In [ ]:

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "ae5374d7",
   "metadata": {},
   "source": [
    "# 📊 Smart Task Assignment Dashboard\n",
    "\n",
    "This notebook runs Greedy, Genetic, and Firefly algorithms to assign tasks to
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "83bca36f",
   "metadata": {},
   "outputs": [],
   "source": [
    "import pandas as pd\n",
    "import matplotlib.pyplot as plt\n",
    "from collections import Counter\n",
    "import random\n",
    "import copy\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "85b1f364",
   "metadata": {},
   "outputs": [],
   "source": [
    "class Employee:\n",
    "    def __init__(self, name, skills, max_hours):\n",
    "        self.name = name.strip()\n",
    "        self.skills = [s.strip() for s in skills.split(';')]\n",
    "        self.max_hours = int(max_hours)\n",
    "        self.assigned_hours = 0\n",
    "        self.assigned_tasks = 0\n",
    "\n",
    "class Task:\n",
    "    def __init__(self, name, skill, hours):\n",
    "        self.name = name.strip()\n",
    "        self.skill = skill.strip()\n",
    "        self.hours = int(hours)\n",
    "\n",
    "def load_employees(filename=\"employees.txt\"):\n",
    "    employees = []\n",
    "    with open(filename, 'r') as f:\n",
    "        for line in f:\n",
    "            if not line.strip(): continue\n",
    "            parts = line.strip().split(',')\n",
    "            if len(parts) != 3: continue\n",
```

```
  "            name, skills, max_hours = parts\n",
  "            employees.append(Employee(name, skills, max_hours))\n",
  "    return employees\n",
  "\n",
  "def load_tasks(filename=\"tasks.txt\"):\n",
  "    tasks = []\n",
  "    with open(filename, 'r') as f:\n",
  "        for line in f:\n",
  "            if not line.strip(): continue\n",
  "            parts = line.strip().split(',')\n",
  "            if len(parts) != 3: continue\n",
  "            name, skill, hours = parts\n",
  "            tasks.append(Task(name, skill, hours))\n",
  "    return tasks\n"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "b21ac966",
 "metadata": {},
 "outputs": [],
 "source": [
  "def run_greedy(employees, tasks):\n",
  "    employees = copy.deepcopy(employees)\n",
  "    assignment = []\n",
  "    for task in tasks:\n",
  "        suitable = [e for e in employees if task.skill in e.skills and e.assig
  "        if suitable:\n",
  "            selected = min(suitable, key=lambda e: e.assigned_hours)\n",
  "        else:\n",
  "            fallback = [e for e in employees if e.assigned_hours + task.hours
  "            selected = min(fallback, key=lambda e: e.assigned_hours) if fallba
  "        if selected:\n",
  "            selected.assigned_hours += task.hours\n",
  "            selected.assigned_tasks += 1\n",
  "        assignment.append(selected)\n",
  "    return assignment, employees\n"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "6415f354",
 "metadata": {},
 "outputs": [],
 "source": [
  "class GeneticAlgorithm:\n",
  "    def __init__(self, employees, tasks, population_size=6, generations=80, mu
  "        self.employees = employees\n",
  "        self.tasks = tasks\n",
  "        self.population_size = population_size\n",
  "        self.generations = generations\n",
  "        self.mutation_rate = mutation_rate\n",
  "\n",
  "    def create_individual(self):\n",
```

```
   "        return [random.choice(self.employees) for _ in self.tasks]\n",
   "\n",
   "    def calculate_fitness(self, individual):\n",
   "        temp_hours = {emp.name: 0 for emp in self.employees}\n",
   "        score = 0\n",
   "        for task, emp in zip(self.tasks, individual):\n",
   "            has_skill = task.skill in emp.skills\n",
   "            has_time = temp_hours[emp.name] + task.hours <= emp.max_hours\n",
   "            if has_skill and has_time:\n",
   "                score += 10\n",
   "                temp_hours[emp.name] += task.hours\n",
   "            elif has_skill:\n",
   "                score += 5\n",
   "        return score\n",
   "\n",
   "    def mutate(self, individual):\n",
   "        if random.random() < self.mutation_rate:\n",
   "            idx = random.randint(0, len(individual) - 1)\n",
   "            individual[idx] = random.choice(self.employees)\n",
   "        return individual\n",
   "\n",
   "    def crossover(self, parent1, parent2):\n",
   "        point = random.randint(1, len(parent1) - 1)\n",
   "        return parent1[:point] + parent2[point:]\n",
   "\n",
   "    def select_parents(self, population, fitnesses):\n",
   "        total_fitness = sum(fitnesses)\n",
   "        if total_fitness <= 0:\n",
   "            return random.sample(population, 2)\n",
   "        return random.choices(population, weights=fitnesses, k=2)\n",
   "\n",
   "    def run(self):\n",
   "        population = [self.create_individual() for _ in range(self.population_
   "        best = max(population, key=self.calculate_fitness)\n",
   "        best_fitness = self.calculate_fitness(best)\n",
   "\n",
   "        for _ in range(self.generations):\n",
   "            fitnesses = [self.calculate_fitness(ind) for ind in population]\n"
   "            new_population = []\n",
   "            for _ in range(self.population_size):\n",
   "                parent1, parent2 = self.select_parents(population, fitnesses)\
   "                child = self.crossover(parent1, parent2)\n",
   "                child = self.mutate(child)\n",
   "                new_population.append(child)\n",
   "            population = new_population\n",
   "            current_best = max(population, key=self.calculate_fitness)\n",
   "            current_fitness = self.calculate_fitness(current_best)\n",
   "            if current_fitness > best_fitness:\n",
   "                best, best_fitness = current_best, current_fitness\n",
   "\n",
   "        return best\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
```

```
 "id": "4e8ebcac",
 "metadata": {},
 "outputs": [],
 "source": [
  "class FireflyAlgorithm:\n",
  "    def __init__(self, employees, tasks, population_size=10, generations=50, a
  "        self.employees = employees\n",
  "        self.tasks = tasks\n",
  "        self.population_size = population_size\n",
  "        self.generations = generations\n",
  "        self.alpha = alpha\n",
  "        self.beta0 = beta0\n",
  "        self.gamma = gamma\n",
  "\n",
  "    def create_solution(self):\n",
  "        return [random.choice(self.employees) for _ in self.tasks]\n",
  "\n",
  "    def fitness(self, solution):\n",
  "        temp_hours = {e.name: 0 for e in self.employees}\n",
  "        score = 0\n",
  "        for task, emp in zip(self.tasks, solution):\n",
  "            has_skill = task.skill in emp.skills\n",
  "            has_time = temp_hours[emp.name] + task.hours <= emp.max_hours\n",
  "            if has_skill and has_time:\n",
  "                score += 10\n",
  "                temp_hours[emp.name] += task.hours\n",
  "            elif has_skill:\n",
  "                score += 5\n",
  "        return score\n",
  "\n",
  "    def distance(self, sol1, sol2):\n",
  "        return sum(e1 != e2 for e1, e2 in zip(sol1, sol2))\n",
  "\n",
  "    def move_firefly(self, firefly_i, firefly_j):\n",
  "        new_solution = []\n",
  "        for i in range(len(self.tasks)):\n",
  "            if firefly_i[i] != firefly_j[i] and random.random() < self.beta0 *
  "                new_solution.append(firefly_j[i])\n",
  "            else:\n",
  "                if random.random() < self.alpha:\n",
  "                    new_solution.append(random.choice(self.employees))\n",
  "                else:\n",
  "                    new_solution.append(firefly_i[i])\n",
  "        return new_solution\n",
  "\n",
  "    def run(self):\n",
  "        population = [self.create_solution() for _ in range(self.population_si
  "        fitnesses = [self.fitness(sol) for sol in population]\n",
  "\n",
  "        for _ in range(self.generations):\n",
  "            for i in range(self.population_size):\n",
  "                for j in range(self.population_size):\n",
  "                    if fitnesses[j] > fitnesses[i]:\n",
  "                        population[i] = self.move_firefly(population[i], popul
  "                        fitnesses[i] = self.fitness(population[i])\n",
  "\n",
```

```
   "        best_index = fitnesses.index(max(fitnesses))\n",
   "        return population[best_index]\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "id": "e9cce521",
  "metadata": {},
  "outputs": [],
  "source": [
   "def summarize_and_plot(tasks, assignments, title):\n",
   "    records = []\n",
   "    for task, emp in zip(tasks, assignments):\n",
   "        if emp:\n",
   "            direct = task.skill in emp.skills\n",
   "            records.append({\n",
   "                \"Task\": task.name,\n",
   "                \"Skill\": task.skill,\n",
   "                \"Hours\": task.hours,\n",
   "                \"Assigned To\": emp.name,\n",
   "                \"Type\": \"Direct\" if direct else \"Fallback\"\n",
   "            })\n",
   "        else:\n",
   "            records.append({\n",
   "                \"Task\": task.name,\n",
   "                \"Skill\": task.skill,\n",
   "                \"Hours\": task.hours,\n",
   "                \"Assigned To\": \"❌ None\",\n",
   "                \"Type\": \"Unassigned\"\n",
   "            })\n",
   "\n",
   "    df = pd.DataFrame(records)\n",
   "    display(df)\n",
   "    print(f\"✅ {title} - Direct: {(df['Type'] == 'Direct').sum()}, Fallback:
  "\n",
   "    fig, axs = plt.subplots(1, 2, figsize=(10, 4))\n",
   "    skill_counts = Counter(df[\"Skill\"])\n",
   "    axs[0].pie(skill_counts.values(), labels=skill_counts.keys(), autopct='%1.
   "    axs[0].set_title(\"Skill Distribution\")\n",
   "    hours = df.groupby(\"Assigned To\")[\"Hours\"].sum()\n",
   "    axs[1].bar(hours.index, hours.values)\n",
   "    axs[1].set_title(\"Employee Load\")\n",
   "    axs[1].tick_params(axis='x', rotation=45)\n",
   "    fig.tight_layout()\n",
   "    plt.show()\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "id": "4e350d3f",
  "metadata": {},
  "outputs": [],
  "source": [
   "employees = load_employees()\n",
```

```
    "tasks = load_tasks()\n",
    "\n",
    "# Greedy\n",
    "greedy_result, _ = run_greedy(employees, tasks)\n",
    "summarize_and_plot(tasks, greedy_result, \"Greedy Algorithm\")\n",
    "\n",
    "# Genetic\n",
    "ga = GeneticAlgorithm(employees, tasks)\n",
    "genetic_result = ga.run()\n",
    "summarize_and_plot(tasks, genetic_result, \"Genetic Algorithm\")\n",
    "\n",
    "# Firefly\n",
    "fa = FireflyAlgorithm(employees, tasks)\n",
    "firefly_result = fa.run()\n",
    "summarize_and_plot(tasks, firefly_result, \"Firefly Algorithm\")\n"
   ]
  }
 ],
 "metadata": {},
 "nbformat": 4,
 "nbformat_minor": 5
}
```