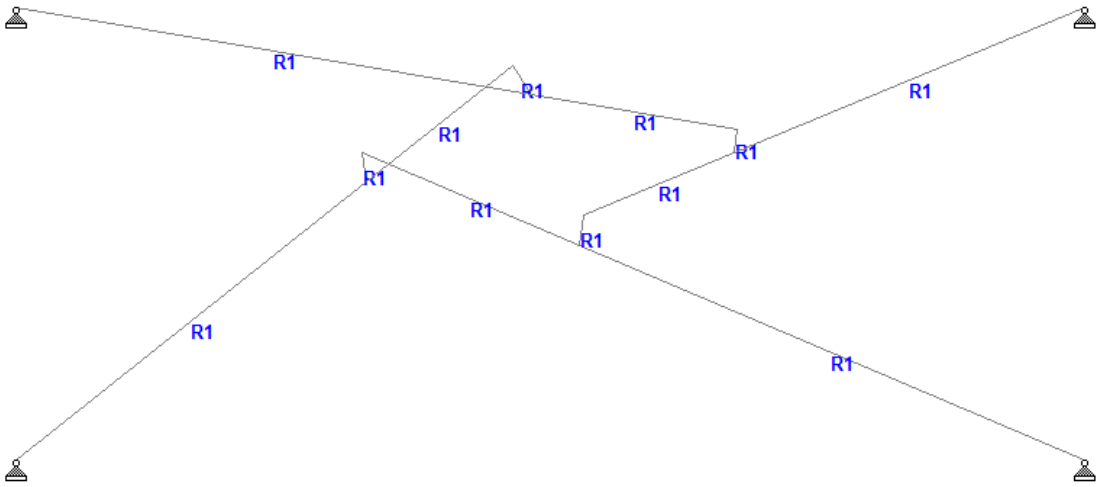
Debido a que un marco recíproco no se puede analizar con el método de rigidez por el hecho de que los puntos donde convergen las barras no se encuentran en la misma línea, se propuso una aproximación para realizar el análisis en el programa. Para esto se colocaron elementos barra de manea vertical entre la coordenada final de cada barra y el punto en donde se apoya sobre la otra. Para simular que este elemento si fuera parte del elemento, se colocaron elementos sumamente rígidos para evitar una deformación muy alta en los elementos verticales y poder emular de manera aproximada el comportamiento.

Se colocaron cargas puntuales a los elementos del marco, colocando la carga en el punto donde cruzan cada uno de los elementos. Para la magnitud de la carga se tomó una carga de 2 kg/m2, suponiendo que la cubierta se realiza de la misma manera que en la propuesta de la Cruz Roja, con lona de plástico.

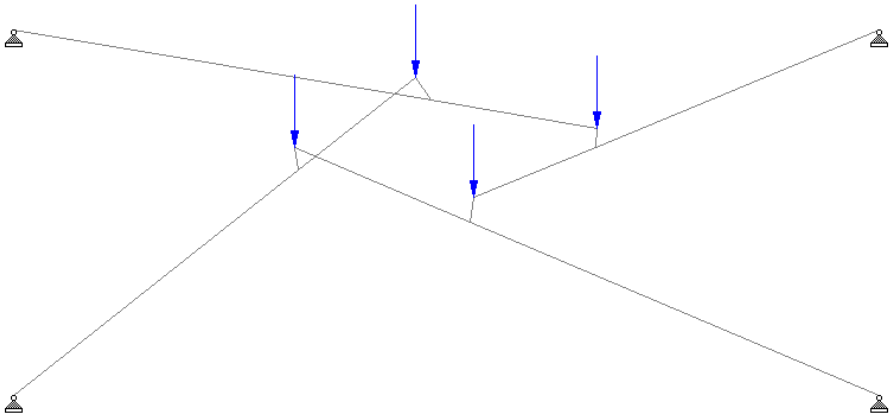
A continuación, se muestran los contenidos del archivo .std que modela el marco; este se puede correr en Staad.Pro por si se desean modificar algunas de las propiedades o cargas.

```
STAAD SPACE
START JOB INFORMATION
ENGINEER DATE 01-Dec-16
END JOB INFORMATION
INPUT WIDTH 79
UNIT METER KG
JOINT COORDINATES
1 0 0 0; 2 2.7 1.85 0.36; 3 4 0 0; 4 2.13 2.69 0.4; 5 4 4 0;
6 1.3 2.14 0.4; 7 0 4 0; 8 1.86 1.31 0.37; 9 1.9323 1.32398 0.25764;
10 1.31562 2.0973 0.26171; 11 2.11 2.698 0.28;
12 2.69107 1.8829 0.279986;
MEMBER INCIDENCES
1 1 9; 2 3 12; 4 8 10; 5 9 2; 6 8 9; 7 10 7; 8 6 10; 9 6 11; 10 11 5;
13 12 4; 14 2 12; 15 4 11;
DEFINE MATERIAL START
ISOTROPIC MADERA
E 20.9042
POISSON 0.3
DENSITY 800
ALPHA 1.2e-005
DAMP 0.03
G 8.04007e+009
END DEFINE MATERIAL
MEMBER PROPERTY
1 2 4 TO 10 13 TO 15 PRIS YD 0.15 ZD 0.15
CONSTANTS
MATERIAL MADERA ALL
SUPPORTS
1 3 5 7 PINNED
LOAD 1 LOADTYPE Dead  TITLE LOAD CASE 1
JOINT LOAD
2 4 6 8 FY 8
FINISH
```

En las siguientes imagenes se puede observar el modelo de la cubierta en Staad.Pro con diferentes vistas del programa.

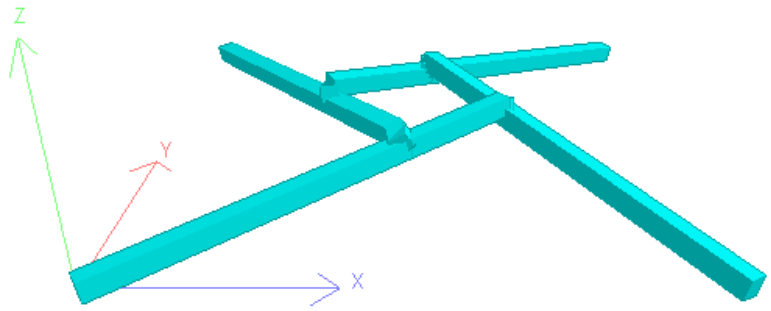


Modelo en StaadPro



Modelo con las cargas colocadas

en los nodos.



Vista en tres dimensiones con los perfiles propuestos.

El modelo mostrado anteriormente se generó con el algoritmo genético propuesto en este proyecto. Este se utilizó para generar una configuración geométrica válida con las características del refugio propuesto por la Cruz Roja. Después de haber realizado la configuración, se procedió a cargar el modelo en StaadPro para obtener los elementos mecánicos siguiendo el proceso que se mostró anteriormente.

Después de haber cargado el elemento se pudo notar que este tipo de estructuras no se puede analizar con este programa; esto puede deberse a los métodos de análisis que se utilizan y a que se está realizando una simplificación de la estructura al colocar las barras verticales para generar las excentricidades.

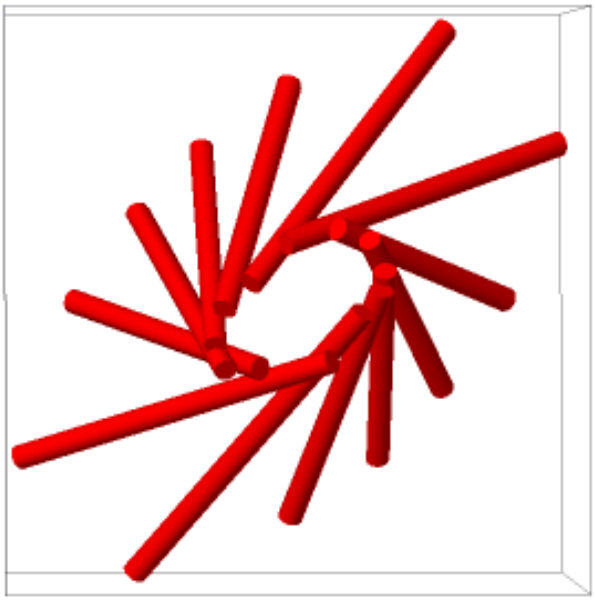
5.2 Resultados alcanzados

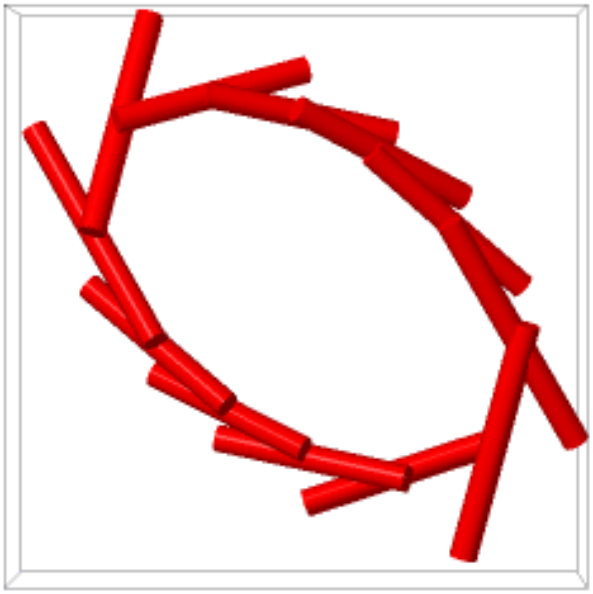
Uno de los resultados principales que se buscaron fue el de desarrollar un sistema que pudiera generar conceptos de solución para refugios, el cual se pudo generar para crear los sistemas de cubiertas, que en este caso se utilizaron para la revisión de la propuesta de la Cruz Roja.

A pesar de que la implementación del algoritmo genético fue exitosa y se pudo utilizar para generar diferentes configuraciones, no se pudo realizar la revisón de esfuerzos en el marco implementado por la Cruz Roja ya que el método de análisis para marcos recíprocos no es el mismo que el de un marco en el cual las barras llegan al mismo punto que las que lo soprtan.

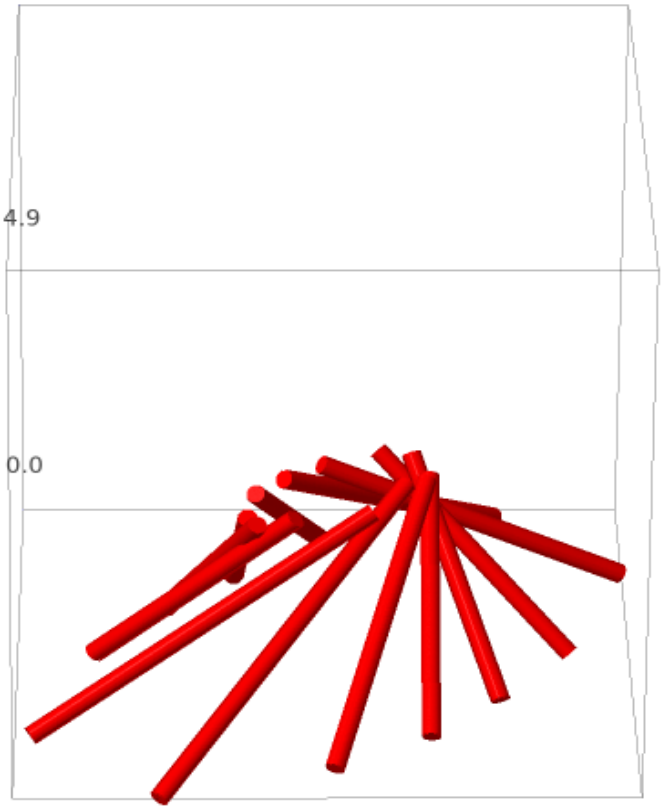
Además de la revisión del elemento de le cruz roja, se generaron varios marcos recíprocos con configuraciones diferentes para probar la capacidad y limitantes del algoritmo. Una de las principales ventajas que se encontró a favor de esta implementación contra la propuesta en el artículo de Baverel y Nooshin, fue que este si puede generar configuraciones con barras de diferentes tamaños, a diferencia de el anterior, que tenía una condición para mantener todas las barras fijas. A continuación, se muestra un marco recíproco generado con el algoritmo, el cual se realizó con una configuración básica que generara de manera forzada una barra más pequeña.

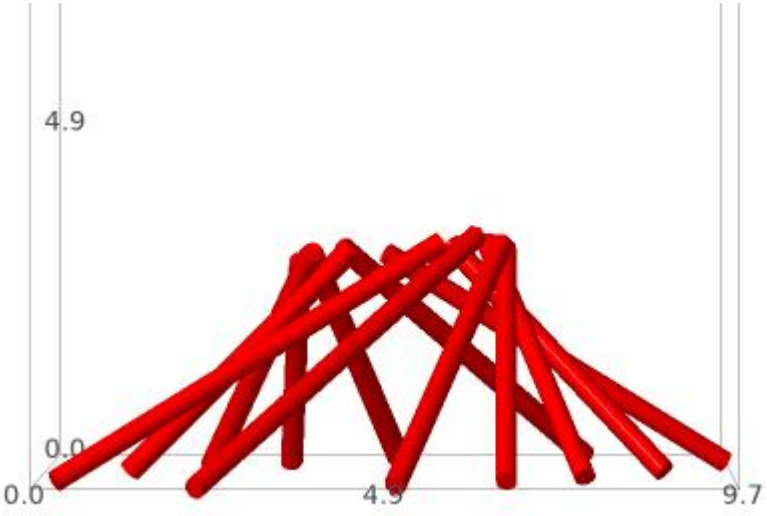
Para continuar con la exploración del algoritmo se realizaron configuraciones con diferentes geometrías externas, una particularmente interesante fue un marco generado con una elipse como figura externa.





Vistas laterales:





Proyectos extras.

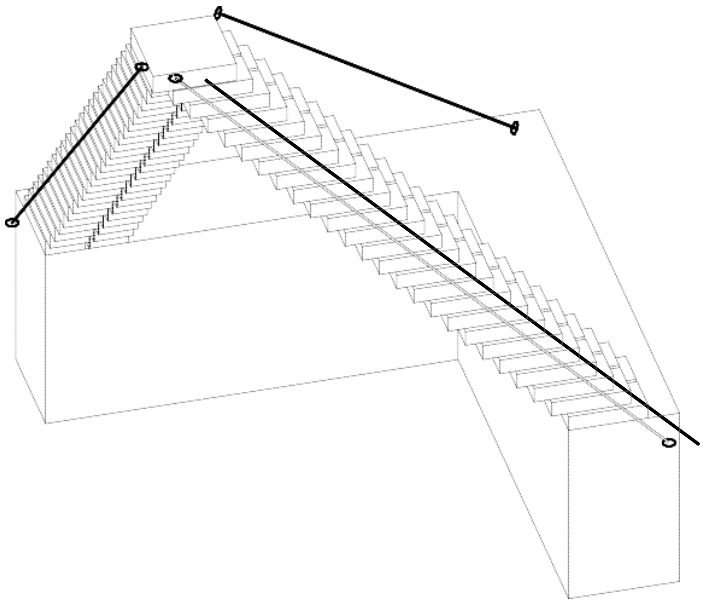
Además del proyecto del algoritmo genético para la generación de marcos recíprocos y la revisión del refugio de la cruz roja, se realizaron varios proyectos dentro del PAP de manera simultánea. Estos proyectos fueron:

- Realizar la revisión de un proyecto de Tu Techo y hacer una revisión técnica.
- Construcción de una cubierta y cuatro columnas de diferentes materiales.

Cubierta.

El proyecto de la cubierta consistió en realizar cuatro columnas de madera, tierra o mampostería en la parte trasera del ITESO. Dichas columnas se apoyan en una escuadra de mampostería previamente existente; en las columnas se apoyaría una cubierta, construida con bambú, a la que se le aplicarían ciertas cargas para probarlas.

Este proyecto consta de dos partes. La primera era el diseño de los elementos, en la que a cada equipo se le asignó una columna (o la cubierta) y se debieron diseñar para ciertas cargas mínimas. Para esto se propuso una altura a la que debía llegar cada columna y cierta ubicación. A continuación, se muestra un diagrama de uno de los apoyos de mampostería que se diseñó para este proyecto.



Una vez realizado el diseño y el proyecto ejecutivo, se pasó el proyecto a otro equipo, el que estaría encargado de la construcción. Para este caso se debió diseñar una columna con una base como la imagen anterior. Se eligió realizar el diseño con mampostería y tensores para ayudar a soportar el momento de volteo que generaría la cubierta.

El proyecto de la columna se realizó con ladrillo para bóveda, con dimensiones de 4cm x 10 cm x 20cm, y unos tensores con cuerda de ixtle para ayudar a la mampostería a soportar las tensiones que se generarían en la parte superior del elemento al colocar la cubierta de bambú y por la carga muerta de los mismos materiales de la columna. A continuación, se muestra el proyecto ejecutivo de la columna de mampostería.

Proyecto Ejecutivo

RESUMEN

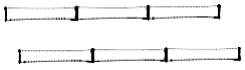
El siguiente manual es el proceso de construcción de una columna de mampostería. Esta es una de cuatro apoyos más, que en conjunto más una cubierta, componen un sistema estructural mixto. Diseñado como parte del Proyecto de Aplicación Profesional: Programa de edificación y vivienda. Este, será construido en el mismo taller y sometido a diferentes pruebas.

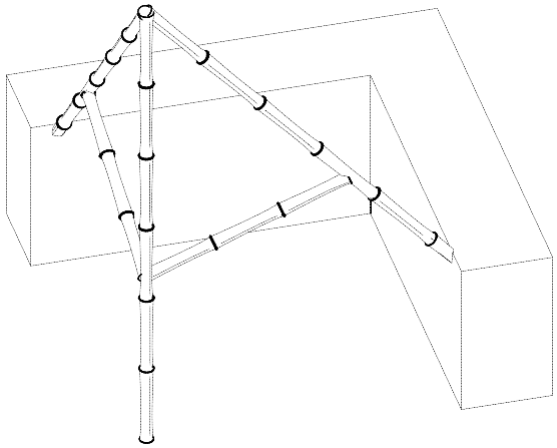
El apoyo aquí descrito consta de dos partes primordiales, una estructura de mampostería que funciona en compresión y una seria de tensores que ayudan en los esfuerzos de tensión en la que la técnica escogida carece de fuerza.

1.6 m

1.55 m

.65m

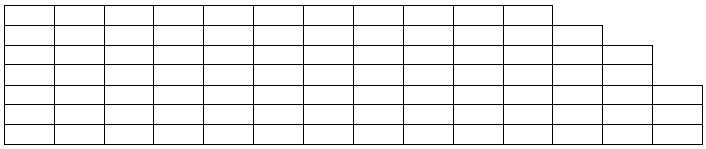




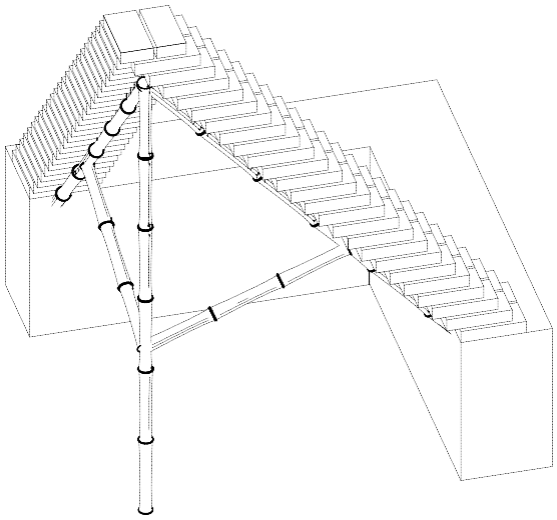
- Hay que hacer cortes radiales en los extremos de las barras que se intersectan entre si para un ensamble adecuado.

- Los puntales se fijan entre ellos con alambre recocido. La fijación a la dala se hace con clavo de concreto y alambre recocido.

98 piezas de ladrillo (4 x 10 x 20cm)

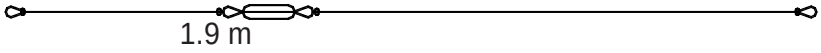


25 lts de mezcla
-4partes de cemento
-1 parte de arena de rio cernida
-agua

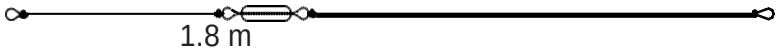


hilada 21 hilada 22 hilada 23 hilada 24 hilada 25

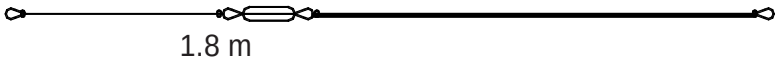
- Las juntas entre ladrillos es de 1 cm de mezcla.
- El desfase entre hilada e hilada es de 4.33 cm.



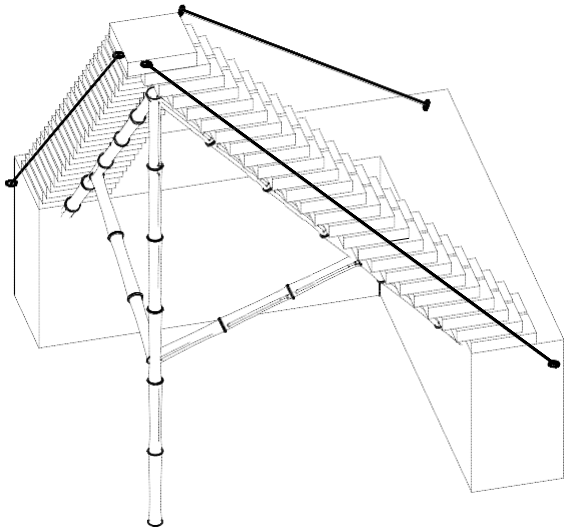
1.9 m



1.8 m

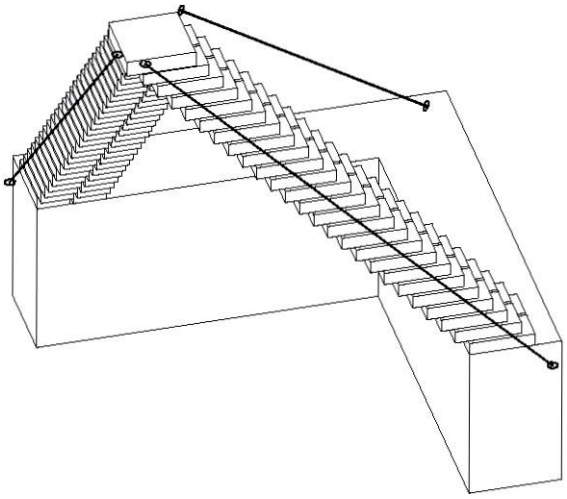


1.8 m

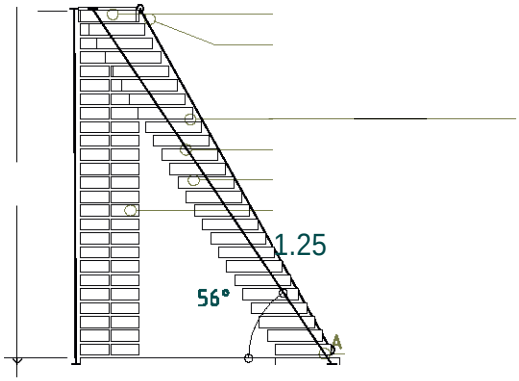


- Fijar todas las armellas antes de colocar el cable de acero.

-Tensar el cable con el tensor hasta que el sistema libere tensión sobre los puntales.



-Retirar puntales de bambú.



Esquinera de Madera

Armella diametro 1 1/2"

Traslape de 29 mm

Cable de acera

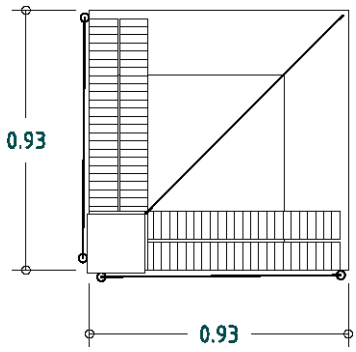
Ladrilla de lama (4x10x20cml

Junta de cemento de 1cm

,Armella diametro 112"

0.30

0.10 0.23



1.56

1.53

Después de realizar el diseño, se realizó el cambio de los proyectos ejecutivos para proceder a la construcción de cada elemento. En nuestro caso, debimos construir una cubierta a base de elementos de bambú, que en la parte interna tenía un marco recíproco de cuatro elementos y en la parte externa cuatro barras horizontales que se encargaban del apoyo en cada columna. A continuación, se muestra el proyecto ejecutivo que se nos presentó para construir el marco.

PAP

CUBIERTA

PROYECTO EJECUTIVO

_____

MARISCAL EMMANUEL

TALAMANTES ALFREDO

VÁZQUEZ FRANCO YAHIR

INTRODUCCIÓN

Se llevará a cabo la construcción de un tipo de refugio con dimensiones cuadradas con un apoyo en cada esquina, con la condición de que en cada esquina sea construido un apoyo de distintos materiales.

Cada una de las esquinas debe tener adaptada una base de al menos 10x10 cm en donde se apoyará la cubierta del refugio también de otro material, para así cerrarlo.

A nuestro equipo le fue asignada realizar el proyecto de la cubierta, la cual será a base de bambú; proponemos utilizar varios sistemas de conexión para demostrar las ventajas y desventajas de cada uno así como un facilidad constructiva.

TEORÍA

Se pretende construir primeramente un marco a base de secciones de bambú machimbradas en los extremos y estas conexiones a su vez reforzadas con amarres de hilaza.

Para crear una especie de cúpula se propone un sistema a base de marcos recíprocos con 4 elementos de bambú simplemente apoyados con un refuerzo de hilaza.

En la imagen se muestra la intención del proyecto:

MATERIALES Y HERRAMIENTA

- MATERIALES
  - Bambú guadua - - - - - - - - - - - - - - - - 4.15 ml
  - Hilaza - 1 rollo
  - Cinta canela - - - - - - - - - - - - - - - - - - 1 rollo



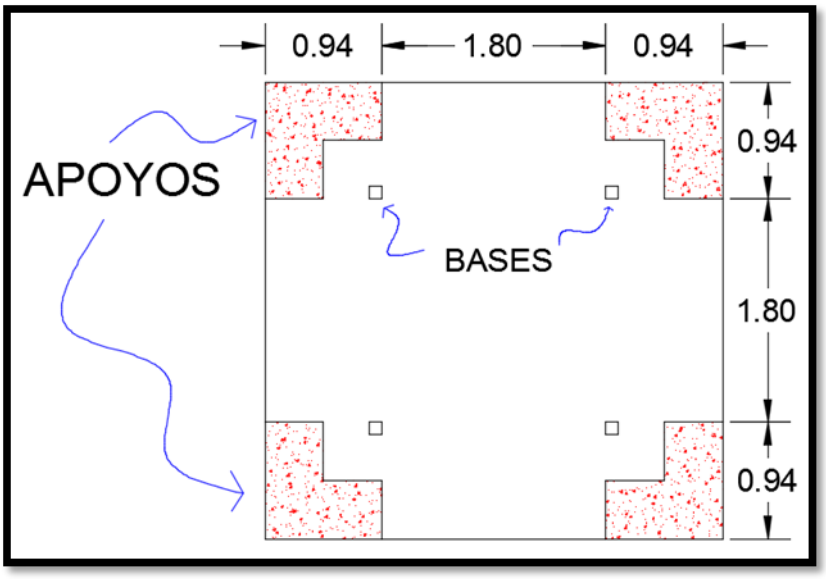
- HERRAMIENTAS
  - Esmeril
  - Guantes de seguridad
  - Gafas de seguridad

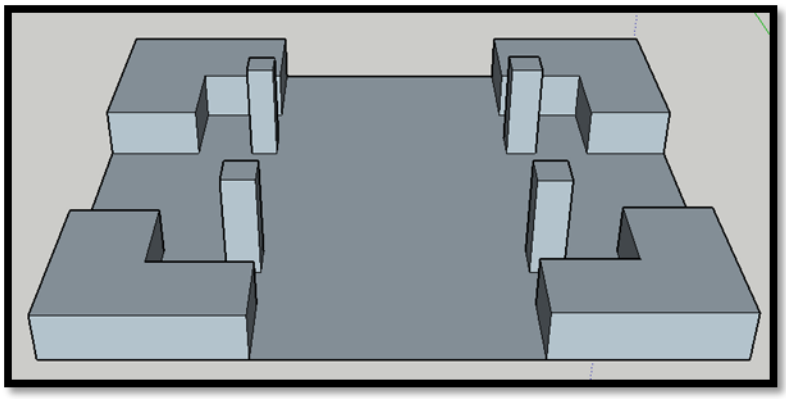


PROCESO CONSTRUCTIVO

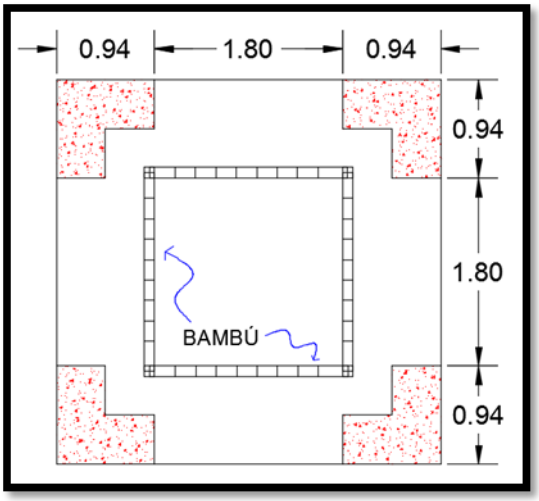
1. En primer lugar, se presentarán los materiales, o sea, acomodar los elementos simulando la posición final que tendrán. Esto para ver en donde se apoyará un elemento sobre otro y con eso hacer una marca para saber en dónde se hará el machimbrado para el amarre.

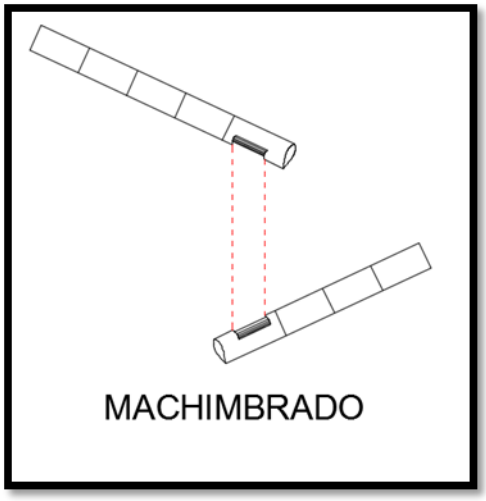
2. Están dados 4 apoyos esquineros, los cuales cada equipo adaptó correctamente para generar una base de 10x10 cm para que en dichas bases se apoye la cubierta.



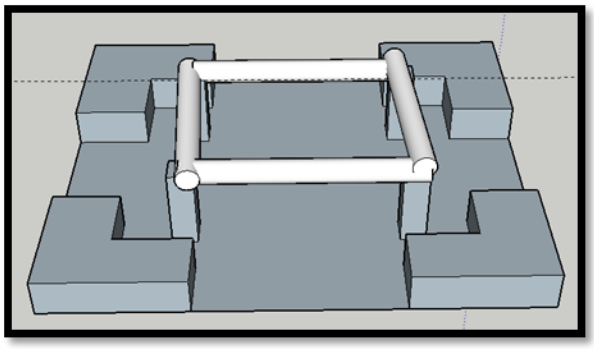


3. Posteriormente se puede proceder a realizar el corte con el esmeril para machimbrar las piezas del marco de inferior.

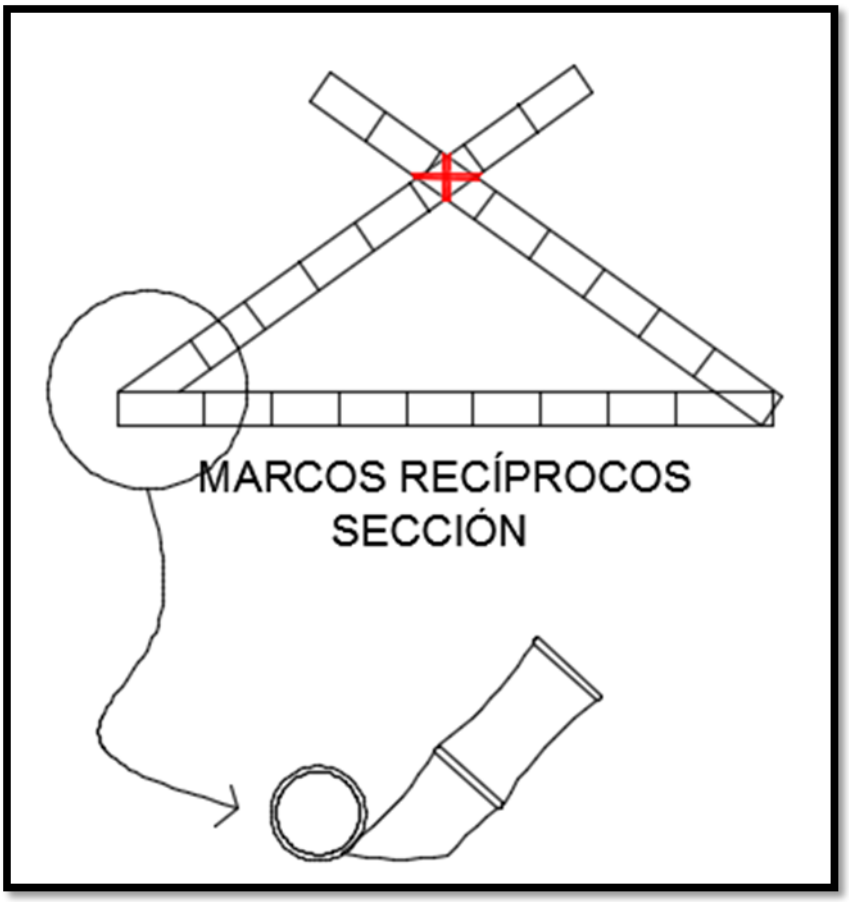




4. Una vez machimbradas las piezas se procede a fijar la conexión mediante un amarre con hilaza a manera de ochos y sobre el perímetro del elemento.

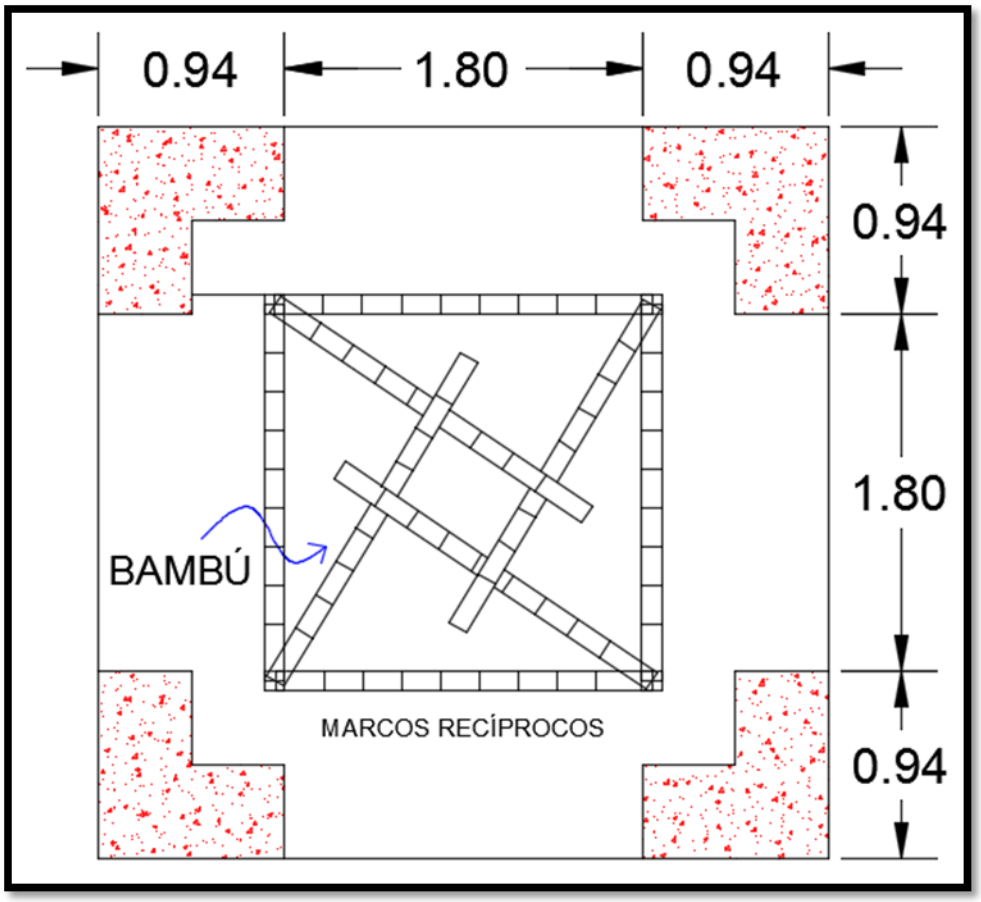


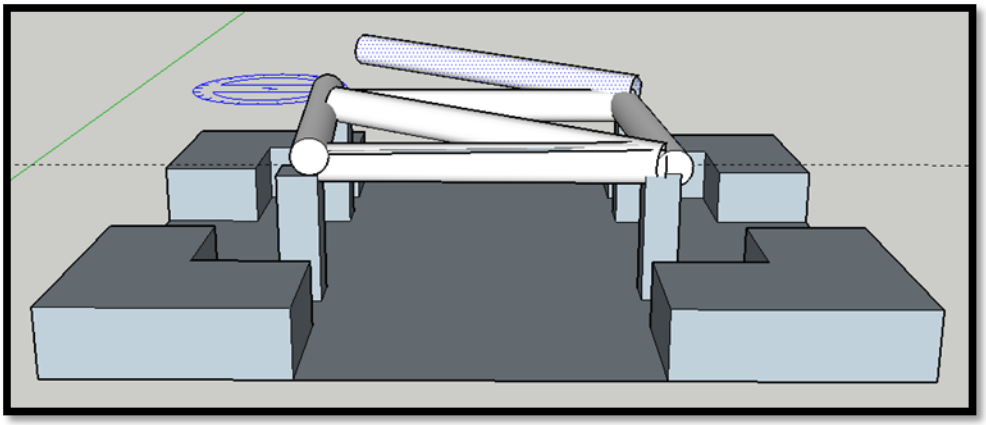
5. Una vez teniendo la estructura de apoyo bien fijada, presentar sobre de ella los elementos que formarán la cubierta a manera de marco recíproco, y de igual manera se hace la marca, una en la parte inferior que se conectará con el marco inferior de apoyo donde llevará un corte en diagonal y otro para colocar la cinta canela la cual ayudará aportando mayor fricción entre los elementos de bambú en conjunto con el amarre.



6. Ya hechas las marcas se encinta el elemento dando 5 vueltas al perímetro del bambú.

7. Por último, se amarran las últimas conexiones.


0.94
1.80
0.94
0.94
1.80
0.94
BAMBÚ
MARCOS RECÍPROCOS



PRESUPUESTO

- DESPIECE

CONCEPTO	DIMENSIONES	COSTO UNITARIO	COSTO $
4 tramos bambú	2 m	$22.20 /ml (3”)	$44.40
4 tramos bambú	2.15 m	$22.20 /ml (3”)	$44.73
Hilaza	Rollo	$15	$15.00
Cinta canela	Rollo	$10	$10.00
TOTAL			$114.13

La construcción de la cubierta se realizó como se especificaba en el proyecto, pero fue necesario cambiar algunas de las medidas de los elementos ya que algunos de los apoyos de las columnas no llegaron a la ubicación solicitada; para esto, se midieron las ubicaciones reales de los apoyos construidos al momento del ensamble de la cubierta y se modificaron los elementos necesarios para que las columnas soportaran la cubierta de bambú.

La realización de este proyecto presentó algunos problemas básicos que suelen ser muy comunes en la construcción en México. Como el hecho de no revisar los proyectos ejecutivos, suponiendo que se pueden modificar en campo para ajustarlos. Este fue un problema con el que tuvimos que lidiar con la cubierta, ya que las medidas presentadas en el proyecto ejecutivo no fueron las correctas y se

tuvieron que modificar los elementos para ajustarlas. Otro problema es el de la viabilidad del sistema constructivo. No se revisaron las conexiones del proyecto y al intentar construirlo, fue necesario implementar una conexión en campo, por lo que fue necesario modificar los elementos del maro recíproco para poderlos unir a la base.

Al final, fue posible la construcción de la cubierta con unas dimensiones similares a las propuestas y con una geometría que cumplió con los requisitos del proyecto ejecutivo, pero fue necesario hacer algunas modificaciones. Se pudo observar la importancia de revisar la viabilidad del proyecto antes de aceptarlo y comenzar a construirlo.

Revisión de Tu Techo.

Este proyecto consistió en revisar los proyectos que se están realizando en San Andrés Cohamiata por los integrantes de Tu Techo y hacer una revisión técnica de los mismos, para posteriormente hacerles propuestas de modificación a sus proyectos y obtener una respuesta de parte de Tu Techo sobre la propuesta realizada. En nuestro caso (dado que el proyecto de algoritmos tenía diferentes tareas), debimos realizar una revisión de la (o las) metodologías utilizadas para conocer las propiedades de la zona, como clima, flora y fauna, materiales, etc., para evaluar si la manera en la que se planteaban las propuestas era la más óptima en función de las características sociales, culturales y climatológicas de la zona.

Para la realización de este proyecto se procedió a realizar una investigación sobre las características sociales y climatológicas de la zona y, en una reunión con

los integrantes de Tu Techo, investigar cuales eran las características que ellos conocían y cuáles de ellas tomaban en cuenta para sus proyectos. A continuación de muestran las características importantes de la zona que se encontraron.

Características sociales de San Andrés Cohamiata.

Una de las principales comunidades wixárika de Jalisco. Está situada en el municipio de Mezquitic, a 1950 metros de altitud. En el censo de 2010 tenía una población de 1317 habitantes, con un porcentaje de analfabetismo del 16%.

La estructura gubernamental es diferente a la de otras comunidades, ya que tienen una asamblea comunitaria en la que se toman en cuenta la opinión de los habitantes para la toma de decisiones en asuntos de la población. Los cargos políticos están relacionados con la vida religiosa. Tienen un "Gobernador Tradicional" y se toman las decisiones a través de la asamblea comunitaria.

Principales actividades:

La agricultura es la principal fuente de auto sustento. También se realiza ganadería y elaboración y venta de artesanías. Recientemente comenzaron a tener puestos gubernamentales y educativos. Obtener proporciones de la población que se dedica a cada actividad.

Tu Techo en San Andrés Cohamiata.

Se encarga de proporcionar viviendas para los habitantes de la zona. Busca crear proyectos con los usuarios para conocer sus necesidades. Conseguir el número de proyectos que se han construido, así como su estado actual.

Al realizar la entrevista se notaron dos puntos importantes. El primero es que la propuesta de vivienda de hace en función de las necesidades de la persona que habitará la casa. El segundo es que no se tiene una base de datos con información de la zona que puede ser interesante y útil para el desarrollo de las propuestas.

Con la información recabada en la investigación y la entrevista realizada con uno de los integrantes de Tu Techo se propone que se realicen los siguientes cambios para intentar tener un mayor conocimiento de la zona e intentar tener unas mejores propuestas de vivienda con el conocimiento de la zona.

- Es necesario conseguir información más específica sobre la estructura de gobierno, como puestos específicos, funcionamiento, cantidad de personas relacionadas / implicadas en esto.
- Es necesario obtener información sobre las relaciones o formas de trabajo de la asamblea comunitaria y las actividades económicas para observar cómo trabajan o se apoyan entre sí.
- Conocer la aceptación o rechazo que tiene Tu Techo en la comunidad y la facilidad con la que se realizan o implementan los productos.

Capítulo VII. CONCLUSIONES Y RECOMENDACIONES

7.1 Conclusiones

Referencias bibliográficas (sistema APA).

Daniel W. Dyer. (2010). Evolutionary Computation in Java. 30 de noviembre de 2016, de - Sitio web: http://watchmaker.uncommons.org/manual/index.html

Regan Pontangroa. (2014). The challenge of shelter in post disaster reconstruction. 11 de noviembre de 2016, de ScienceDirect Sitio web: http://www.sciencedirect.com/science/article/pii/S1877042815017784.

Olga Popovic Larsen. (2008). Capítulo 1. En Reciprocal Frame Architecture (207). Oxford: Elsevier Ltd.

NASA Evolvable Antenna. Imagen obtenida de: https://ti.arc.nasa.gov/m/pub-archive/1244h/1244%20(Hornby).pdf, 11 de noviembre de 2016.

Kang, Chuang. (2011). A genetic algorithm-based Boolean delay model of intracellular signal transduction in inflammation. 30 de noviembre de 2011, de Openi Sitio web: https://openi.nlm.nih.gov/detailedresult.php?img=PMC3044271_1471-2105-12-S1-S17-9&req=4.