

Universidad Nacional de Ingeniería
Unidad de Posgrado de la Facultad de Ciencias
Maestría en Ciencia de la Computación



Informe 02

Gestión de Recursos Humanos en Proyectos de Software

Elaborado por:

Eduardo Yauri Lozano

Milton Palacin Grijalva

Profesor:

Dr. Glen Dario Rodriguez Rafael

Lima - Peru

5 de septiembre de 2021

Índice

1. Introducción	2
2. Marco Teórico	2
2.1. Definición del problema	2
2.2. Solución y Normalización de Funciones Multiobjetivo	5
2.3. Algoritmo Particle Swarm Optimization (PSO)	6
2.4. Algoritmo Ant Colony Optimization (ACO)	6
3. Estado del Arte	7
4. Metodología Utilizada	8
5. Implementacion y Desarrollo	9
5.1. Implementación con PSO	9
5.2. Implementación con ACO	14
5.2.1. Diseño de Grafo para la Tarea	14
5.2.2. Implementación de ACO para SPSP	16
5.2.3. Administración de Feromona	19
5.2.4. Selección del Mejor Nodo	20
5.2.5. Información de la Heurística	20
5.2.6. Función Fitness y Pesos	21
6. Ejecución y Resultados	22
6.1. Caso de Prueba	22
6.2. Resultados PSO	23
6.3. Resultados ACO	28
6.4. Menores Fitness para ACO y PSO	32
7. Conclusiones	32
Appendices	34
A. Codigos en Python	34
A.1. Código PSO	34
A.2. Código ACO	39

1. Introducción

Desarrollar proyectos modernos de software por lo general exige una gestión compleja que incluye la planificación, monitoreo y control de tareas y gestión de los riesgos del proyecto. El problema de programación de proyectos de software (SPSP - Software Project Scheduling Problem) busca optimizar la asignación de recursos humanos para el cumplimiento de una o varias tareas (actividades desagregadas dentro del cronograma) con el mínimo costo de salarios y tiempo de duración del proyecto [3]. SPSP es un problema NP-Hard para los administradores de proyectos, caracterizado por combinaciones extremadamente complejas de optimización de la asignación de recursos humanos, consumo de tiempo, costo del proyecto y satisfacción de todas las restricciones relacionadas. Por lo tanto, la programación automática de proyectos de software es muy útil para los administradores de proyectos.

Optimización por Colonia de Hormiga (ACO) y Optimización por Enjambre de Partículas (PSO) son técnicas metaheurísticas que son usados en este informe para resolver el problem SPSP. Para resolver SPSP con ACO se construye un grafo, esta construcción incluye todas las rutas posibles de una tarea a otra, hace uso de la información de las feromonas, en base a la probabilidad de las máximas rutas seleccionadas dentro del grafo, la heurística es calculada usando información previa de las soluciones.

La alta complejidad de la administración de proyectos de software existentes en la actualidad hace necesaria la investigación en herramientas asistidas por computadora para planificar adecuadamente el desarrollo del proyecto. Los proyectos de software actuales generalmente exigen una gestión compleja que implica tareas de planificación, planificación y monitoreo. Es necesario controlar a las personas y los procesos, y asignar recursos de manera eficiente para lograr objetivos específicos mientras se satisfacen una variedad de limitaciones.

Este informe se organiza de la siguiente manera. En la sección 2 se verán los conceptos teóricos importantes a considerar en la implementación de los algoritmos, además de definir el problema a optimizar. En la sección 3 se define el estado del arte, mencionando distintos trabajos hechos en referencia a la gestión de proyectos de software usando Metaheurísticas. En la sección 4 se define la metodología utilizada. En la sección 5 se define la implementación y el desarrollo en el cual se describen los algoritmos utilizados. En la sección 6 se describe la ejecución y los resultados de la prueba de los algoritmos implementados en Python. Finalmente, en la sección 7 se mencionan las principales conclusiones desarrolladas

2. Marco Teórico

2.1. Definición del problema

Para la implementación de Proyectos de Software uno de los principales retos a resolver es la adecuada distribución de recursos con el fin de obtener el mayor beneficio

posible en términos de tiempo, costos y calidad. Uno de los principales componentes a considerar es la gestión de recursos humanos y sus respectivos grados de participación en las diferentes tareas programadas. SPSP es un problema que busca una planificación óptima de recursos y tareas para un proyecto de software, de modo que, las tareas precedentes y restricciones de recursos son cumplidas y el costo final del proyecto, el cual consiste en el salario del personal y la duración del proyecto, son minimizados.

El problema considera como recurso a un grupo de N personas (empleado) con un conjunto de habilidades (*skills*), un salario y una dedicación máxima por día de trabajo como se ve en la Figura 1. Se define a cada miembro del equipo como e_i , i en el rango de 1 hasta N , además se define e_i^{skills} , e_i^{salary} y e_i^{maxded} al conjunto de habilidades, salario y máxima dedicación de un programador respectivamente. Por otro lado como parte del proyecto se tiene un conjunto de T tareas t_j , j en el rango de 1 a T , cada una de las cuales tiene t_j^{skills} y t_j^{effort} , conjunto de requerimientos y dedicación necesarios para completar dicha tarea respectivamente [2]. Finalmente se tiene un Grafo de Precedencia de Tareas (TPG), ver Figura 3, que representa la dependencia entre tareas y una matriz de dedicación o de asignación de recursos $X = x_{ij}$ de tamaño $N \times T$ donde $x_{ij} \geq 0$ que representa la solución consistente en el grado de dedicación que se le asignará al empleado e_i a la tarea t_j [2].

Para este problema lo que se busca es minimizar tanto el tiempo de ejecución como el costo de total de asignación de recursos humanos, lo cual es mostrado en las siguientes ecuaciones:

$$Min \sum_{i=1}^N \sum_{j=1}^T e_i^{salary} . x_{ij} . t_j^{dur} \quad (1)$$

$$Min \sum_{j=1}^T t_j^{dur} \quad (2)$$

s.a

$$\sum_{i=1}^N x_{ij} > 0 \quad \forall i \in \{1, 2, \dots, N\} \quad (3)$$

$$t_j^{skills} \subseteq \bigcup_{i|x_{ij}>0} e_j^{skills} \quad \forall j \in \{1, 2, \dots, T\} \quad (4)$$

La *fórmula* 1 busca minimizar el costo total expresado como la multiplicación del salario de un trabajador, su grado de participación en una determinada tarea y el tiempo que lleva completarla. La *fórmula* 2 expresa la suma de todos los tiempos de las tareas que conforman el proyecto. La *fórmula* 3 es una condicional de que todas las tareas son asignadas como mínimo a una persona, por lo cual la suma de cada columna de la matriz X debe ser mayor a cero. Finalmente la *fórmula* 4 expresa el requerimiento de que el conjunto de skills pedidos por una tarea están incluidos en la unión de los requisitos de los programadores seleccionados.

$$t_j^{dur} = \frac{t_j^{effot}}{\sum_{i=1}^N x_{ij}} \quad (5)$$

Adicionalmente se define al t_j^{dur} como el tiempo que toma en concluir una tarea y está definido como la división entre el esfuerzo requerido y la suma de las dedicaciones de cada miembro seleccionado [2].

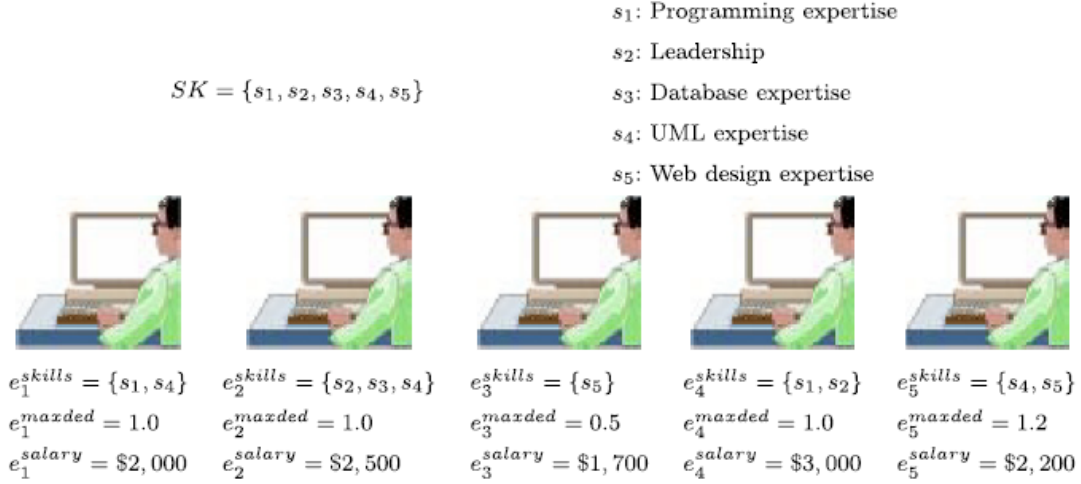




Figura 1: Ejemplo de un Staff de programadores con todos sus características

Como se muestra en la figura 2 la matriz X representa la dedicación que cada programador le dará a una determinada tarea del proyecto.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7
e_1	1.00	1.00	1.00	0.00	0.71	0.29	1.00
e_2	1.00	1.00	1.00	0.00	1.00	0.00	1.00
e_3	0.57	0.14	0.29	0.29	0.29	0.29	0.43
e_4	1.00	1.00	1.00	0.00	0.57	0.57	1.00
e_5	1.00	0.86	0.86	0.29	1.00	0.14	1.00

Figura 2: Matriz solución X

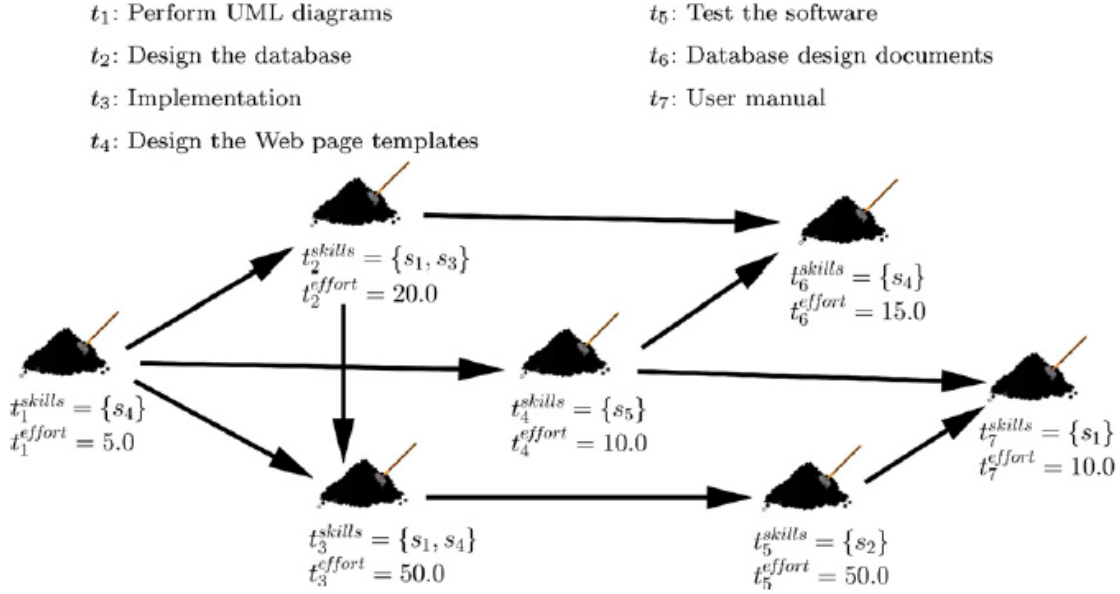


Figura 3: Ejemplo de Grafo de dependencia de tareas (TPG) y sus características

2.2. Solución y Normalización de Funciones Multiobjetivo

Cuando se trata de problemas de optimización, lo más común generalmente es trabajar con una sola función objetivo lo cual también es conocido como *single-objective optimization* u optimización con objetivo simple. Sin embargo en la práctica diseñar problemas involucra tratar con más de una función objetivo las cuales en muchos casos pueden entrar en conflicto entre ellas [1]. Matemáticamente define una función multiobjetivo con k funciones componente como:

$$f(x) = (f_1(x), f_2(x), \dots, f_k(x)) \quad (6)$$

Existen varios métodos para trabajar optimización con funciones multiobjetivo, entre ellos uno de los más simples es el método de suma ponderada el cual consiste en optimizar reduciendo todo a una sola función objetivo F generada por la suma de cada función componente multiplicada por su respectivo peso w_i [1].

$$F = \sum_{i=1}^k w_i f_i(x) \quad (7)$$

Donde $w_i > 0$ y $\sum_{i=1}^k w_i = 1$ en cierto sentido los pesos servirán como factores de escalamiento para las funciones objetivo.

Muchos métodos de optimización multiobjetivo implican comparar y tomar decisiones sobre diferentes funciones objetivas. Sin embargo, los valores de diferentes funciones pueden tener diferentes unidades y/o órdenes de magnitud significativamente

diferentes, lo que dificulta las comparaciones. Por lo tanto, generalmente es necesario transformar las funciones objetivas de modo que todas tengan órdenes de magnitud similares. Aunque se proponen diferentes enfoques para tal propósito, uno de los enfoques más simples es normalizar las funciones objetivas individuales dividiéndolas entre el máximo teórico [5].

$$f_i^{norm}(x) = \frac{f_i(x)}{f_i^{max}} \quad (8)$$

Donde f_i^{max} representa el valor máximo de la función f_i , esto produce una función con valor no dimensional con $f_i^{max} \leq 1$.

2.3. Algoritmo Particle Swarm Optimization (PSO)

El algoritmo PSO consiste básicamente en generar una población (denominada enjambre) de individuos que representan a cada una de las posibles soluciones candidatas a un problema denominados partículas. Dichas partículas se mueven por un espacio de búsqueda en busca de la mejor posición individual y su mejor posición global, este proceso se realiza múltiples veces con la esperanza de ubicar la mejor posición global del enjambre. Los movimientos son definidos usando funciones matemáticas definidas en [6] y el cálculo del óptimo se realiza en base a una función fitness definida en cada caso. A continuación, se describen algunos elementos importantes.

Se le denomina partícula a cada miembro del enjambre. El objetivo de cada partícula es dirigirse hacia la mejor posición individual y a la mejor posición global en el espacio de búsqueda [6]. Para nuestro caso cada partícula generará una matriz X solución con los grados de participación de cada empleado.

La mejor posición de una partícula se puede definir como una coordenada la cual permite que la partícula obtenga el mejor valor al aplicarle la función fitness. Por ejemplo, en el caso estudiado en el presente informe la mejor posición permitirá crear una asignación de recursos a las tareas, en el cual el tiempo y el costo sea menor.

La función fitness del algoritmo permite determinar cuál es la partícula que genera el mejor candidato según los criterios establecidos para definirla, generalmente toma como entrada las coordenadas de la partícula y devuelve un valor que representa la factibilidad de un candidato y adicionalmente se pueden establecer reglas para controlar los valores o cambios que experimenta el desplazamiento de partículas con los cuales se logran las restricciones definidas.

2.4. Algoritmo Ant Colony Optimization (ACO)

El algoritmo de colonia de hormigas es un algoritmo heurístico, propuesto por primera vez por el erudito italiano Dorigo [4]. El algoritmo de colonia de hormigas se inspira en la observación del comportamiento de búsqueda de alimento de hormigas en la naturaleza. Durante el proceso de búsqueda de alimento, las hormigas siempre pueden encontrar una hormiga esclava quien encontró un camino más corto desde el agujero

hasta la comida. Aunque las hormigas individuales son extremadamente simples y no inteligentes, la colonia de hormigas puede confiar en una sustancia química llamada feromona para lograr una inteligencia de enjambre durante el proceso de búsqueda de alimento. Inspirado por esto, el algoritmo de la colonia de hormigas simula el comportamiento de búsqueda de alimento de hormigas en la naturaleza, usan las feromonas como una pista para la optimización y confían en la inteligencia de enjambre de las hormigas para resolver varios problemas de optimización combinatoria. Dado que el algoritmo de colonias de hormigas se ha propuesto durante más de 20 años, ha habido una variedad de diferentes variantes de algoritmos, de las cuales las más utilizadas son el algoritmo AS (sistema de hormigas), el algoritmo ACS (sistema de colonias de hormigas) y el MMAS (min-max ant system) algoritmo. [4].

ACO estudia sistemas artificiales que toman la inspiración del comportamiento de las colonias de hormigas y son utilizadas por primera vez para resolver problemas de optimización discretas, especialmente problemas complejos de optimización combinatoria. La metaheurística de ACO se caracteriza por ser distribuida, el método de búsqueda estocástico se basa en las comunicaciones indirectas de una colonia de hormigas artificiales, influenciados por rastros artificiales de feromonas. Las hormigas artificiales modifican el rastro de las feromonas durante la ejecución del algoritmo. También fue adaptado para resolver problemas de optimización continua, en esos casos el rendimiento de ACO no es tan bueno como la optimización en casos discretos. ACO ha sido ampliamente utilizado para resolver problemas discretos de optimización tales como problemas NP-Hard, problemas dinámicos del camino más corto dinámico y problemas donde la arquitectura computacional es distribuida de manera. En este informe, aplicamos ACO para resolver el problema planificación de recurso y tareas de proyectos de software.

3. Estado del Arte

Existen varios trabajos relacionados a la aplicación de metaheurísticas y heurísticas al planeamiento de Proyectos y Gestión de Recursos Humanos en proyectos de Software. Algunos de ellos consideran solo el *staffing* (Distribución de trabajos a los miembros del equipo), sin embargo, otros trabajos realizan una combinación junto a *cheduling* para mejora y afinar dicha distribución, se mencionan algunos de ellos, los cuales sirvieron de guía para el presente trabajo:

1. ***Software project management with GAs***: Este estudio del año 2006, utiliza algoritmos genéticos para la implementación de una distribución de trabajos en base a las habilidades requeridas para las tareas, la dedicación que cada programador le pone al trabajo y los sueldos del personal. Se busca minimizar tanto el **costo total del proyecto** como el **tiempo de ejecución total** sujeto a condicionales como los requisitos a cumplir o la distribución de una tarea a por lo menos una persona. Adicionalmente definen un grafo o matriz de dependencias de tareas el cual sirve de guía inicial [2].

2. ***Team Staffing in Software Development using Particle Swarm Optimization***: Este estudio del año 2012, emplea el algoritmo de Enjambre de Partículas (PSO) para contruir la distribución de trabajos entre los miembros del equipo. De manera similar a la referencia anterior, para evaluar la calidad de una partícula o solución considera una suma ponderada entre el valor del fitness (a su vez depende del tiempo de duración y los requisitos cumplidos) y el cumplimiento de las condicionales (establece una fórmula matemática para darle un valor numerico de penalidad por cada condicional no cumplida) [6].
3. ***Dynamic Staffing and Rescheduling in Software Project Management***: Este estudio del año 2016, utiliza un combinación de Algoritmos Genéticos (GA) con Hill Climbing (HC) para la creación del proyecto de Software. Al igual que los casos anteriores considera los tiempos, costo total y restricciones para implementar las funciones objetivo o de calidad, pero adicionalmente también considera algunos parametros como calidad del trabajo (en base a la experiencia en un determinado requisito) o la capacidad de aprender de los programadores para poder realizar trabajos nuevos [7].

4. Metodología Utilizada

La metodología seguida en el presente trabajo describe a grandes rasgos los pasos que se siguieron para implementar el presente trabajo y se definen en las siguientes partes:

- **Formulación y Planteamiento del Problema**: En esta parte se define el problema principal y los posibles medios para su solución. En nuestro caso implementación de metaheurísticas para lograr una adecuada Gestión de Recursos Humanos para proyectos de Software.
- **Revisión de Bibliografía y documentación**: El siguiente paso fue buscar referencias y documentación sobre el tema, en base a la documentación encontrada se definieron los diferentes enfoques, condiciones y elementos con los cuales otros autores resuelven este caso.
- **Selección de algoritmos a utilizar**: Seguidamente se procedió a seleccionar las heurísticas y metaheurísticas a utilizar basándose principalmente en la complejidad de implementación y la claridad y disponibilidad de información acerca de ello.
- **Implementación de los algoritmos**: Una vez definidos los algoritmos PSO y ACO multiobjetivo, se procedió a implementarlos en Python. Se seleccionó este lenguaje dado su facilidad y simpleza para desarrollo.

- **Ejecución y Pruebas:** Una vez terminada la implementación de los algoritmos se procedió a probar su eficiencia utilizando casos de prueba concretos los cuales se definirán en la parte de ejecución y resultados. Esto permitió ir depurando las implementaciones y a su vez obtener información de tiempos y resultados obtenidos para hacer las respectivas comparaciones.
- **Elaboración de conclusiones:** Una vez obtenidos los resultados se procede a realizar comparaciones y analizar los datos obtenidos que junto a la documentación permiten dar las conclusiones finales del trabajo.

5. Implementacion y Desarrollo

5.1. Implementación con PSO

A continuación, se describe la implementación de PSO para el problema definido. Se procede a implementar las ecuaciones planteadas en la sección 2 dentro del entorno de programación en Python y siguiendo los pasos generales del algoritmo Enjambre de Partículas.

Para la implementación se tienen en cuenta las siguientes especificaciones generales:

- Cada una de las *partículas* del enjambre generará una matriz X solución con los respectivos valores de dedicación que ponen los empleados a las tareas. De forma similar a la figura 2.
- Se considera que solo los empleados que tienen alguna experiencia en los requisitos requeridos por cada tarea sean los que participen, por lo que se define una matriz de participación M que controlará los componentes de la matriz X que se mantendrán en 0.
- Al ser un problema multiobjetivo de minimizar el *Costo Total* y *Duración Total* se utilizó el método de suma ponderada normalizada según la formula 7 para calcular el valor del fitness de las partículas.
- La información inicial es recolectada de Archivos *.CSV* y guardados en DataFrames para su posterior uso. Esta información consiste en: detalles de los requerimientos, información de los empleados e información de las tareas.
- El cálculo de las velocidades y posiciones de las partículas se rigen a las ecuaciones comunes del Algoritmo PSO, las cuales son:

$$v_{t+1}(i) = w.v_t(i) + c_1.r_1.(pbest_t(i) - x_t(i)) + c_2.r_2.(gbest_t(i) - x_t(i)) \quad (9)$$

$$x_{t+1}(i) = x_t(i) + w.v_{t+1}(i) \quad (10)$$

Según las especificaciones planteadas anteriormente se procede a implementar el algoritmo el cual está compuesto de algunas funciones principales las cuales detallamos a continuación.

1. **Función TiempoTotal:** Según la ecuación 2, el tiempo total es calculado como la suma de todos los tiempos que toma cada tarea t_i . A su vez el tiempo de cada tarea según la ecuación 5 se calcula como la división del esfuerzo total requerido y la suma de esfuerzos de los empleados asignados $t_i^{effort} / \sum_{i=1}^N X_{ij}$,

Algorithm 1 función TiempoTotal

```

1: Entrada: partícula  $P_i$ 
2: Salida: Tiempo
3:  $X \leftarrow$  Matriz generada por  $P_i$ 
4:  $esfuerzoTarea \leftarrow []$ 
5:  $Tiempo \leftarrow 0$ 
6: for cada  $x_j$  de  $X$  do
7:   /*Calcular la suma de todos los valores de la columna */
8:    $esfuerzoTarea_j \leftarrow \text{sum}(x_j)$ 
9: end for
10: for cada  $t_j$  en tareas do
11:   /*calcular tiempo de ejecución  $t_{dur}$  como la división entre el
12:   esfuerzo requerido y el  $esfuerzoTarea_i$ */
13:    $t_{dur} \leftarrow t_j^{requerido} / esfuerzoTarea_j$ 
14:    $Tiempo \leftarrow Tiempo + t_{dur}$ 
15: end for

```

2. **Función CostoTotal:** Según lo planteado en la formula 1, el costo total se calcula como la suma de costo de cada empleado en cada tarea en donde participará. Esto se calcula multiplicando el salario de cada participante, su participación según la matriz X y el tiempo de duración de la tarea t_i .

Algorithm 2 función CostoTotal

```
1: Entrada: particula  $P_i$ 
2: Salida: Costo
3:  $X \leftarrow$  Matriz generada por  $P_i$ 
4: Costo  $\leftarrow 0$ 
5: for cada  $t_j$  en tareas do
6:   for cada  $e_i$  en empleados do
7:     /*calcular el salario de cada empleado segun la ecuación 1
8:      $salario_j \leftarrow t_i^{salario} \cdot X_{ij} \cdot t_j^{dur}$ 
9:   end for
10: end for
11:  $Costo \leftarrow \text{sum}(salario_j)$ 
```

3. **Función para verificar si se cubren todos los requisitos de las tareas:** Esta función se encarga de verificar si la matriz X solución generada permite cubrir todos los requisitos de cada una de las tareas. Esto permitirá descartar algunas soluciones que no cumplan con la restricción de la formula 4

Algorithm 3 función requisitosReunidos

```
1: Entrada: particula  $P_i$ 
2: Salida: CumpleRequisitos
3:  $X \leftarrow$  Matriz generada por  $P_i$ 
4: CumpleRequisitos  $\leftarrow$  False
5: listaskills  $\leftarrow []$  ▷ lista de requisitos reunidos para cada para tarea
6: for cada  $x_{ij}$  en  $X$  do
7:   if  $x_{ij} > 0$  then ▷ El programador i participa de la tarea j
8:     adicionar a  $listaskills_j$  la lista de skills del programador  $i$ 
9:   end if
10: end for
11: Generar una lista de conjuntos de skills  $conjuntoskills$  por cada tarea en base a  $listaskills$  ▷ Se eliminar requisitos duplicados y otros
12:  $skilltareas \leftarrow$  genera el conjunto de requisitos pedidos por cada tarea.
13: for cada  $tarea_j$  en tareas do
14:   if  $skilltareas_j$  esta incluido en  $conjuntoskills_j$  then
15:     CumpleRequisitos  $\leftarrow$  True
16:   else
17:     CumpleRequisitos  $\leftarrow$  False
18:   end if
19: end for
```

4. **Función Fitness:** Al ser un problema multiobjetivo de minimizar el costo total y el tiempo total del proyecto, se emplea el método de la suma ponderada normalizada asociando un peso o importancia a cada función objetivo según las formulas 7 y 8. Para nuestro problema se define la función fitness:

$$f(p_i) = w_{cost} \times (f_{cost}(p_i)/\max(f_{cost})) + w_{dur} \times (f_{dur}(p_i)/\max(f_{dur})) \quad (11)$$

Donde:

w_{cost} = Peso/importancia del costo.

$f_{cost}(p_i)$ = Costo total del proyecto producido por la partícula p_i .

w_{dur} = Peso/importancia de la duración.

$f_{dur}(p_i)$ = Duración total del proyecto producida por la partícula p_i .

$\max(f_{dur})$ = Valor máximo de la función de duración.

$\max(f_{cost})$ = Valor máximo de la función de costo.

Algorithm 4 función Fitness

- 1: **Entrada:** partícula P_i
 - 2: **Entrada:** w_{cost} y w_{dur} pesos para la función fitness
 - 3: **Salida:** valor de fitness
 - 4: fitness $\leftarrow 0$
 - 5: $costo_{maximo} \leftarrow$ Costo máximo hallado corriendo 100 veces con pesos 0.5, 0.5
 - 6: $duracion_{maxima} \leftarrow$ Duración máxima hallada corriendo 100 veces con pesos 0.5, 0.5
 - 7:
 - 8: fitness $\leftarrow w_{cost} \cdot CostoTotal(P_i)/costo_{maximo} + w_{dur} \cdot TiempoTotal(P_i)/duracion_{maxima}$
-

5. **Función test PSO:** Esta función implementa el algoritmo en si el cual recibe como entrada la cantidad de partículas, los pesos para la función fitness y la matriz de de participación, según esto se siguen los siguientes pasos:

- a) Generar un conjunto de partículas.
- b) Inicializar, posición, mejor posición y velocidad inicial aleatoria con un rango entre 0 y máxima capacidad del respectivo empleado.
- c) Evaluar la función fitness en cada partícula y actualizar las posiciones optimizadas global y local. Según la matriz de participación solo modificar de los que cumplen los requisitos en determinada tarea sino dejarlo en 0.
- d) Actualizar la posición y velocidad según las ecuaciones mostradas en las fórmulas 9 y 10.
- e) retornar la mejor posición global al cual genera a la mejor matriz X de participación.

Algorithm 5 Test PSO

```
1: Entrada: Número de partículas  $m$ 
2: Entrada: Matriz  $\mathbf{M}$  de participación
3: Entrada:  $w_{cost}$  y  $w_{dur}$  pesos para la función fitness
4: Salida:  $P_g$ , fitness( $P_g$ )
5: for cada partícula  $i$  do
6:   if  $M_{p,q} == \text{True}$  then
7:     Inicializar las las coordenadas y velocidad de los componentes
8:     con índice p y q de de  $\vec{P}_i$  de forma aleatoria
9:   end if
10: end for
11: while  $NC \leq NC_{max}$  do
12:   for cada partícula  $i$  do
13:     Evaluar fitness  $f(\vec{X}_i, w_{cost}, w_{dur})$ 
14:     if  $f(\vec{X}_i, w_{cost}, w_{dur}) < f(\vec{P}_i, w_{cost}, w_{dur})$  then
15:        $\vec{P}_i \leftarrow \vec{X}_i$ 
16:     end if
17:     if  $f(\vec{X}_i, w_{cost}, w_{dur}) < f(\vec{P}_g, w_{cost}, w_{dur})$  then
18:        $\vec{P}_g \leftarrow \vec{X}_i$ 
19:     end if
20:   end for
21:   for cada partícula  $i$  do
22:     if  $M_{p,q} == \text{True}$  then
23:       Modificar la velocidad según la fórmula de la fórmula 9 de los
24:       componente con índice p y q de de  $\vec{P}_i$ 
25:       Modificar la posición según la fórmula de la fórmula 10 de los
26:       componente con índice p y q de de  $\vec{P}_i$ 
27:     end if
28:   end for
29: end while
```

6. **Función de comparación de pesos para fitness:** Esta función implementa un cálculo de PSO con diferentes pesos w_i para las funciones de costo y duración. En base a los datos producidos se obtiene el mejor a criterio propio o basado en el fitness.

Algorithm 6 función Iterador PSO

```
1: Entrada: Número de partículas  $m$ 
2: Entrada:  $t_{max}$  Tiempo máximo de proyecto
3: Cargar toda la data en los DataFrames correspondientes
4:  $M \leftarrow \text{generarMatrizParticipacion}()$ 
5:  $\text{listaW} = [0.0, 0.1, \dots, 0.9, 1.0]$ 
6:  $\text{listaFitness} \leftarrow []$ 
7:  $\text{listaSoluciones} \leftarrow []$ 
8:  $\text{listaSCostos} \leftarrow []$ 
9:  $\text{listaSoDuracion} \leftarrow []$ 
10: for  $w_{cost}$  en  $\text{listaW}$  do
11:    $X, \text{fitness} \leftarrow \text{testPSO}(m, M, w_{cost}, 1-w_{cost})$   $\triangleright w_{dur} = 1-w_{cost}$ 
12:   Calcular el costo y duracion generada por la solución X
13:   if todas las tareas fueron distribuidas y se cubren todos sus requisitos then
14:      $\text{listaFitness.append}(\text{fitness})$ 
15:      $\text{listaCostos.append}(\text{costo})$ 
16:      $\text{listaDuracion.append}(\text{duracion})$ 
17:   end if
18: end for
19: Mostrar solución con el mejor valor de Fitness o a criterio del analista
```

5.2. Implementación con ACO

Para la implementación de ACO para SPSP requiera la construcción de grafo, diseño del comportamiento de las feromonas e información de la heurística. En esta sección en desarrollaremos cada uno de estos temas.

5.2.1. Diseño de Grafo para la Tarea

El primer paso para aplicar ACO, es buscar una manera eficiente de distribuir los esfuerzo o dedicaciones de cada empleado dentro de la búsqueda de las mejores rutas por las hormigas, basado en el modelo de actualización de las feromonas y la información heurística. Cada empleado puede contribuir con su dedicación a cada tarea del proyecto. Entonces, en la práctica el comportamiento la búsqueda de las mejores rutas por las hormigas estará dentro de cada tarea en lugar de seguir las rutas de precedencia de las tareas (TPG). Esto significa que la dedicación de cada empleado es determinada en cada tarea, esta estrategia es construida de la siguiente manera: cada tarea del TPG es dividida en un grafo de vértices y aristas, $G(V, A)$ donde los vértices $V = \{d_1, d_2, \dots, d_v\}$ representan a los nodos de dedicación de un empleado e_i y las aristas $A = \{(d_i, d_j), \dots, (d_n, d_m)\}$ representan a los caminos/rutas de comunicación entre los nodos de todos los empleados dentro en la misma tarea t_k , como se puede ver en el ejemplo de la Figura 4, describe la construcción de un grafo para la t_1 . Los pasos

seguidos para la construcción del grafo son los siguientes:

1. Determinar la densidad de los nodos (número de nodos), dado por los niveles de dedicación a una tarea, esto puede generarse a criterio del implementador o usando la fórmula $den = min_ded^{-1} + 1$, en ejemplo se puede ver min_ded (mínima dedicación) es 0,25 y den (densidad) es 5.
2. Generar un nodo inicial para la *columna* 0.
3. De acuerdo al número de empleados generar N columnas de nodos: *columna* 1, *columna* 2, ..., *columna* N . Cada columna contiene den número nodos.
4. Generar un nodo final en la *columna* $N + 1$. El nodo inicial y final son referenciales, no representa valores en el algoritmo implementado.
5. Generar todas las rutas entre los nodos desde *columna* 0 hasta *columna* $N + 1$.

Después de la división dentro de la tarea, la hormiga empieza su recorrido en el nodo inicial y selecciona las rutas de viaje desde *columna* 1 hasta *columna* N . La hormiga solo puede seleccionar un nodo por cada columna, de esta forma hasta llegar al nodo final. La dedicación práctica de cada nodo está definido por el valor de la posición del nodo dentro de la columna, para ejemplo, el $nodo_1 = 0, nodo_2 = 0,25, \dots, nodo_5 = 1$. En general el viaje de la hormiga dentro de una tarea, no tiene nada que con la relación de precedencias de las tareas (TPG), la hormiga pasa a la siguiente tarea según un orden aleatorio (orden de ingreso). Al respecto, un empleado tiene que evaluar todas las tareas, y de acuerdo a las habilidades que necesita la tarea y a las que dispone se aplicará la dedicación que ha seleccionado la hormiga.

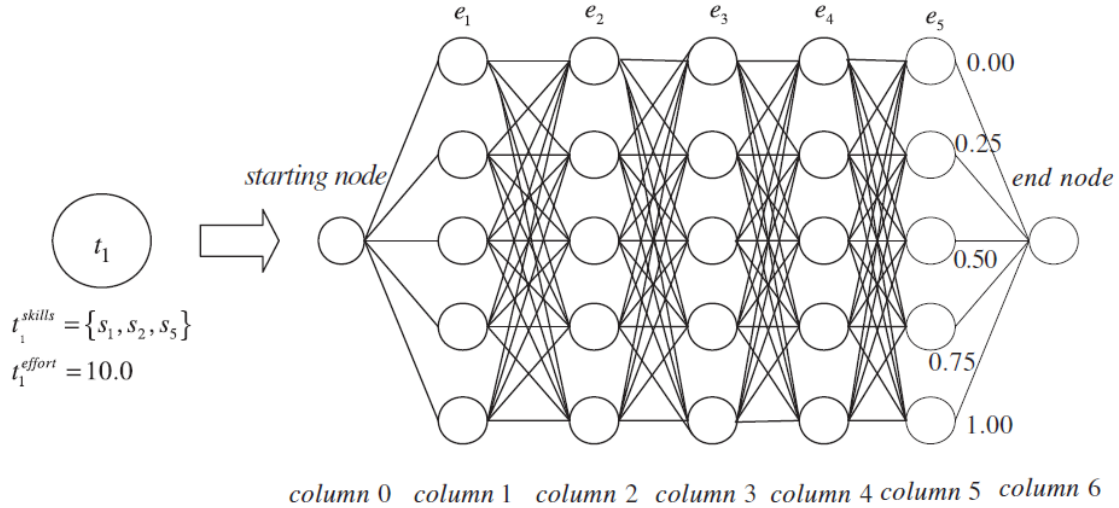


Figura 4: Ejemplo de grafo generado de una tarea (t_1).

5.2.2. Implementación de ACO para SPSP

El algoritmo implementado es presentado en el *Algoritmo 8*, el detalle se describe a continuación:

1. **Inicializar de los parámetros del proyecto.** Lectura de toda la información necesaria del proyecto: staff(salario,máximo esfuerzo), skills, tasks(esfuerzo necesario, plazo máximo), staff x skill, tasks x skill, tasks x predecet (TPG), dedication x strategy.
2. textbfInicialización de los siguientes parámetros:

Tabla 1: *Tabla de parámetro iniciales del algoritmo.*

Parámetro	Descripción
N_{ant}	Número de hormigas
α	Coeficiente para el control de la influencia/- peso de las feromonas
β	Coeficiente para el control de la influencia/- peso de la heurística
ρ	Tasa de volatilidad de las feromonas
τ	Valor inicial de las feromonas
q_0	Valor que permite balancear (regla de proporcionalidad aleatoria) entre la exploración y la explotación
N_{gen}	Máximo número de generaciones ACO

3. **Inicializar de la matriz de feromonas en τ .** La matriz de feromonas es una matriz en 3D; consiste en valores para cada tarea, cada empleado y cada nodo de dedicación. Cada hormiga tiene una matriz para guardar su solución, estos valores se inicializan en 0,0 y se, a medida que realiza el viaje, se asignarán los valores.
4. **La hormiga seleccionada el camino para la solución.** Cada hormiga en el viaje por el grafo donde cada tarea está divida en base a la dedicación por cada empleado, selecciona nodos (empleado, dedicación) y asigna los valores a la matriz solución. Después de que el recorrido por la tarea ha sido completado, la hormiga mantiene el viaje a través de la siguiente tarea. Cuando finaliza el recorrido por todas las tareas, la matriz solución está completa.
5. **Evaluar la factibilidad de la solución.** Se consideran las siguientes reglas:
 - Asegurar que cada tarea este a cargo por mínimo un empleado.

- Validar que la unión de las habilidades de todos los empleados dedicados a una tarea t_j , contenga las habilidades requeridas por la tarea.

Algorithm 7 Función de factibilidad ACO

```

1: Input: Matriz solución candidata
2: Output: True si cumple las reglas
3: for  $x_{ij}$  in CandidateSolution do
4:   if Sum( $x_{ij}$ )  $\neq$  0 then
5:     return False
6:   end if
7:   if Skills( $t_i$ ) not in Skills( $e_{ji}$ ) then
8:     return False
9:   end if
10: end for
11: return True

```

6. **Evaluar el costo, duración en la función fitness.** La función fitness consiste en minimizar la siguiente función:

$$f(x) = w_{cost} \times p_{cost}/p_{cost}^{max} + w_{dur} \times p_{dur}/p_{dur}^{max} \quad (12)$$

Donde:

w_{cost} = Peso/importancia del costo.

p_{cost} = Costo total del proyecto.

w_{dur} = Peso/importancia de la duración.

p_{dur} = Duración total del proyecto.

p_{dur}^{max} = Valor máximo de la función de duración.

p_{cost}^{max} = Valor máximo de la función de costo.

7. **Evaluar el costo, duración y Actualización de los valores de las matriz de feromonas.** Todas las rutas seleccionadas en cada tarea, cualesquiera sean los nodos, actualizan de manera local las feromonas. La mejor solución, después de recorrer todas las tareas, actualiza de manera global las feromonas.
8. **Viaje de las hormigas a la siguiente generación.** Las hormigas realizan un nuevo viaje, usando feromonas que han sido actualizadas en recorridos anteriores por otras hormigas, y repetirán el mismo proceso desde el paso 4 hasta 7 hasta terminar la condición de parada. La condición de parada es el parámetro inicial de número de generaciones.
9. **Obtener la mejor solución en términos de la función fitness.** Obtener la mejor matriz solución entre los mejores candidatos de cada generación (mejor solución en función del costo, duración y sobretiempo).

Algorithm 8 Algoritmo ACO

```
1: Input: Tablas de (skills, staff, precedent, strategy)
2: Input: Parámetro  $(\alpha, \beta, \rho, \tau, q_0, N_{gen})$ 
3: Input:  $w_{cost}$  y  $w_{dur}$  pesos para la funcion fitness
4: Output: Mejor solución  $S_{best}$  y objetivo  $G_{best}$  (fitness, cost, duration, TPG scheduler)
5: Cargar todos los datos del proyecto
6: Cargar todos los parámetros de ACO
7: Cargar los pesos de la función fitness
8: Generar la matriz de feromonas
9: Generar una solución inicial aleatoria y almacenar el costo, duración y sobretiempo
10: Generar la construcción de grafos por cada tarea usando algoritmo de división
    Algoritmo 9
11:  $gen \leftarrow 0$ 
12: while NO hay Mejor Solución or  $gen \leq N_{gen}$  do
13:   for  $gen \leftarrow 0$  to  $N_{ant}$  do
14:     for  $t \leftarrow 0$  to  $T_{task}$  do ▷ Hormiga selecciona ruta
15:       Generar matriz de heurística
16:        $q \leftarrow random(0, 1)$ 
17:       if  $q > q_0$  then
18:         Explora nueva ruta (heurística)
19:       else
20:         Explora ruta (feromonas, heurística)
21:       end if
22:       Actualizar localmente las feromonas
23:     end for
24:      $Actual_{solucion} \leftarrow$  Generar solución candidata
25:     Calcular costo de la solución
26:     Calcular duración y sobretiempo de la solución
27:     if Validar función de factibilidad = True then
28:       if  $Actual_{fitness} < Mejor_{fitness}$  then ▷ Minimizar
29:          $Mejor_{cost} \leftarrow Actual_{cost}$ 
30:          $Mejor_{duracion} \leftarrow Actual_{duracion}$ 
31:          $Mejor_{sobretiempo} \leftarrow Actual_{sobretiempo}$ 
32:          $Mejor_{solucion} \leftarrow Actual_{solucion}$  ▷ Mejor matriz solución
33:         Actualizar globalmente las feromonas
34:       end if
35:     end if
36:   end for
37:    $gen \leftarrow gen + 1$ 
38: end while
39: return  $Mejor_{solucion}, Mejor_{cost}, Mejor_{duracion}, Mejor_{sobretiempo}$ 
```

Algorithm 9 Algoritmo de división (Split)

```
1: Input: Total tareas  $T$ , Total empleados  $N$ , y valor densidad  $den$ 
2: Output: Matriz grafo del proyecto
3:  $j \leftarrow 0$ 
4: for  $task_i \leftarrow 0$  to  $T_{task}$  do
5:   Asigna  $task_i$  a la  $Columna_j$ 
6:   for  $e_i \leftarrow 0$  to  $N_{staff}$  do
7:     Crear  $Columna_j$ 
8:      $q \leftarrow random(0, 1)$ 
9:     for  $densidad_i \leftarrow 0$  to 1 do
10:      Crear todos los caminos desde  $task_i$  a la  $Columna_j$ 
11:    end for
12:  end for
13:  Agregar la  $Columna_j$ 
14:  Construir todos los caminos desde  $Columna_{j-1}$  a  $Columna_j$ 
15:  return Matriz de grafo del proyecto
16: end for
17: return Matriz de grafo del proyecto
```

En la *Figura 5*, se puede visualizar un ejemplo de matriz de asignación generada por el algoritmo.

	t_1	t_2	t_3	t_4	t_5	t_6
e_1	1.00	0.50	0.00	0.75	0.00	0.00
e_2	0.00	0.75	0.25	0.00	1.00	0.50
e_3	1.00	0.50	0.50	0.25	0.75	0.50
e_4	0.25	0.75	0.25	0.00	0.75	0.00
e_5	0.25	0.00	0.00	1.00	0.00	0.00

Figura 5: Matriz ejemplo de asignación de recurso (M_{ij}).

5.2.3. Administración de Feromona

Todas las hormigas al finalizar el recorrido por los nodos de dedicación por empleado de una tarea, actualizan el valor de las feromonas de la siguiente manera:

1. **Actualización Local.** La actualización local es aplicada a la ruta selecciona, por la hormiga, dentro de la tarea examinada. Entonces, la actualización local aplica solo a la tarea, no necesita la finalización del recorrido por todas las tareas.

$$\tau_{ij} = (1 - \rho) \times \tau_{ij} + \rho \times \Delta\tau \quad (13)$$

Donde:

$$\Delta\tau = (w_{cost} \times t_{cost}^k + w_{dur} \times (t_{dur}^k + t_{over}^k))^{-1} \quad (14)$$

Donde, ρ , $0 < \rho < 1$, es el factor de influencia sobre las feromonas y $\Delta\tau$ es la compensación por la calidad de la solución actual. Se observa que los valores t_{cost}^k , t_{dur}^k y t_{over}^k corresponde solo a la tarea k .

2. **Actualización global.** La actualización global es aplicada solo a la **mejor ruta** selecciona, por la hormiga, en la evaluación de todas las tareas. La actualización es similar a la fórmula 13. Para el caso del $\Delta\tau$ se hará uso de la información de todo el proyecto, esta información es generada por la mejor solución, hasta el momento, de todas las generaciones.

$$\Delta\tau = (w_{cost} \times p_{cost} + w_{dur} \times (p_{dur} + p_{over}))^{-1} \quad (15)$$

5.2.4. Selección del Mejor Nodo

En consecuencia a lo descrito en los punto anteriores, cuando una hormiga recorre un camino, selecciona cada nodo n_{ij} de cada empleado e_i dentro de la tarea t_k usando la probabilidad determina en la *fórmula* 16. Cabe precisar, que de la matriz de probabilidades $p_k(i, j)$ se escoge a los mejor probabilidad (máximo valor).

$$p_k(i, j) = \frac{[\tau(i, j)]^\alpha \times [\eta(i, j)]^\beta}{\sum_{s=1}^{den} [\tau(i, s)]^\alpha \times [\eta(i, s)]^\beta} \quad (16)$$

Donde, $\tau(i, j)$ en la feromona en el nodo de dedicación n_j en la tarea t_k del empleado e_i y $\eta(i, j)$ es la información de la heurística del problema. Estos valores son positivos y menores que uno, la información heurística es más importante cuando α es menor que β y viceversa.

5.2.5. Información de la Heurística

La información de la heurística se genera en base a la dedicación asignada en las tareas evaluadas, de esta manera se considera las contribuciones realizadas por el empleado en las otras tareas y si el empleado ha dedicado más esfuerzo en tareas previas, el empleado debería dedicar menos esfuerzo a la tarea actual. La información heurística se determina por la siguiente *fórmula*.

$$\eta(i, j) = \begin{cases} Temp[den - i - 1]/TOTAL & \text{si } AllocDec[K] > 0,5 \\ Temp[i]/TOTAL & \text{caso contrario} \end{cases} \quad (17)$$

Donde:

$$Temp[i] = \begin{cases} Dedi[i] + AllocDec[K] - 0,5 & \text{si } AllocDec[K] > 0,5 \\ Dedi[i] + AllocDec[K] & \text{caso contrario} \end{cases} \quad (18)$$

$$TOTAL = \sum_{i=0}^{den} Temp[i] \quad (19)$$

$Temp[i]$ es una matriz temporal de valores de los nodos candidatos $i = (1...den)$, $Dedi[i]$ es una matriz de las dedicaciones por cada nodo candidato y $AllocDec[K]$ es una matriz de dedicaciones asignadas por cada empleado.

Un tema bien importante acerca de generación de la heurística, es acerca, que esta incluye una penalización en caso detecte a un empleado que no tenga como mínimo una habilidad (skill) que requiere la tarea examinada, sugiriendo a la hormiga el nodo con dedicación 0 %.

5.2.6. Función Fitness y Pesos

Como se puede observar en el punto 6 de la sección 5.2.2, la función fitness consiste en minimizar la suma ponderada de los objetivos (costo y duración) dada por la *fórmula* 12. En este informe se asignará pesos de importancia iguales para el costo y duración del proyecto. Sin embargo, se tendrá en cuenta el orden de magnitud de ambos objetivos. Esto se logrará configurando los pesos iguales $w_{cost} = 0,5$, $w_{dur} = 0,5$ y ejecutar el algoritmo por cien (100) veces. Luego, la función objetivo del costo será dividido por el valor máximo de costo obtenido de los datos generados, para la función objetivo de la duración del proyecto se seguirá el mismo procedimiento. De esta forma los objetivos de la función fitness estarán en el mismo orden de magnitud.

6. Ejecución y Resultados

Para las comparación del rendimiento de los algoritmos desarrollados se plantea un caso de prueba base para los algoritmos ACO y PSO, el cual se describe a continuación.

6.1. Caso de Prueba

Desarrollar un proyecto de software web de gestión de para una institución. Para lograr finalizar la construcción del software se requiere completar siete (7) tareas, además se cuenta con un equipo de seis (6) personas con conocimientos variados. Los datos del proyecto se detallan a continuación en las siguientes tablas:

Tabla 2: *Lista de Habilidades (skills)*

	Id	Código	Nombre de Habilidad Requerida
	1	h1	Planeamiento y diseño
	2	h2	Manejo de Base de datos
	3	h3	Programación Front End
	4	h4	Programación Back end
	5	h5	Pruebas de calidad

La *tabla 2* contiene información de las habilidades (skills) requeridas para el desarrollo del proyecto.

Tabla 3: *Información de Empleados*

	Id	Código	Sueldo	Esfuerzo Máximo	h1	h2	h3	h4	h5
	1	P1	12.5	1.0	1	0	0	1	0
	2	P2	15.6	1.1	0	1	1	1	0
	3	P3	10.5	0.5	0	0	0	0	1
	4	P4	18.7	1.3	1	1	0	1	1
	5	P5	13.8	1.0	0	0	0	1	1
	6	P6	15.6	1.2	1	0	1	0	1

En la *tabla 3*, se muestra información de los empleados, entre los que destacan el sueldo, el esfuerzo máximo y las habilidades con las que cuenta cada uno de ellos. En las columnas de "h" Se ha marcado en 1 si el empleado domina determinada habilidad y 0 en caso contrario.

Tabla 4: *Información de Tareas del Proyecto de Software*

Id	Descripción	Requerimiento	Habilidades	Precendencia	Limite tiempo
1	Diseño de Software	10	1	-	10
2	Diseño BD	20	2,4	1	10
3	Programación Sistema	50	1,4	1,2	10
4	Diseño de Interfaz	15	1,3	1	10
5	Testeo Programa	50	5	3,4	10
6	Documentacion DB	15	1,4	2,5	10
7	Manual de usuario	10	1	4, 5	10

Finalmente, en la *tabla 4*, se muestra la información para todas las tareas del proyecto a desarrollar.

6.2. Resultados PSO

Para las pruebas de este algoritmo se asignan los siguientes valores como parámetros de Entrada.

Tabla 5: *Argumentos de entrada PSO*

Variable	Valor	Descripción
w	0.6	Peso de Inercia
c1,c2	2	Constantes de Aceleración
r1,r2	Aleatorio entre 0 y 1	Factores de escalamiento
Iteraciones	250	Número de veces que la partícula modificará su posición
Particulas	100	Número de partículas usadas

De acuerdo a las secciones 2.1 y 2.2 de este informe, el problema de la asignación de los recursos a las tareas de un proyecto de software es un caso de optimización multiobjetivo, por tanto tenemos que normalizar las funciones objetivos individuales de la función fitness con el objetivo de evitar posibles problemas de diferente unidad o magnitud significativamente diferente. Con ese proposito, a continuación se muestra un cuadro de 100 ejecuciones del algoritmo PSO con la función fitness sin normalizar y con los parámetros de la *tabla 5* y pesos $w_{cost} = 0,5$ y $w_{dur} = 0,5$. De los resultados generados se obtiene el valor máximo para costo y duración, cuyos valores servirán para normalizar la función fitness (función multiobjetivo) en futuras ejecuciones del programa.

Tabla 6: 100 corridas para determinar máximos de salario y Duración PSO

Ejecución	Tiempo (seg.)	Fitness	Costo	Duración	w_{cost}	$w_{duration}$
1	12.792	1339.052	2637.616	40.488	0.5	0.5
2	11.729	1333.169	2623.063	43.276	0.5	0.5
3	13.063	1345.667	2650.493	40.840	0.5	0.5
4	13.066	1323.800	2605.242	42.359	0.5	0.5
5	12.288	1348.241	2656.896	39.585	0.5	0.5
6	11.454	1325.557	2608.869	42.244	0.5	0.5
7	12.279	1340.349	2638.095	42.603	0.5	0.5
8	11.225	1354.992	2667.307	42.677	0.5	0.5
9	13.742	1339.973	2634.237	45.709	0.5	0.5
10	11.941	1340.438	2639.510	41.366	0.5	0.5
11	11.927	1333.356	2617.962	48.750	0.5	0.5
12	12.499	1346.744	2652.825	40.662	0.5	0.5
13	10.139	1329.131	2612.202	46.060	0.5	0.5
14	12.045	1349.728	2658.858	40.597	0.5	0.5
15	11.518	1335.309	2628.382	42.236	0.5	0.5
16	11.043	1339.302	2634.876	43.728	0.5	0.5
17	11.178	1350.791	2658.938	42.645	0.5	0.5
18	12.742	1335.849	2626.117	45.581	0.5	0.5
19	11.369	1344.108	2648.720	39.496	0.5	0.5
20	11.991	1354.670	2668.495	40.844	0.5	0.5
21	9.909	1350.171	2657.152	43.190	0.5	0.5
22	9.832	1296.205	2523.300	69.110	0.5	0.5
23	10.843	1322.920	2598.542	47.297	0.5	0.5
24	12.281	1336.688	2634.031	39.344	0.5	0.5
25	11.453	1355.657	2670.764	40.549	0.5	0.5
26	11.854	1319.121	2595.211	43.030	0.5	0.5
27	11.993	1370.432	2690.278	50.585	0.5	0.5
28	11.307	1328.997	2612.293	45.702	0.5	0.5
29	9.930	1344.026	2644.711	43.341	0.5	0.5
30	11.400	1331.261	2621.340	41.183	0.5	0.5
31	11.616	1357.950	2675.252	40.648	0.5	0.5
32	12.620	1338.466	2635.818	41.115	0.5	0.5
33	11.069	1318.550	2591.494	45.607	0.5	0.5
34	11.844	1356.557	2670.437	42.676	0.5	0.5
35	11.717	1337.365	2635.966	38.765	0.5	0.5
36	12.733	1348.016	2655.336	40.696	0.5	0.5
37	9.647	1339.120	2637.218	41.021	0.5	0.5
38	8.782	1346.197	2649.708	42.685	0.5	0.5
39	8.919	1339.443	2632.093	46.792	0.5	0.5
40	8.756	1330.469	2617.940	42.999	0.5	0.5
41	9.061	1342.633	2642.339	42.927	0.5	0.5
42	10.586	1327.717	2610.943	44.490	0.5	0.5
43	10.219	1316.642	2587.407	45.878	0.5	0.5
44	10.039	1331.580	2623.565	39.594	0.5	0.5
45	10.001	1332.421	2623.553	41.288	0.5	0.5
46	9.883	1338.419	2628.361	48.477	0.5	0.5
47	9.831	1349.833	2655.341	44.326	0.5	0.5
48	9.973	1342.676	2644.432	40.921	0.5	0.5
49	9.527	1314.790	2580.315	49.266	0.5	0.5
50	8.994	1342.354	2645.208	39.500	0.5	0.5
51	9.948	1320.031	2597.277	42.786	0.5	0.5
52	9.099	1322.519	2597.814	47.223	0.5	0.5
53	9.453	1322.530	2600.759	44.302	0.5	0.5
54	9.671	1321.059	2593.549	48.570	0.5	0.5
55	9.791	1341.373	2633.193	49.553	0.5	0.5
56	8.696	1337.647	2624.942	50.352	0.5	0.5
57	9.635	1332.883	2622.958	42.808	0.5	0.5
58	9.721	1355.606	2668.541	42.672	0.5	0.5
59	9.834	1353.451	2665.539	41.364	0.5	0.5
60	9.975	1326.546	2612.630	40.462	0.5	0.5
61	9.679	1321.968	2602.457	41.479	0.5	0.5
62	10.674	1336.973	2631.671	42.276	0.5	0.5
63	9.310	1346.812	2647.625	45.999	0.5	0.5
64	8.871	1326.066	2604.721	47.411	0.5	0.5
65	10.296	1345.124	2650.097	40.151	0.5	0.5
66	8.999	1332.273	2620.553	43.993	0.5	0.5
67	8.800	1318.904	2591.029	46.780	0.5	0.5
68	10.051	1345.848	2650.017	41.679	0.5	0.5
69	9.350	1337.687	2636.318	39.055	0.5	0.5
70	9.614	1321.970	2598.386	45.554	0.5	0.5
71	10.190	1341.792	2639.471	44.113	0.5	0.5
72	9.312	1349.644	2653.745	45.544	0.5	0.5
73	9.634	1351.597	2663.387	39.807	0.5	0.5
74	10.076	1344.815	2647.062	42.567	0.5	0.5
75	10.415	1329.612	2616.353	42.871	0.5	0.5
76	9.733	1325.791	2605.924	45.657	0.5	0.5
77	9.364	1330.178	2615.743	44.612	0.5	0.5
78	9.268	1337.349	2634.284	40.413	0.5	0.5
79	9.221	1324.258	2605.912	42.604	0.5	0.5
80	10.191	1329.904	2619.954	39.855	0.5	0.5
81	12.459	1329.719	2616.966	42.472	0.5	0.5
82	9.690	1346.495	2653.102	39.888	0.5	0.5
83	9.305	1322.704	2600.100	45.308	0.5	0.5
84	10.009	1313.281	2579.793	46.769	0.5	0.5
85	9.163	1353.631	2665.643	41.619	0.5	0.5
86	10.216	1331.013	2611.327	50.699	0.5	0.5
87	11.400	1342.881	2646.486	39.275	0.5	0.5
88	9.683	1323.886	2606.068	41.704	0.5	0.5
89	9.498	1337.167	2635.741	38.594	0.5	0.5
90	9.515	1340.024	2636.258	43.790	0.5	0.5
91	9.605	1323.980	2605.532	42.428	0.5	0.5
92	9.832	1338.844	2635.570	42.119	0.5	0.5
93	9.685	1344.543	2649.568	39.518	0.5	0.5
94	9.411	1335.692	2628.320	43.064	0.5	0.5
95	10.325	1336.182	2628.102	44.262	0.5	0.5
96	11.200	1347.779	2654.230	41.328	0.5	0.5
97	10.216	1338.649	2633.362	43.936	0.5	0.5
98	9.624	1326.327	2607.752	44.903	0.5	0.5
99	9.989	1337.799	2632.359	43.240	0.5	0.5
100	10.708	1352.164	2661.103	43.224	0.5	0.5
MAX	-	-	2690.278	69.110	-	-

De la tabla 6 se determinaron que los valores máximos son $\text{Salario}_{max} = 2690.278$ y $\text{Tiempo}_{max} = 69.110$. Una vez que se hallaron los valores máximos, estos se utilizan para normalizar la función multiobjetivo según la función fitness.

Finalmente se hacen una serie de ejecuciones para distintos valores de w_{cost} y w_{dur} tal que $w_{cost} + w_{dur} = 1$.

Tabla 7: Valores de w_{cost} y w_{dur} para las pruebas PSO multiobjetivo

w_{cost}	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
w_{dur}	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0.0

Para nuestra prueba se toman los valores mostrados en la tabla 7 Además para cada par de valores se ejecutan un total de 10 veces con 11 valores de pesos, haciendo un total de 110 ejecuciones bajo los parámetros descritos en la sección 6.2. Los resultados finales se observan en las siguientes tablas.

Tabla 8: Valores de la función Fitness para 110 ejecuciones PSO

w_{cost}	w_{dur}	1	2	3	4	5	6	7	8	9	10
0.0	1.0	0.586	0.586	0.586	0.586	0.586	0.586	0.586	0.584	0.700	0.584
0.1	0.9	0.631	0.646	0.625	0.655	0.656	0.630	0.659	0.697	0.610	0.603
0.2	0.8	0.662	0.672	0.658	0.677	0.715	0.662	0.716	0.669	0.668	0.650
0.3	0.7	0.720	0.720	0.720	0.720	0.720	0.720	0.720	0.736	0.761	0.710
0.4	0.6	0.741	0.756	0.738	0.743	0.764	0.741	0.766	0.751	0.777	0.744
0.5	0.5	0.802	0.787	0.794	0.810	0.789	0.802	0.790	0.790	0.818	0.790
0.6	0.4	0.835	0.835	0.835	0.835	0.835	0.835	0.835	0.823	0.825	0.833
0.7	0.3	0.879	0.879	0.879	0.879	0.879	0.879	0.879	0.862	0.861	0.898
0.8	0.2	0.908	0.912	0.911	0.909	0.909	0.908	0.908	0.912	0.913	0.905
0.9	0.1	0.937	0.958	0.941	0.940	0.945	0.937	0.945	0.951	0.944	0.930
1.0	0.0	0.972	0.976	0.971	0.972	0.975	0.972	0.975	0.938	0.963	0.955

Tabla 9: Valores de Costo Total del Proyecto para 110 ejecuciones PSO

w_{cost}	w_{dur}	1	2	3	4	5	6	7	8	9	10
0.0	1.0	2637.616	2637.616	2637.616	2637.616	2637.616	2637.616	2637.616	2652.235	2689.148	2655.072
0.1	0.9	2632.072	2634.855	2637.782	2620.581	2641.380	2632.452	2638.265	2612.202	2634.031	2635.966
0.2	0.8	2653.081	2626.452	2650.749	2663.205	2664.021	2652.944	2661.403	2665.493	2670.764	2653.952
0.3	0.7	2605.242	2605.242	2605.242	2605.242	2605.242	2605.242	2605.242	2632.779	2612.644	2637.218
0.4	0.6	2664.327	2641.410	2656.948	2658.943	2645.562	2664.219	2647.179	2647.949	2695.595	2649.995
0.5	0.5	2651.263	2633.850	2655.555	2635.277	2640.332	2651.483	2644.773	2651.928	2623.520	2612.629
0.6	0.4	2638.095	2638.095	2638.095	2638.095	2638.095	2638.095	2638.095	2656.722	2650.969	2617.940
0.7	0.3	2667.307	2667.307	2667.307	2667.307	2667.307	2667.307	2667.307	2651.679	2621.340	2669.342
0.8	0.2	2636.804	2675.975	2635.319	2640.989	2672.422	2636.756	2671.306	2668.495	2675.252	2610.943
0.9	0.1	2622.438	2626.856	2634.616	2619.764	2648.218	2622.735	2650.094	2657.152	2525.831	2569.334
1.0	0.0	2614.724	2624.698	2612.124	2616.004	2624.297	2614.748	2624.191	2523.300	2591.494	2568.903

Tabla 10: *Valores Duración total del proyecto para 110 ejecuciones PSO*

w_{cost}	w_{dur}	1	2	3	4	5	6	7	8	9	10
0.0	1.0	40.488	40.488	40.488	40.488	40.488	40.488	40.488	40.344	48.347	40.330
0.1	0.9	40.910	42.102	40.474	42.779	42.872	40.874	43.094	46.060	39.344	38.765
0.2	0.8	40.155	41.199	39.811	41.419	44.621	40.138	44.762	40.639	40.549	39.132
0.3	0.7	42.359	42.359	42.359	42.359	42.359	42.359	42.359	43.638	46.324	41.021
0.4	0.6	39.740	41.800	39.525	40.071	42.734	39.734	42.842	41.137	43.343	40.361
0.5	0.5	42.779	41.091	41.537	44.250	41.245	42.761	41.248	41.013	45.737	42.040
0.6	0.4	42.603	42.603	42.603	42.603	42.603	42.603	42.603	39.813	40.355	42.999
0.7	0.3	42.677	42.677	42.677	42.677	42.677	42.677	42.677	39.657	41.183	46.876
0.8	0.2	42.690	40.214	44.103	42.750	39.517	42.705	39.421	40.844	40.648	44.490
0.9	0.1	40.986	55.035	40.997	44.052	40.584	40.963	40.725	43.190	68.419	48.652
1.0	0.0	49.317	69.654	49.859	50.725	71.149	49.320	71.401	69.110	45.607	53.579

Tabla 11: *Tiempos de ejecución para 110 ejecuciones de PSO en segundos*

w_{cost}	w_{dur}	1	2	3	4	5	6	7	8	9	10
0.0	1.0	11.531	11.478	12.946	11.232	10.896	12.528	10.670	10.821	9.225	10.157
0.1	0.9	10.154	13.339	22.183	19.862	11.505	12.446	9.782	10.796	10.309	9.364
0.2	0.8	12.372	22.556	11.012	23.023	10.179	12.781	11.861	10.340	9.410	10.642
0.3	0.7	13.041	21.921	10.602	20.450	11.754	11.281	10.635	10.079	10.234	10.584
0.4	0.6	13.035	22.405	10.612	18.612	10.305	11.177	9.910	9.918	10.354	10.433
0.5	0.5	10.274	13.465	9.977	18.389	11.420	9.559	9.571	10.465	10.868	9.670
0.6	0.4	9.550	9.230	10.041	19.353	9.752	12.121	13.025	9.315	10.448	9.212
0.7	0.3	9.808	9.628	10.105	10.359	11.320	9.800	11.210	10.129	9.746	9.836
0.8	0.2	10.427	9.775	11.905	10.335	11.457	11.735	10.146	11.153	11.010	10.46
0.9	0.1	11.301	12.798	12.331	10.413	15.199	9.744	9.673	12.179	6.503	9.799
1.0	0.0	11.166	10.342	10.907	10.231	8.615	9.645	7.897	10.947	10.520	10.412

En la siguiente tabla se ven los valores mínimos, máximos y promedios de las 110 ejecuciones.

Tabla 12: *Estadísticas 110 corridas PSO*

w_{cost}	w_{dur}	Fitness			Duración Proyecto			Costo Proyecto			Tiempo Ejecución		
		Min	Max	Promedio	Min	Max	Promedio	Min	Max	Promedio	Min	Max	Promedio
0.0	1.0	0.584	0.700	0.597	40.330	48.347	41.244	2637.616	2689.148	2645.977	9.225	12.946	11.148
0.1	0.9	0.603	0.697	0.641	38.765	46.060	41.727	2612.202	2641.380	2631.959	9.364	22.183	12.974
0.2	0.8	0.650	0.716	0.675	39.132	44.762	41.243	2626.452	2670.764	2656.206	9.410	23.023	13.418
0.3	0.7	0.710	0.761	0.725	41.021	46.324	42.750	2605.242	2637.218	2611.934	10.079	21.921	13.058
0.4	0.6	0.738	0.777	0.752	39.525	43.343	41.129	2641.410	2695.595	2657.213	9.910	22.405	12.676
0.5	0.5	0.787	0.818	0.797	41.013	45.737	42.370	2612.629	2655.555	2640.061	9.559	18.389	11.366
0.6	0.4	0.823	0.835	0.833	39.813	42.999	42.139	2617.940	2656.722	2639.230	9.212	19.353	11.205
0.7	0.3	0.861	0.898	0.877	39.657	46.876	42.646	2621.340	2669.342	2661.351	9.628	11.320	10.194
0.8	0.2	0.905	0.913	0.910	39.421	44.490	41.738	2610.943	2675.975	2652.426	9.775	11.905	10.840
0.9	0.1	0.930	0.958	0.943	40.584	68.419	46.360	2525.831	2657.152	2617.704	6.503	15.199	10.994
1.0	0.0	0.938	0.976	0.967	45.607	71.401	57.972	2523.300	2624.698	2601.448	7.897	11.166	10.068

Tabla 13: *Matriz de asignaciones de recursos a las tareas con PSO*

	T1	T2	T3	T4	T5	T6	T7
P1	1.00	0.43	1.00	1.00	0.00	0.76	0.97
P2	0.00	0.27	1.10	1.10	0.00	1.05	0.00
P3	0.00	0.00	0.00	0.00	0.50	0.00	0.00
P4	1.27	1.30	0.43	1.21	1.30	1.30	1.30
P5	0.00	1.00	1.0	0.00	1.00	1.00	0.00
P6	1.2	0.00	1.20	1.20	0.93	0.87	1.20

En la *tabla* 13 se puede observar uno de los mejores resultados generados al ejecutar PSO en el cual se obtuvo un costo total de 2634.031 dolares y un tiempo total de 39.344 días para el proyecto de software web con $w_{cost} = 0,1$ y $w_{dur} = 0,9$.

6.3. Resultados ACO

Para la pruebas del algoritmo ACO se definen los siguientes valores iniciales para los parámetros principales:

Tabla 14: *Tabla de parámetro iniciales del algoritmo ACO.*

Parámetro	Valor	Descripción
N_{ant}	20	Número de hormigas
α	7	Coficiente para el control de la influencia/- peso de la feromonas
β	1	Coficiente para el control de la influencia/- peso de la heurística
ρ	0.5	Tasa de volatilidad de las feromonas
τ	0.4	Valor inicial de las feromonas
q_0	0.5	Valor que permite balancear (regla de proporcionalidad aleatoria) entre la exploración y la explotación
N_{gen}	250	Máximo número de generaciones ACO
w_{cost}	0.5	Peso inicial del costo, utilizado para la normalización de costo del proyecto en la función fitness
w_{dur}	0.5	Peso inicial del la duración, utilizado para la normalización de la duración del proyecto en la función fitness

De acuerdo a la sección 2.1 de este informe, el problema de la asignación de los recursos a las tareas de un proyecto de software es un caso de optimización multiobjetivo. En este informe se compara el rendimiento del algoritmo ACO y PSO, para lograr una comparación en las misma condiciones: función fitness y pesos similares, tenemos que normalizar las objetivos individuales de la función fitness. A continuación se muestra un cuadro de 100 ejecuciones del algoritmo ACO, con los parámetros detallados en la *tabla 14*, luego se obtiene el valor máximo para costo y duración de la lista, esta información servirá para normalizar la función fitness (función multiobjetivo). De la *tabla 15*, se determina que $Salario_{max} = 2227.498$ y $Tiempo_{max} = 217.613$.

Tabla 15: *Ejecuciones para hallar el máximo valor Costo y Duración ACO*

Ejecución	Tiempo (seg.)	Fitness	Costo	Duración	w_{cost}	$w_{duration}$
1	20.344	1165.727	2197.220	152.983	0.5	0.5
2	11.875	1161.279	2152.280	180.817	0.5	0.5
3	11.578	1167.500	2142.618	192.381	0.5	0.5
4	11.719	1165.205	2190.540	156.660	0.5	0.5
5	11.594	1167.176	2163.508	184.508	0.5	0.5
6	12.125	1175.406	2187.575	169.487	0.5	0.5
7	11.594	1166.998	2161.275	183.261	0.5	0.5
8	11.531	1165.021	2164.272	172.021	0.5	0.5
9	11.703	1169.400	2184.459	164.881	0.5	0.5
10	11.672	1165.373	2219.699	136.048	0.5	0.5
11	11.625	1166.528	2154.257	188.175	0.5	0.5
12	11.563	1163.339	2164.020	162.657	0.5	0.5
13	11.594	1174.597	2170.515	184.930	0.5	0.5
14	11.563	1156.100	2145.740	191.459	0.5	0.5
15	11.656	1164.628	2213.750	140.506	0.5	0.5
16	11.719	1170.734	2175.847	182.411	0.5	0.5
17	11.750	1166.102	2142.902	192.427	0.5	0.5
18	11.563	1166.723	2158.946	187.000	0.5	0.5
19	11.578	1175.974	2184.868	174.495	0.5	0.5
20	11.703	1161.758	2182.245	153.770	0.5	0.5
21	11.703	1171.062	2198.893	152.606	0.5	0.5
22	11.750	1171.639	2164.385	185.144	0.5	0.5
23	13.766	1164.120	2188.333	168.032	0.5	0.5
24	16.531	1169.159	2157.210	190.483	0.5	0.5
25	13.625	1169.612	2180.585	161.764	0.5	0.5
26	11.688	1165.583	2162.754	188.327	0.5	0.5
27	11.563	1172.528	2179.554	174.877	0.5	0.5
28	11.672	1158.721	2145.661	188.571	0.5	0.5
29	12.125	1169.166	2164.386	183.321	0.5	0.5
30	11.500	1162.385	2167.874	185.021	0.5	0.5
31	11.594	1167.837	2197.624	150.550	0.5	0.5
32	11.766	1155.287	2144.323	188.126	0.5	0.5
33	11.594	1169.657	2162.104	184.624	0.5	0.5
34	11.703	1174.320	2224.210	148.635	0.5	0.5
35	11.641	1171.418	2227.498	143.463	0.5	0.5
36	11.875	1174.313	2185.826	173.340	0.5	0.5
37	11.781	1168.097	2184.584	170.360	0.5	0.5
38	11.766	1173.812	2175.286	184.838	0.5	0.5
39	11.953	1166.318	2206.429	157.457	0.5	0.5
40	11.766	1170.082	2152.905	191.548	0.5	0.5
41	11.672	1171.432	2181.216	183.523	0.5	0.5
42	11.766	1175.182	2191.916	167.822	0.5	0.5
43	11.719	1164.109	2188.650	155.192	0.5	0.5
44	11.750	1173.920	2158.469	192.496	0.5	0.5
45	12.000	1180.394	2217.921	153.407	0.5	0.5
46	12.469	1167.729	2184.405	183.469	0.5	0.5
47	12.438	1172.166	2172.831	187.126	0.5	0.5
48	12.422	1171.166	2159.952	182.381	0.5	0.5
49	12.328	1170.537	2202.020	153.883	0.5	0.5
50	11.969	1159.317	2175.082	160.341	0.5	0.5
51	11.938	1173.271	2204.403	158.929	0.5	0.5
52	11.969	1165.738	2192.199	148.652	0.5	0.5
53	12.016	1173.809	2214.848	148.395	0.5	0.5
54	11.984	1169.058	2207.669	146.073	0.5	0.5
55	12.031	1163.885	2157.484	187.076	0.5	0.5
56	12.125	1166.942	2167.496	178.889	0.5	0.5
57	11.906	1157.594	2185.666	142.022	0.5	0.5
58	13.250	1162.242	2177.366	160.842	0.5	0.5
59	12.031	1162.197	2160.628	182.515	0.5	0.5
60	11.953	1164.703	2175.317	169.714	0.5	0.5
61	12.125	1166.403	2198.051	156.630	0.5	0.5
62	12.422	1165.927	2143.043	195.060	0.5	0.5
63	11.922	1178.638	2208.436	158.215	0.5	0.5
64	12.219	1174.579	2200.187	172.381	0.5	0.5
65	11.969	1159.192	2144.095	186.789	0.5	0.5
66	12.125	1171.669	2197.878	159.124	0.5	0.5
67	12.094	1170.132	2153.819	189.571	0.5	0.5
68	11.953	1168.269	2191.155	159.048	0.5	0.5
69	11.828	1164.305	2153.373	187.736	0.5	0.5
70	11.813	1170.811	2165.907	175.714	0.5	0.5
71	11.625	1172.507	2133.651	217.613	0.5	0.5
72	11.719	1161.022	2190.390	156.654	0.5	0.5
73	11.906	1171.757	2219.620	156.309	0.5	0.5
74	11.906	1168.943	2174.778	184.188	0.5	0.5
75	12.188	1169.982	2165.582	190.007	0.5	0.5
76	11.922	1152.815	2140.393	174.612	0.5	0.5
77	11.984	1167.500	2148.868	192.381	0.5	0.5
78	11.938	1169.632	2167.238	176.316	0.5	0.5
79	12.078	1171.605	2191.556	154.779	0.5	0.5
80	12.250	1169.097	2200.408	153.411	0.5	0.5
81	11.828	1165.738	2182.857	168.535	0.5	0.5
82	11.828	1157.451	2158.574	171.953	0.5	0.5
83	11.781	1169.503	2188.259	160.122	0.5	0.5
84	12.141	1169.153	2160.603	191.368	0.5	0.5
85	11.734	1159.702	2209.884	128.271	0.5	0.5
86	11.969	1161.114	2198.408	142.571	0.5	0.5
87	11.891	1168.772	2156.831	194.378	0.5	0.5
88	11.781	1167.413	2154.299	189.901	0.5	0.5
89	11.953	1155.577	2148.802	171.727	0.5	0.5
90	11.813	1167.367	2145.959	191.900	0.5	0.5
91	11.813	1169.713	2159.016	189.785	0.5	0.5
92	11.844	1163.758	2144.885	189.280	0.5	0.5
93	11.813	1167.998	2186.052	169.858	0.5	0.5
94	11.813	1170.980	2184.862	169.598	0.5	0.5
95	11.859	1160.371	2178.751	161.905	0.5	0.5
96	12.344	1170.648	2166.330	182.381	0.5	0.5
97	12.156	1176.389	2225.510	150.309	0.5	0.5
98	11.984	1159.596	2168.576	163.117	0.5	0.5
99	12.266	1169.001	2188.089	166.703	0.5	0.5
100	12.500	1171.125	2209.350	151.650	0.5	0.5
MAX	-	-	2227.498	217.613	-	-

Finalmente, se hacen una varias de ejecuciones para distintos valores de w_{cost} y w_{dur} tal que $w_{cost} + w_{dur} = 1$.

Tabla 16: Valores de w_{cost} y w_{dur} para las pruebas ACO multiobjetivo

w_{cost}	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
w_{dur}	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0.0

Para las pruebas se toman los valores mostrados en la *tabla 16*, donde para cada para de valores se ejecuta 10 veces con 11 valores de pesos, haciendo un total de 110 ejecuciones bajo los parámetros descritos en la sección 6.3. Los resultados finales se observan en las siguientes tablas.

Tabla 17: Valores de la función Fitness para 110 ejecuciones ACO

w_{cost}	w_{dur}	1	2	3	4	5	6	7	8	9	10
0.0	1.0	0.152	0.151	0.152	0.152	0.151	0.152	0.153	0.151	0.154	0.149
0.1	0.9	0.255	0.257	0.254	0.259	0.254	0.255	0.258	0.257	0.252	0.255
0.2	0.8	0.365	0.363	0.364	0.367	0.357	0.362	0.360	0.361	0.362	0.365
0.3	0.7	0.464	0.469	0.467	0.468	0.465	0.462	0.466	0.471	0.467	0.468
0.4	0.6	0.565	0.570	0.570	0.566	0.568	0.567	0.567	0.569	0.568	0.571
0.5	0.5	0.667	0.666	0.670	0.668	0.670	0.668	0.664	0.667	0.668	0.665
0.6	0.4	0.757	0.763	0.756	0.759	0.754	0.757	0.763	0.766	0.765	0.765
0.7	0.3	0.849	0.848	0.849	0.852	0.846	0.851	0.849	0.844	0.849	0.847
0.8	0.2	0.908	0.924	0.917	0.918	0.922	0.907	0.912	0.920	0.914	0.908
0.9	0.1	0.955	0.950	0.949	0.958	0.962	0.951	0.949	0.952	0.962	0.956
1.0	0.0	0.952	0.951	0.951	0.953	0.951	0.952	0.952	0.948	0.959	0.952

Tabla 18: Valores de Costo Total del Proyecto para 110 ejecuciones ACO

w_{cost}	w_{dur}	1	2	3	4	5	6	7	8	9	10
0.0	1.0	2718.449	2751.584	2678.213	2726.855	2750.604	2728.006	2720.960	2707.218	2759.184	2737.111
0.1	0.9	2690.888	2710.247	2661.133	2666.791	2697.695	2693.964	2748.421	2725.928	2720.637	2736.485
0.2	0.8	2735.448	2732.638	2703.304	2720.876	2637.410	2694.641	2645.162	2718.767	2644.616	2684.449
0.3	0.7	2647.068	2685.383	2652.696	2676.031	2687.588	2635.898	2650.925	2664.948	2634.499	2684.720
0.4	0.6	2587.287	2583.165	2631.418	2614.736	2656.524	2547.246	2644.859	2628.229	2638.525	2632.335
0.5	0.5	2555.468	2536.401	2612.860	2591.185	2642.274	2601.224	2576.947	2552.539	2592.542	2583.534
0.6	0.4	2458.739	2500.941	2472.089	2494.254	2449.473	2516.383	2506.230	2565.128	2547.962	2467.398
0.7	0.3	2428.920	2420.774	2400.637	2439.107	2415.264	2437.616	2349.679	2430.145	2449.279	2456.038
0.8	0.2	2289.667	2338.734	2361.977	2337.472	2332.058	2343.664	2268.095	2387.485	2330.879	2261.545
0.9	0.1	2230.335	2186.230	2178.484	2193.313	2224.886	2161.804	2172.702	2244.611	2226.384	2170.108
1.0	0.0	2128.190	2120.791	2126.586	2126.853	2124.841	2124.304	2119.591	2115.732	2135.152	2127.851

Tabla 19: *Valores Duración Total del Proyecto para 110 ejecuciones ACO*

w_{cost}	w_{dur}	1	2	3	4	5	6	7	8	9	10
0.0	1.0	33.073	32.926	33.106	33.154	32.926	32.978	33.286	32.893	33.451	32.512
0.1	0.9	33.146	33.498	33.173	34.220	32.877	33.274	33.661	33.498	32.468	32.877
0.2	0.8	34.454	34.220	34.495	34.816	33.802	33.895	34.753	33.876	35.022	34.839
0.3	0.7	35.780	36.651	36.015	35.274	35.380	34.585	35.578	36.445	35.861	36.016
0.4	0.6	38.695	39.572	37.800	37.278	36.523	40.537	35.779	38.179	36.380	37.579
0.5	0.5	44.581	43.699	40.790	39.763	39.169	40.083	40.960	42.052	41.410	38.192
0.6	0.4	54.704	50.562	52.919	49.995	53.976	46.614	52.240	45.787	49.431	56.929
0.7	0.3	68.304	64.390	74.706	69.326	69.386	66.144	87.304	63.641	60.747	57.747
0.8	0.2	99.438	99.008	87.705	87.781	98.770	79.805	111.747	80.707	92.509	111.929
0.9	0.1	126.455	155.714	158.102	171.905	144.515	174.881	163.523	111.953	142.537	187.216
1.0	0.0	330.707	319.353	324.969	327.001	229.785	332.381	351.100	332.955	323.340	331.152

Tabla 20: *Tiempos de ejecución para 110 ejecuciones de ACO en segundos*

w_{cost}	w_{dur}	1	2	3	4	5	6	7	8	9	10
0.0	1.0	22.203	36.609	38.078	38.859	37.703	41.297	46.672	43.156	44.266	47.125
0.1	0.9	51.141	39.219	43.078	40.625	75.406	68.703	41.422	38.313	38.766	37.719
0.2	0.8	38.859	40.531	39.328	38.891	37.984	39.359	39.531	38.625	38.313	37.766
0.3	0.7	39.000	43.406	38.500	38.938	38.750	38.453	39.125	37.906	38.469	39.438
0.4	0.6	39.875	39.203	38.656	37.516	39.375	39.672	41.766	43.938	38.781	39.469
0.5	0.5	37.625	37.391	39.672	39.063	38.422	38.344	39.672	38.000	37.578	39.078
0.6	0.4	38.344	37.750	37.750	38.531	38.313	38.906	40.141	38.359	38.094	39.234
0.7	0.3	37.219	39.219	40.172	35.969	38.578	37.250	37.984	37.156	37.344	37.688
0.8	0.2	38.094	37.141	37.813	39.391	37.313	37.844	37.313	39.125	37.641	37.250
0.9	0.1	36.484	36.172	36.906	37.031	37.063	38.656	37.344	37.531	35.766	33.906
1.0	0.0	32.906	32.438	33.766	34.234	32.734	32.391	32.391	31.875	33.344	32.750

En la siguiente tabla se ven los valores mínimos, máximos y promedios de las 110 ejecuciones.

Tabla 21: *Estadísticas 110 ejecuciones ACO*

w_1	w_2	Fitness			Duración Proyecto			Costo Proyecto			Tiempo Ejecución		
		Min	Max	Promedio	Min	Max	Promedio	Min	Max	Promedio	Min	Max	Promedio
0.0	1.0	0.149	0.154	0.152	32.512	33.451	33.030	2678.213	2759.184	2727.818	22.203	47.125	39.597
0.1	0.9	0.252	0.259	0.256	32.468	34.220	33.269	2661.133	2748.421	2705.219	37.719	75.406	47.439
0.2	0.8	0.357	0.367	0.363	33.802	35.022	34.417	2637.410	2735.448	2691.731	37.766	40.531	38.919
0.3	0.7	0.462	0.471	0.467	34.585	36.651	35.759	2634.499	2687.588	2661.975	37.906	43.406	39.198
0.4	0.6	0.565	0.571	0.568	35.779	40.537	37.832	2547.246	2656.524	2616.432	37.516	43.938	39.825
0.5	0.5	0.664	0.670	0.667	38.192	44.581	41.070	2536.401	2642.274	2584.497	37.391	39.672	38.484
0.6	0.4	0.754	0.766	0.760	45.787	56.929	51.316	2449.473	2565.128	2497.860	37.750	40.141	38.542
0.7	0.3	0.844	0.852	0.848	57.747	87.304	68.170	2349.679	2456.038	2422.746	35.969	40.172	37.858
0.8	0.2	0.907	0.924	0.915	79.805	111.929	94.940	2261.545	2387.485	2325.158	37.141	39.391	37.892
0.9	0.1	0.949	0.962	0.954	111.953	187.216	153.680	2161.804	2244.611	2198.886	33.906	38.656	36.686
1.0	0.0	0.948	0.959	0.952	229.785	351.100	320.274	2115.732	2135.152	2124.989	31.875	34.234	32.883

Tabla 22: *Matriz de asignaciones de recursos a las tareas con ACO*

	T1	T2	T3	T4	T5	T6	T7
P1	1.00	1.00	1.00	0.75	0.00	0.50	1.00
P2	0.00	1.10	0.83	0.55	0.00	1.10	0.00
P3	0.00	0.00	0.00	0.00	0.50	0.00	0.00
P4	1.30	0.98	1.30	1.30	1.30	1.30	1.30
P5	0.00	1.00	0.75	0.00	1.00	1.00	0.00
P6	1.20	0.00	1.20	1.20	1.20	1.20	1.20

En la *tabla 22* se puede observar uno de los mejores resultados generados al ejecutar ACO en el cual se obtuvo un costo total de 2720.637 dolares y un tiempo total de 32.468 días para el proyecto de software web.

6.4. Menores Fitness para ACO y PSO

En las siguiente *tabla 23* se observan los resultados con menor Fitness para PSO y ACO. Para seleccionar dicho valor, no se consideran los casos donde alguno de los pesos es igual a 0 dado que se reduciría a un caso mono-objetivo.

Tabla 23: *Tabla de mejores resultados PSO vs ACO*

Algoritmo	w_{cost}	w_{dur}	Fitness	Tiempo Proyecto	Monto a gastar	Tiempo Ejecución
PSO	0.1	0.9	0.603	38.765 Dias	2635.966 dólares	9.364 segundos
ACO	0.1	0.9	0.252	32.468 Dias	2720.637 dólares	38.766 segundos

7. Conclusiones

Una vez terminada la ejecución de los algoritmos propuestos en el presente trabajo y usando el caso presentado se pueden plantear las siguientes conclusiones:

- Tal como se ve en la definición al ser un problema multiobjetivo (minimizar el Costo total y Duración total) en las tablas de resultados se observa que se afectan mutuamente por la variación de pesos, con más notoriedad en ACO. Por ello aplicar el método de **suma ponderada** podemos asignarle pesos y obtener una serie de soluciones donde la decisión final dependerá de la selección valor optimizado del fitness a criterio del responsable del proyecto.
- Las funciones objetivo del costo y la duración del proyecto, f_{cost} y f_{dur} para PSO y ACO se han igualado en orden magnitud usando un procedimiento matemático de normalización, esto mejora la evaluación del fitness para la comparación.
- En el caso del PSO los valores de los pesos w_{cost} y w_{dur} no necesariamente decantan una mayor influencia o un mejor valor tanto para el costo como en el tiempo total al momento de calcular el fitness.

- Los algoritmos PSO y ACO tienden a asignar el máximo esfuerzos disponible para cada empleado a las tareas de proyecto, cuando el peso de la función duración es mayor que el peso de la función costo.
- La exploración y explotación en el caso de ACO mejoraron considerablemente la generación de mejores resultados.
- Tanto para ACO y PSO es necesario hacer una búsqueda de valores óptimos (*afinar*) para sus parámetros de entrada, dado que de ello dependerá la eficacia en el cálculo de Costo y Duración óptimo para nuestro caso.
- La complejidad de la implementación del algoritmo ACO es mayor a la del algoritmo PSO, debido a sus componentes estocásticos. También se puede notar que es más sensible a las variaciones de pesos.

Referencias

- [1] Kuang-Hua Chang. Multiobjective optimization and advanced topics. *Design Theory and Methods Using CAD/CAE*, pages 325–406, 2015. URL <https://doi.org/10.1016/B978-0-12-398512-5.00005-0>.
- [2] J. Francisco Chicano Enrique Alba *. Software project management with gas. *Science Direct*, pages 2380–2401, Diciembre 2006.
- [3] Yong Tang Jing Xiao, Xian Ting Ao. Solving software project scheduling problems with ant colony optimization. *Science Direct*, pages 33–46, Mayo 2021.
- [4] Usman Qamar Syed Khizer Abass Mazhar Hameed, Hiba Khalid. Optimizing software project management staffing and work-force deployment processes using swarm intelligence. *Computing Conference 2017*, Julio 2017. URL DOI:10.1109/SAI.2017.8252084.
- [5] Jasbir S. Arora R. Timothy Marler. Function-transformation methods for multi-objective optimization. *Engineering Optimization*, 6:551–570, 08 2006. URL <http://dx.doi.org/10.1080/03052150500114289>.
- [6] Constantinos Stylianou² Simos Gerasimou¹ and Andreas S. Andreou¹. An investigation of optimal project scheduling and team staffing in software development using particle swarm optimization. *14th ICEIS*, 1:168–171, Enero 2012. URL <https://doi.org/10.5220/0004001001680171>.
- [7] Bin Xu¹ Yujia Ge¹. Dynamic staffing and rescheduling in software project management: A hybrid approach. *PLoS ONE*, 62(1), Junio 2016. URL <http://DOI:10.1371/journal.pone.0157104>.

Appendices

A. Códigos en Python

A.1. Código PSO

```
1 from datetime import *
2 import random,os,copy,time
3 import itertools,math
4 import numpy as np
5 import pandas as pd
6 from itertools import chain, combinations, product
7 import matplotlib.pyplot as plt
8
9 #leyendo datos en dataframes
10 #habilidades
11 pdskills = pd.read_csv('skills.csv',encoding='ISO-8859-1').set_index('id')
12 #programadores
13 pdprogramadores = pd.read_csv('programadores.csv',encoding='ISO-8859-1').set_index('id')
14 #tareas
15 pdtareas = pd.read_csv('tareas.csv',encoding='ISO-8859-1').set_index('id')
16 pdtareas['Skills']=[[1],[2,4],[1,4],[1,3],[5],[1,4],[1]]
17 pdtareas['Precedencia']=[[0],[1],[1,2],[1],[3,4],[2,5],[4,5]]
18
19 codigoskills=['PYD','MBD','PF','PB','TEST']
20 print("Informacion del proyecto")
21 print("Lista de requerimientos tecnicos del proyecto:")
22 print(pdskills)
23 print("Lista de programadores y sus conocimientos:")
24 print(pdprogramadores)
25 print("Lista de tareas:")
26 print(pdtareas)
27
28 #dimensiones de la matriz de dedicacion
29 k=[6,7]
30
31 #variables de entrada para le algoritmo
32 w = 0.6
33 c1 = c2 = 2
34 r1 = np.random.uniform(0,1)
35 r2 = np.random.uniform(0,1)
36 random.seed(442)
37
38 #clase que almacena la informacion de la particula
39 class particle:
40     def __init__(self):
41         self.k= k
42         self.velocity = []
43         self.position = []
44         self.pbesti = []
45         #actualizar la velocidad
46         def update_velocity(self,newvel):
47             self.velocity= newvel
48         # actualizar la posicion
49         def update_position(self,newposit):
50             self.position= newposit
51         # actualizar la mejor posicion local de la particula
52         def update_pbesti(self,newpbesti):
53             self.pbesti= newpbesti
54         # generar una lista con las velocidades
```

```

55     def generate_listVelocity(self):
56         return self.velocity
57     ## generar una lista con posiciones
58     def generate_listPosition(self):
59         return self.position
60     def generate_listbestPosition(self):
61         return self.pbesti
62
63 #ajustar la posici n de la particula
64 #la posicion representar el esfuerzo de cada programador
65 def ajustarposicion(x,i):
66     if x < 0:
67         return 0
68     elif x > pdprogramadores['Capacidad'][i+1]:
69         return pdprogramadores['Capacidad'][i+1]
70     else:
71         return x
72
73 #funcion auxiliar
74 def mostrarTabla(solucion):
75     for i in range(0,k[0]):
76         for j in range(0,k[1]):
77             print(f"{solucion[i][j]:.4f}",end=" ")
78         print("")
79
80 #calcular el tiempo total de tareas
81 def tiempoTareas(solucion):
82     esfuerzo_equipo = [0]*k[1]
83     tiempos = [0]*k[1]
84     es_cubierto = True
85     #sumar los esfuerzos de cada miembro
86     for j in range(0,k[1]):
87         suma = 0
88         for i in range(0,k[0]):
89             suma = suma + solucion[i][j]
90         esfuerzo_equipo[j] = suma
91     #tiempo = esfuerzo de la tarea / esfuerzo del grupo
92     #se le da un valor muy grande al tiempo de tarea si el esfuerzo es 0 lo que
93     #significa que la tarea no fue asignada a nadie, controlando una de las
94     #restricciones
95     for j in range(0,k[1]):
96         if esfuerzo_equipo[j]!=0:
97             tiempos[j]=pdtareas['Esfuerzo'][j+1]/esfuerzo_equipo[j]
98         else:
99             tiempos[j] = 1000000
100         es_cubierto = False
101     return tiempos,sum(tiempos), es_cubierto
102
103
104 #calcula el salario total y a su vez devuelve junto a informacion del tiempo.
105 def salarioytiempoTareas(solucion):
106     #calcular el tiempo de las tareas producidas por la solucion
107     tiempos,tiempoTotal,todocubierto = tiempoTareas(solucion)
108     salario_tarea = [0]*k[1]
109     #se calcula el salario de cada tarea y luego se suma
110     for j in range(0,k[1]):
111         salario = 0
112         for i in range(0,k[0]):
113             salario = salario + pdprogramadores['Sueldo'][i+1]*solucion[i][j]*tiempos[
114                 j]
115         salario_tarea[j] = salario
116     salarioTotal = sum(salario_tarea)
117     return salarioTotal, tiempos,tiempoTotal,todocubierto
118

```

```

119 #funcion fitness
120 def fitnessMulti(solucion,w_cost,w_dur):
121     costo_max = 2690.278#valor obtenido de correr 100 veces antes
122     tiempo_max = 69.110 #valor obtenido de correr 100 veces antes
123     costo, tiempos,duracion,todocubierto = salarioytiempoTareas(solucion)
124     #normalizada
125     f = w_cost*(costo/costo_max) + w_dur*(duracion/tiempo_max)
126     return f
127
128 #funcion auxiliar de skills
129 def skillsEquipo():
130     skills_programador=[]
131     for i in range(0,k[0]):
132         sk=[]
133         for m in codigoskills:
134             if pdprogramadores[m][i+1] == 1:
135                 sk.append(pdskills[pdskills['codigo']==m].index.values.astype(int)[0])
136         skills_programador.append(sk)
137     return skills_programador
138
139
140 def skillsTareas():
141     conjunto_requisitos=[]
142     for j in range(0,k[1]):
143         requisitos = pdtareas['Skills'][j+1]
144         conjunto_requisitos.append(set(requisitos))
145     return conjunto_requisitos
146
147 #verificar si un programador tiene un requisito
148 def cumpleRequisito(i,skillid):
149     codigo = pdskills['codigo'][skillid]
150     if pdprogramadores[codigo][i]==1:
151         return True
152     else:
153         return False
154 # genera una matriz de habilita solo a los que cumplen con alg n requisito
155 def cumpleRequisitosTarea():
156     matriz_participacion=[]
157     for i in range(0,k[0]):
158         matriz_participacion.append([False]*k[1])
159     for j in range(0,k[1]):
160         requisitos = pdtareas['Skills'][j+1]
161         for i in range(0,k[0]):
162             for skillid in requisitos:
163                 if cumpleRequisito(i+1,skillid):
164                     matriz_participacion[i][j]=True
165     return matriz_participacion
166
167 #funcion que verifica si todos los requisitos fueron cubiertos
168 def requisitosReunidos(solucion):
169     skillsporequipo= skillsEquipo()
170     lista_requisitos = [[]]*k[1]
171     conjunto_requisitos = [[]]*k[1]
172     cumple_requisitos = [False]*k[1]
173     for j in range(0,k[1]):
174         lista = []
175         for i in range(0,k[0]):
176             if solucion[i][j]>0:
177                 lista.append(skillsporequipo[i])
178
179         lista_requisitos[j]= lista
180 #creando los conjuntos de requisitos reunidos en cada tarea
181 for j in range(0,k[1]):
182     aux1 = []
183     elemento = lista_requisitos[j]

```

```

184         for sk in elemento:
185             aux1.extend(sk)
186             elemento_c = set(aux1)
187             conjunto_requisitos[j]=elemento_c
188 #comparando las dos listas de conjuntos
189 skillsportarea = skillsTareas()
190 for j in range(0,k[1]):
191     if skillsportarea[j].issubset(conjunto_requisitos[j]):
192         cumple_requisitos[j]=True
193 if False in cumple_requisitos:
194     return False
195 else:
196     return True
197
198 #funcion pso
199 def test_pso(m,matriz_participacion,w_cost,w_dur):
200     NCmax = 250
201     bestTest = None
202     NC = 0
203     #print(sss)
204     particles_list= []
205     # inicializando las posiciones y velocidades de la particula
206     for i in range(0,m):
207         #generando y definiendo aleatoriamente la posicion de las m particulas
208         vel_ini=[]
209         post_ini=[]
210         p1 = None
211         for d1 in range(0,k[0]):
212             vel_ini_row = []
213             post_ini_row = []
214             for d2 in range(0,k[1]):
215                 if matriz_participacion[d1][d2]:
216                     vel_ini_row.append(random.uniform(0,pdprogramadores['Capacidad
217                                     '][d1+1]))
218                     post_ini_row.append(random.uniform(0,pdprogramadores['
219                                     Capacidad'][d1+1]))
220                 else:
221                     vel_ini_row.append(0.0)
222                     post_ini_row.append(0.0)
223             vel_ini.append(vel_ini_row)
224             post_ini.append(post_ini_row)
225             p1 = particle() # cuidado
226             p1.update_velocity(vel_ini)
227             p1.update_position(post_ini)
228             p1.update_pbesti(post_ini)
229             particles_list.append(p1)
230 gBest=particles_list[0].generate_listPosition()
231 #movimiento de las particulas
232 while NC < NCmax:
233     #comparar las funciones fitness con la finalidad de minimizar
234     pp = particles_list[i].generate_listPosition()
235     pb = particles_list[i].generate_listBestPosition()
236     if fitnessMulti(pp,w_cost,w_dur) < fitnessMulti(pb,w_cost,w_dur):
237         particles_list[i].update_pbesti(pp)
238     if fitnessMulti(pp,w_cost,w_dur) < fitnessMulti(gBest,w_cost,w_dur):
239         gBest= pp
240     if fitnessMulti(pp,w_cost,w_dur) == 0:
241         bestTest = pp
242 #actualizar posicion y velocidad
243 for i in range(0,m):
244     for d1 in range(0,k[0]):
245         for d2 in range(0,k[1]):
246             if matriz_participacion[d1][d2]:
247                 particles_list[i].velocity[d1][d2] = (w*particles_list[i].
248                     velocity[d1][d2] +

```

```

246         c1*r1*(particles_list[i].
                pbesti[d1][d2] -
                particles_list[i].
                position[d1][d2]))+
247         c2*r2*(gBest[d1][d2]-
                particles_list[i].
                position[d1][d2]))
248         #particles_list[i].velocity[d1][d2] = ajustvel(particles_list[i].
                velocity[d1][d2],d1)
249     for i in range(0,m):
250         for d1 in range(0,k[0]):
251             for d2 in range(0,k[1]):
252                 if matriz_participacion[d1][d2]:
253                     particles_list[i].position[d1][d2] = particles_list[i].
                        position[d1][d2] + particles_list[i].velocity[d1][d2]
254                     particles_list[i].position[d1][d2] = ajustarposicion(
                        particles_list[i].position[d1][d2],d1)
255     NC = NC + 1
256     bestTest = gBest
257     return bestTest,fitnessMulti(bestTest,w_cost,w_dur)
258
259 def PSO_iterator(npart):
260     matriz_participacion = cumpleRequisitosTarea()
261     listaw = [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
262     soluciones = [] # lista de soluciones
263     solucionesMonto = [] #lista de montos
264     solucionesTiempo= [] # lista de tiempo totales de proyecto
265     solucionesFitness=[] # lista de funcion fitness
266     solucionesTiempos=[] #lista de lista de tiempos por cada tarea
267     tiempos_lista=[] # lista de tiempos de ejecucion
268     #iterar para cada par de pesos w_cost y w_dur / w_cost + w_dur = 1
269     for w_cost in listaw:
270         time_star = time.time()
271         solucion, fitness = test_pso(npart,matriz_participacion,w_cost,1-w_cost)
272         costo,tiempos,duracion,todocubierto = salarioytiempoTareas(solucion)
273         time_final = time.time()-time_star
274         valida_requisitos = requisitosReunidos(solucion)
275         #print(f"{w_cost:.1f} {1-w_cost:.1f} {fitness:.3f} {tiempoTotal:.3f} {monto:.3f} {time_final:.3f}")
276         #mostrando soluciones para cada par de pesos
277         print(f"w_cost = {w_cost}, w_dur = {1-w_cost}")
278         mostrarTabla(solucion)
279         print("Mejor fitness"+ str(fitness))
280         print("Costo total "+ str(costo)+ " Dolares")
281         print("Duracion total "+ str(duracion) + " Dias")
282         print(valida_requisitos)
283         print(f"Tiempo de corrida {time_final:.10f} segundos")
284         if todocubierto and valida_requisitos:
285             soluciones.append(solucion)
286             solucionesFitness.append(fitness)
287             solucionesMonto.append(costo)
288             solucionesTiempo.append(duracion)#duracion total
289             solucionesTiempos.append(tiempos)#array de tiempos por cada tarea
290             tiempos_lista.append(time_final)
291     #obtener el que tiene el menor fitness dado que se busca minimizar ello
292     index_min = solucionesFitness.index(min(solucionesFitness))
293     print("Mejor Solucion:")
294     mostrarTabla(soluciones[index_min])
295     print("Mejor fitness"+ str(solucionesFitness[index_min]))
296     print("Costo total "+ str(solucionesMonto[index_min])+ " Dolares")
297     print("tiempos por Trabajo")
298     print(solucionesTiempos[index_min])
299     print("Tiempo tareas (Dias) "+ str(solucionesTiempo[index_min]) + " Dias")
300     print(f"Tiempo de corrida {tiempos_lista[index_min]:.10f} segundos")
301

```

```

302 ##funcion principal a ejecutar
303 PSO_iterator(100)

```

A.2. Código ACO

```

1  # coding: UTF-8
2
3  import random
4  from random import randint
5  import time
6  import collections
7  import itertools as it
8  import numpy as np
9  from ..help import combination, tool
10 from ..help import log
11 from tabulate import tabulate
12
13
14 class AcoStaffing:
15
16     def __init__(self, staff=None, skill=None, task=None, staff_skill=None, task_skill
17                 =None, task_precedent=None, mind_strategy=None):
18         """
19         Implementacion de Ant Colony Optimization para asignacion de personal a las
20         actividades
21         de un proyecto, sujeto:
22         * Ningun trabajador debe estar sobrecargado al mismo tiempo, es decir,
23         la suma de toda dedicacion a sus tareas asignadas debe ser maximo el 100%
24         de su dedicacion.
25         * Todas los skill de las tareas debe ser cubiertos:
26         * Seleccionar aquellas soluciones (asignaciones de personal) de minimo costo
27         (salario) y duracion.
28         (mas adelante se considererara la calidad)
29         """
30         # cada valor del parametro representa la cantidad de variables que puede
31         # aceptar o cambiar.
32         self.__staff = staff
33         self.__skill = skill
34         self.__task = task
35         self.__staff_skill = staff_skill
36         self.__task_skill = task_skill
37         self.__task_precedent = task_precedent
38         self.__mind_strategy = mind_strategy
39         # configuracion inicial que cambiara en las pruebas de rendimiento
40         self.config(ants=20, alpha=3, beta=1, rho=0.5, tau=0.4, quu=0.5, generation
41                   =1000)
42         self.weight_config(wcost=0.1, wdur=0.9)
43
44     def config(self, ants=None, alpha=None, beta=None, rho=None, tau=None, quu=None,
45               generation=None, is_normalize=None, is_stress=None):
46         self.__ants = ants # numero de hormigas
47         self.__alpha = alpha # coeficiente para el control de la
48                             # influencia/peso de la cantidad de feromonas
49         self.__beta = beta # coeficiente para el control de la
50                            # influencia/peso de la inversa (una ruta/distancia) de la distancia
51         self.__rho = rho # tasa de volatilidad de las feromonas
52         self.__tau = tau # valor inicial de la feromona
53         self.__quu = quu # valor que permite la explotacion o
54                          # exploracion de nuevas rutas (regla de proporcionalidad aleatoria)

```



```

45     self.__generation = generation      # numero de generaciones (numero de veces)
46         que hara el recorrido de todas las hormigas
47     self.__is_normalize = is_normalize   # aplicar la normalizacion del costo y
        duracion
48
49     self.__is_stress = is_stress         # permite habilitar/deshabilitar para las
        pruebas de esfuerzo
48
49     def weight_config(self, wcost=None, wdur=None, cost_fit=None, duration_fit=None):
50         self.__wcost = wcost             # Peso de importancia del costo
51         self.__wdur = wdur               # Peso de importancia de la duracion
52         self.__cost_fit = cost_fit        # Ajuste para el valor del costo
53         self.__duration_fit = duration_fit # Ajuste para el valor del duracion
54
55     def run(self):
56
57         # 01. Inicializar el valor feromonas
58         pheromone_values = self.generate_pheromone_values()
59         # print(pheromone_values)
60
61         # 02. Generar solucion aleatoria
62         current_solution, current_goals = self.generate_initial_solution()
63         if not self.__is_stress:
64             self.result_print_own(current_solution, current_goals)
65
66         # contador de generaciones
67         iteration = 0
68
69         # numero maximo de veces que se estanca la mejor solucion
70         max_wait = 200
71         count_wait = 0 # contador del numero maximo que se estanca la solucion
72
73         # 03. Ejecucion principal del algoritmo
74         while (count_wait <= max_wait) and (iteration <= self.__generation):
75             iteration += 1
76             max_wait += 1
77             for _ in range(self.__ants):
78                 ant_values = self.generate_ant_values()
79                 alloc_dedicate = np.asarray([0.0 for _ in range(len(self.__staff))])
80                 for i in range(len(self.__task)):
81                     quu = random.uniform(0, 1)
82                     heuristic_values = self.heuristic_values(alloc_dedicate, i)
83                     if quu > self.__quu:
84                         # exploramos un nuevo ruta
85                         ant_values[i] = self.explore_ant_path(heuristic_values)
86                     elif quu <= self.__quu:
87                         # explotamos un nuevo ruta
88                         ant_values[i] = self.exploit_ant_path(pheromone_values[i],
89                             heuristic_values)
89
90                 # Actualizacion local feromona
91                 alloc_dedicate = alloc_dedicate + [(self.__mind_strategy[np.argmax(
92                     node)][1] * self.__staff[n][2]) for n, node in enumerate(
93                     ant_values[i])]
94                 pheromone_values[i] = self.local_pheromones_update(
95                     pheromone_values[i], [np.argmax(node) for node in ant_values[i]
96                     ]], i)
97
98                 candidate_solution = np.asarray([(self.__mind_strategy[np.argmax(node)
99                     ])[1] * self.__staff[n][2]) for n, node in enumerate(staff)] for
100                     staff in ant_values])
101                 if self.assess_feasibility(candidate_solution):
102                     candidate_solution, candidate_goals = self.
103                         compute_candidate_solution(candidate_solution)
104                 # BUSCANDO EL MINIMO:
105                 if candidate_goals[0] < current_goals[0]:

```

```

99         max_wait = 0
100         if not self.__is_stress:
101             log.debug_timer("Mejor candidato:", "Fitness:",
102                             candidate_goals[0], "Cost:", candidate_goals[1], "
103                             Duration:", candidate_goals[2])
104             current_solution = candidate_solution
105             current_goals = candidate_goals
106             # Update feromona de forma global
107             pheromone_values = self.global_pheromones_update(
108                 pheromone_values, [[np.argmax(node) for node in staff] for
109                                     staff in ant_values], candidate_goals)
110
111         return current_solution, current_goals
112
113     # #####
114     # ##### INI:FUNCIONES PRINCIPALES #####
115     # #####
116
117     # ACTUALIZACION LOCAL DE LA FEROMONA
118     def local_pheromones_update(self, pheromone, path, task):
119         delta = self.delta_pheromone_local(path, task)
120         for idx, edge in enumerate(path):
121             for var in range(len(pheromone[idx])):
122                 if var == edge:
123                     pheromone[idx][var] = (1 - self.__rho) * pheromone[idx][var] +
124                         self.__rho*delta
125         return pheromone
126
127     # ACTUALIZACION LOCAL DE LA FEROMONA
128     def global_pheromones_update(self, pheromone, solution, candidate_goals):
129         delta = self.delta_pheromone_global(candidate_goals[1], candidate_goals[2],
130                                             candidate_goals[4], candidate_goals[5], candidate_goals[8])
131         for i in range(len(self.__task)):
132             path = solution[i]
133             for idx, edge in enumerate(path):
134                 for var in range(len(pheromone[i][idx])):
135                     if var == edge:
136                         pheromone[i][idx][var] = (1 - self.__rho) * pheromone[i][idx][
137                             var] + self.__rho*delta
138         return pheromone
139
140     # delta local de la feromona: compensa la calidad de le feromona de la solucion
141     # actual (path)
142     def delta_pheromone_local(self, path, task):
143         solution_matrix = [self.__mind_strategy[edge][1] * self.__staff[n][2] for n,
144                             edge in enumerate(path)]
145
146         staff_dur = np.sum(solution_matrix)
147         task_dur = (0 if staff_dur <= 0 else (self.__task[task][1] / staff_dur))
148         task_cost = np.sum([e[1]*solution_matrix[j]*task_dur for j, e in enumerate(
149             self.__staff)])
150         over_task = (task_dur-self.__task[task][2]) if task_dur > self.__task[task][2]
151         else 0
152
153         if not self.__is_normalize:
154             fitness_pre = self.__wcost*task_cost + self.__wdur*(task_dur + over_task)
155         else:
156             fitness_pre = self.__wcost*task_cost/self.__cost_fit + self.__wdur*(
157                 task_dur + over_task)/self.__duration_fit
158         return 0 if fitness_pre <= 0 else (fitness_pre**-1)
159
160     # delta global de la feromona: compensa la calidad de le feromona de la solucion
161     # de proyecto
162     def delta_pheromone_global(self, project_cost, project_dur, project_over_task,
163                               project_over_staff, project_cost_ajust):

```

```

150         if not self.__is_normalize:
151             fitness_pre = self.__wcost*project_cost + self.__wdur*(project_dur +
152                 project_over_task + project_over_staff)
153         else:
154             fitness_pre = self.__wcost*project_cost/self.__cost_fit + self.__wdur*(
155                 project_dur + project_over_task + project_over_staff)/self.
156                 __duration_fit
157         return 0 if fitness_pre <= 0 else (fitness_pre**-1)
158
159     # INFORMACION DE LA HEURISTICA
160     def heuristic_values(self, alloc_dedicate, task):
161         heuristic = np.asarray([[g[1] * self.__staff[i][2] for g in self.
162             __mind_strategy] for i in range(len(self.__staff))])
163
164         # calcular el total
165         for i, e in enumerate(heuristic):
166             alloc = alloc_dedicate[i]
167             if(alloc > 0.5):
168                 heuristic[i] = e + alloc_dedicate[i] - 0.5 # 50% de la dedicacion
169             else:
170                 heuristic[i] = e + alloc_dedicate[i]
171
172         total_tmp = np.sum(heuristic, axis=1)
173
174         task_skill = self.__task_skill[task, 1:]
175         for i, e in enumerate(heuristic):
176             total = total_tmp[i]
177             alloc = alloc_dedicate[i]
178             # calcular la heuristica de la dedicacion asignada
179             if(alloc > 0.5):
180                 heuristic[i] = (e/total)[::-1] # invertir para permitir mas
181                 dedicacion a los otros empleados
182             else:
183                 heuristic[i] = e/total
184
185         # validar que el skill de staff contenga minimo uno del skill task
186         staff_skill = self.__staff_skill[i, 1:]
187         if np.sum([int(staff_skill[k] and task_skill[k]) for k in range(len(
188             task_skill))]) <= 0:
189             # penaliza y asignar el valor inicial de la estrategia, que es 0%
190             heuristic[i][0] = np.max(heuristic[i])
191             for j in range(1, len(heuristic[i])):
192                 heuristic[i][j] = 0
193
194         return np.asarray(heuristic)
195
196     # ##### #
197     # ## IMPORTANTE ## #
198     # ##### #
199     # FUNCION DE APTITUP - FITNESS
200     def fitness_function(self, cost, duration):
201         # Maximizar
202         if not self.__is_normalize:
203             fitness_pre = self.__wcost*cost + self.__wdur*duration
204         else:
205             fitness_pre = self.__wcost*cost/self.__cost_fit + self.__wdur*duration/
206                 self.__duration_fit
207         return 0 if fitness_pre <= 0 else fitness_pre
208
209     # ##### #
210     # ## IMPORTANTE ## #
211     # ##### #
212
213     # FUNCION DE EVALUACION DE RESTRICCIONES
214     def assess_feasibility(self, solution_matrix):

```

```

208     # validar que cada tarea este a cargo de minimo un empleado
209     if np.min([np.sum(v) for _, v in enumerate(solution_matrix)]) > 0:
210         # validar que las personas asignadas a una tarea cubran las habilidades
211             requeridad (skills)
212         for i in range(len(self.__task)):
213             a = self.__task_skill[i, 1:]
214             c = [0 for _ in range(len(a))]
215             for j, val in enumerate(solution_matrix[i]):
216                 if val > 0:
217                     b = self.__staff_skill[j, 1:]
218                     c = [int(c[k] or b[k]) for k in range(len(b))]
219             d = [int(c[k] and a[k]) for k in range(len(a))]
220             if np.sum(a) != np.sum(d):
221                 return False
222
223     return True
224
225     # #####
226     # ##### FIN:FUNCIONES PRINCIPALES #####
227     # #####
228
229     # #####
230     # ##### FIN: FUNCIONES SECUNDARIAS #####
231     # #####
232
233     def compute_candidate_solution(self, solution_matrix):
234
235         # clcular la duracion del proyecto
236         project_cost_total, task_matrix_dur = self.project_cost_calc(solution_matrix)
237
238         # caclular el inicio y el final de las actividades
239         project_dur_total, tpg_scheduler = self.project_duration_calc(task_matrix_dur)
240
241         # calcular el inicio y el final maximo de las actividades
242         project_max_dur_total, tpg_max_scheduler = self.project_max_duration_calc()
243
244         # calcular trabajo de sobretiempo en el personal y tareas
245         project_overtime_task_total, project_overeffort_staff_total,
246             project_cost_ajust = self.project_over_calc(task_matrix_dur, tpg_scheduler
247                 , solution_matrix)
248
249         fitness_value = self.fitness_function(project_cost_total, project_dur_total)
250
251         # ajuste al costo del proyecto por el exceso de horas del personal (pago doble
252             , la primera se dio en calculo inicial)
253         # si el empleado esta en n tareas al mismo tiempo y sobre pasa su capacida se
254             le paga n veces (este parte se tiene que MEJORAR)
255         project_cost_total += project_cost_ajust
256
257         return solution_matrix, np.asarray([fitness_value,
258             project_cost_total,
259             project_dur_total,
260             project_max_dur_total,
261             project_overtime_task_total,
262             project_overeffort_staff_total,
263             np.asarray(task_matrix_dur),
264             np.asarray(tpg_scheduler),
265             project_cost_ajust])
266
267     def exploit_ant_path(self, pheromone_values, heuristic_values):
268         up = (pheromone_values ** self.__alpha) * (heuristic_values ** self.__beta)
269         down = [[u if u > 0 else 1.0] for u in np.sum(up, axis=1)]
270         return up/down

```

```

268 def explore_ant_path(self, heuristic_values):
269     up = (heuristic_values ** self.__beta)
270     down = [[u if u > 0 else 1.0] for u in np.sum(up, axis=1)]
271     random_explore = np.asarray([np.asarray(np.random.uniform(0, 1, len(self.
272         __mind_strategy))) for _ in range(len(self.__staff))])
273     return random_explore*(up/down)
274
275 def generate_ant_values(self):
276     return self.generate_staff_task_matrix(np.asarray([0.0 for _ in range(len(self
277         __mind_strategy))]))
278
279 def generate_pheromone_values(self):
280     return self.generate_staff_task_matrix(np.asarray([0.0 for _ in range(len(self
281         __mind_strategy))]))
282
283 def generate_staff_task_matrix(self, strategy):
284     return np.asarray([strategy for _ in range(len(self.__staff))] for _ in range
285         (len(self.__task)))
286
287 def generate_initial_solution(self):
288     solution_matrix = []
289     while True:
290         solution_matrix = []
291         for i in range(len(self.__task)):
292             task_skill = self.__task_skill[i, 1:]
293             ded = []
294             for j in range(len(self.__staff)):
295                 staff_skill = self.__staff_skill[j, 1:]
296                 k = 0
297
298                 # validar que el skill de empleado contenga minimo uno del skill
299                 task
300                 if np.sum([int(staff_skill[k] and task_skill[k]) for k in range(
301                     len(task_skill))]) > 0:
302                     k = randint(0, len(self.__mind_strategy) - 1)
303                     ded.append(self.__mind_strategy[k][1] * self.__staff[j][2])
304             solution_matrix.append(ded)
305
306             if self.assess_feasibility(solution_matrix):
307                 break
308
309     return self.compute_candidate_solution(np.asarray(solution_matrix))
310
311 def project_cost_calc(self, solution_matrix):
312     # calcular duracion del proyecto
313     project_cost_total = 0
314     task_matrix_dur = []
315     for i in range(len(self.__task)):
316         #staff_dur = np.sum([e[2]*solution_matrix[i][j] for j, e in enumerate(self
317             __staff)])
318         staff_dur = np.sum([solution_matrix[i][j] for j in range(len(self.__staff)
319             )])
320         task_dur = (0 if staff_dur <= 0 else (self.__task[i][1] / staff_dur))
321         task_matrix_dur.append(task_dur)
322         project_cost_total += np.sum([e[1]*solution_matrix[i][j]*task_dur for j, e
323             in enumerate(self.__staff)])
324
325     return project_cost_total, task_matrix_dur
326
327 def project_duration_calc(self, task_matrix_dur):
328     # calcular el inicio y el final de las actividades
329     tpg_scheduler = []
330     project_dur_total = 0
331     for i in range(len(self.__task)):

```

```

324         precedents = self.__task_precedent[i, 1:]
325
326         start = 0
327         for j in range(len(self.__task)):
328             if precedents[j] == 1:
329                 start = max(start, tpg_scheduler[j][1])
330         end = start + task_matrix_dur[i]
331         tpg_scheduler.append([start, end])
332         project_dur_total = end
333     return project_dur_total, tpg_scheduler
334
335     def project_max_duration_calc(self):
336         # calcular el inicio y el final maximo de las actividades
337         tpg_max_scheduler = []
338         project_max_dur_total = 0
339         for i in range(len(self.__task)):
340             precedents = self.__task_precedent[i, 1:]
341             start = 0
342             for j in range(len(self.__task)):
343                 if precedents[j] == 1:
344                     start = max(start, tpg_max_scheduler[j][1])
345             end = start + self.__task[i][2]
346             tpg_max_scheduler.append([start, end])
347             project_max_dur_total = end
348
349     return project_max_dur_total, tpg_max_scheduler
350
351     def project_over_calc(self, task_matrix_dur, tpg_scheduler, solution_matrix):
352         # calcular trabajo de sobretiempo en el staff y tareas
353         # de tareas
354         project_overtime_task_total = 0
355         for i, v in enumerate(task_matrix_dur):
356             time_max = self.__task[i][2]
357             if v > time_max:
358                 project_overtime_task_total += (v - time_max)
359
360         # de personas
361         # MEJORAR el calculo para tareas que se sobrepones, usando una unidad de
362         # tiempo (por ejemplo dias)
363         # MEJORAR esta parte para que no se cuente doble, ademas se sugiere ponerlo
364         # dentro del fitness
365         # se resuelve con una funcion recursiva, iterando la activades cruzadas hasta
366         # el final
367         tpg_scheduler_tmp = []
368         project_overeffort_staff_total = 0
369         project_cost_ajust = 0
370         for i, e in enumerate(self.__staff):
371             tpg_scheduler_tmp = np.copy(tpg_scheduler)
372             over_effort = 0
373             cost_effort = 0
374             for j in range(len(self.__task)):
375                 end = tpg_scheduler_tmp[j][1]
376                 if end - tpg_scheduler_tmp[j][0] > 0:
377                     over_effort = solution_matrix[j][i]
378                     for k in range(j+1, len(self.__task)):
379                         if end - tpg_scheduler_tmp[k][0] > 0 and tpg_scheduler_tmp[k]
380                             [1] > tpg_scheduler_tmp[k][0]:
381                             dif = min(end - tpg_scheduler_tmp[k][0], task_matrix_dur[k]

```

```

382         cost_effort += dif_over*e[1] # calculando el costo en
                                     exceso
383         over_effort -= dif_over
384     if cost_effort > 0:
385         project_overeffort_staff_total += (cost_effort/e[1])
386         project_cost_ajust += cost_effort
387
388     return project_overtime_task_total, project_overeffort_staff_total,
        project_cost_ajust
389
390 # #####
391 # ##### FIN: FUNCIONES SECUNDARIAS #####
392 # #####
393
394 # #####
395 # ##### INI: FUNCIONES DE APOYO #####
396 # #####
397
398 # Funcion para imprimir de manera interna
399 def result_print_own(self, solution, goal):
400     print(20*" ", 60*" ", 20*" ")
401     print(20*" ", 60*" ", 20*" ")
402     print(20*" ", 60*" ", 20*" ")
403     print(20*" ", "MATRIZ DE ASIGNACION", 20*" ")
404     print(solution)
405     print(20*" ", "VALOR FITNESS", 20*" ")
406     print(goal[0])
407     print(20*" ", "VALOR COST", 20*" ")
408     print(goal[1])
409     print(20*" ", "VALOR DURACION DEL PROYECTO", 20*" ")
410     print(goal[2])
411     print(20*" ", "VALOR DURACION MAXIMA", 20*" ")
412     print(goal[3])
413     print(20*" ", "SOBRETIEMPO DE TAREAS", 20*" ")
414     print(goal[4], )
415     print(20*" ", "SOBRETIEMPO DE PERSONAL", 20*" ")
416     print("Esfuerzo extra:", goal[5], "Costo extra:", goal[8])
417     print(20*" ", "DURACION DE CADA TAREA", 20*" ")
418     print(goal[6])
419     print(20*" ", "RUTA CRITICA DE TAREAS (CRONOGRAMA)", 20*" ")
420     print(goal[7])
421     print(20*" ", 60*" ", 20*" ")
422     print(20*" ", 60*" ", 20*" ")
423     print(20*" ", 60*" ", 20*" ")
424
425 # Funcion para imprimir de manera externa
426 def result_print(self, solution, goal, task_desc, staff_desc):
427     print(20*" ", 60*" ", 20*" ")
428     print(20*" ", 60*" ", 20*" ")
429     print(20*" ", 60*" ", 20*" ")
430     print(20*" ", "MATRIZ DE ASIGNACION", 20*" ")
431     sol = [list(map(str, x)) for x in np.array(solution.transpose())]
432     sol = [[10*" " + i for i in e] for e in sol]
433     sol = np.insert(sol, 0, task_desc[:, 1:].transpose()[0], axis=0)
434     sol = np.insert(sol, 0, np.insert(staff_desc[:, 1:], 0, " ", axis=0).
        transpose()[0], axis=1)
435
436     table = tabulate(sol, headers="firstrow", tablefmt="grid")
437     tabulate.PRESERVE_WHITESPACE = True
438     print(table)
439     print(20*" ", "VALOR FITNESS", 20*" ")
440     print(goal[0])
441     print(20*" ", "VALOR COST", 20*" ")
442     print(goal[1])
443     print(20*" ", "VALOR DURACION DEL PROYECTO", 20*" ")

```

```

444     print(goal[2])
445     print(20*" ", "VALOR DURACION MAXIMA", 20*" ")
446     print(goal[3])
447     print(20*" ", "SOBRETIEMPO DE TAREAS", 20*" ")
448     print(goal[4])
449     print(20*" ", "SOBRETIEMPO DE PERSONAL", 20*" ")
450     print("Esfuerzo extra:", goal[5], "Costo extra:", goal[8])
451     print(20*" ", "DURACION DE CADA TAREA", 20*" ")
452     print(goal[6])
453     print(20*" ", "RUTA CRITICA DE TAREAS (CRONOGRAMA)", 20*" ")
454     print(goal[7])
455     print(20*" ", 60*" ", 20*" ")
456     print(20*" ", 60*" ", 20*" ")
457     print(20*" ", 60*" ", 20*" ")
458
459     # #####
460     # ##### FIN: FUNCIONES DE APOYO #####
461     # #####

```