به نام خدا

دوربین‌های نظارتی

امیرحسین علی مردانی هروی

منیره صدربزاز

مبین صمدی

فهرست

مقدمه

در این ارائه قصد داریم مسئله کاربردی دوربین‌های نظارتی را بررسی کنیم.

دوربین‌های نظارتی به‌عنوان یکی از ابزارهای کلیدی در افزایش امنیت و مدیریت شهری، نقش بسزایی در پایش محیط‌های عمومی، خصوصی و صنعتی ایفا می‌کنند. این سیستم‌ها، با قابلیت ثبت و ضبط مداوم داده‌های بصری، امکان نظارت بر رفتارهای انسانی، پیشگیری از وقوع جرایم، و تحلیل شرایط اضطراری را فراهم می‌سازند. با این حال، توسعه و استفاده گسترده از این فناوری‌ها چالش‌های قابل‌توجهی را به همراه دارد. از جمله این چالش‌ها می‌توان به هزینه های نصب و نگهداری اشاره کرد. هدف از این پژوهش کاهش هزینه ها با مینیمم کردن تعداد دوربین‌ها است.

اهداف پژوهش

از اهداف اصلی این پژوهش می‌توان به کاهش هزینه انرژی (در صورت استفاده از دوربین‌های پرمصرف)، پوشش مطلوب در ساعات مختلف روز و شرایط آب و هوایی مختلف، پوشش کامل نواحی با کمترین تعداد دوربین، افزایش وضوح و کیفیت تصاویر در مناطق کلیدی و کاهش هزینه‌های نصب و نگهداری اشاره کرد. در این مسئله با ارائه مدل، بهینه‌ترین حالت مناسب برای منیمیم‌سازی تعداد دوربین‌های نصبی و پوشش حداکثری را پیدا می‌کنیم.

کاربردها

1. طراحی موقعیت پهپادهای نظارتی[1]

این تحقیق بر طراحی نقاط مناسب پهپادها برای نظارت بهینه بر مناطق حساس با مصرف انرژی کمتر تمرکز دارد.

روش‌های علمی استفاده‌شده :مدل‌سازی ریاضی سه‌بعدی :تعریف نقاط به‌صورت سه‌بعدی با توابع هدف چندگانه :کاهش مصرف انرژی .کاهش خطرات پرواز .افزایش اولویت مناطق حساس .الگوریتم :PSO این الگوریتم موقعیت و سرعت پهپادها را برای یافتن بهترین مکان به‌روزرسانی می‌کند .به‌روزرسانی اولویت مناطق :تغییر اولویت نظارت بر اساس وقایع جدید، مانند تشخیص نفوذ پهپادهای غیرمجاز

.نتایج :نقاط بهینه که پوشش مناطق حساس را افزایش داده‌اند .کاهش مصرف انرژی پهپادها و افزایش ایمنی در پرواز

2. برنامه ضد شکار غیرقانونی[3]

این تحقیق در آفریقای جنوبی برای حفاظت از کرگدن‌ها توسعه یافته و سیستم بهینه‌ای را ارائه کرده است که با تعداد دوربین کمتر، پوشش بیشتری فراهم می‌کند.

روش‌های علمی استفاده‌شده :تعریف مناطق نظارتی :CZ1 :پوشش کامل پیرامون منطقه (60 متر) :CZ2 .مناطق پرخطر با احتمال نفوذ :CZ3 .مناطق داخلی برای نظارت بر حیوانات .مدل‌سازی ریاضی :MCLP :یافتن مکان‌هایی با پوشش حداکثری :BCLP .تضمین پوشش دوگانه :AMCLP .بهینه‌سازی زاویه دید دوربین‌ها .الگوریتم ژنتیکی :NSGA-II نمایش راه‌حل‌ها به‌صورت کروموزوم شامل مکان دوربین‌ها .عملیات تقاطع و جهش برای تولید و کشف راه‌حل‌های جدید .انتخاب بهترین راه‌حل‌ها بر اساس معیارهای چندهدفه.

نتایج :کاهش تعداد دوربین‌ها و هزینه‌های نصب .پوشش کامل مناطق و بهبود نظارت شبانه در .CZ2 بهبود کیفیت پوشش مناطق پرخطر تا 31.6٪ با تنظیم زاویه دید

توضیحات کامل الگوریتم Tabu Search برای مسئله پوشش نقاط با دوربین‌ها

Tabu Searchیک الگوریتم جستجوی محلی است که برای حل مسائل بهینه‌سازی پیچیده به کار می‌رود. این الگوریتم برای جلوگیری از گیر کردن در بهینه‌های محلی از دو ویژگی اصلی استفاده می‌کند:

1. تابو لیست :(Tabu List) مجموعه‌ای از حرکت‌ها (یا انتخاب‌ها) که اخیراً انجام شده‌اند و نمی‌توانند مجدداً در تکرارهای بعدی استفاده شوند. این کار به جلوگیری از بازگشت به وضعیت‌های قبلی کمک می‌کند.
2. معیار آرزو :(Aspiration Criteria) اگر یک حرکت تابو، بهبود بهینه‌تری نسبت به بهترین راه‌حل پیدا کند، می‌تواند پذیرفته شود. این امر به الگوریتم این اجازه را می‌دهد که از بهینه‌های محلی فراتر رود.

الگوریتم Tabu Search به طور کلی با جستجوی محلی (Local Search) مشابه است، اما با افزودن تابو لیست و معیار آرزو، دامنه جستجو گسترش می‌یابد و از گیر کردن در بهینه‌های محلی جلوگیری می‌شود.

مراحل الگوریتم Tabu Search برای مسئله پوشش نقاط با دوربین‌ها:

ورودی:

- دوربین‌ها :(cameras) یک دیکشنری که برای هر دوربین، لیستی از نقاطی که می‌تواند پوشش دهد، دارد.
- نقاط :(points) لیستی از نقاطی که باید تحت پوشش قرار گیرند.
- حداکثر تعداد تکرارها :(max_iter) حداکثر تعداد تکرارهای الگوریتم.
- اندازه تابو لیست :(tabu_size) تعداد حرکات که باید در تابو لیست ذخیره شود.
- معیار آرزو :(aspiration_criteria) آیا باید از معیار آرزو استفاده شود یا نه.

خروجی:

- بهترین راه‌حل :(best_solution) مجموعه‌ای از دوربین‌هایی که نقاط را پوشش می‌دهند و تعداد آن‌ها کمترین است.
- بهترین هزینه :(best_cost) تعداد دوربین‌های استفاده‌شده برای پوشش تمامی نقاط.
- زمان اجرا :(execution_time) مدت زمانی که الگوریتم برای رسیدن به بهترین راه‌حل صرف کرده است.

مراحل الگوریتم:

1. ایجاد راه‌حل اولیه:
   - یک راه‌حل تصادفی تولید می‌شود که شامل مجموعه‌ای از دوربین‌ها است که تمامی نقاط را پوشش دهند.
2. تابو لیست:
   - لیستی از حرکت‌ها (تغییرات در مجموعه دوربین‌ها) که اخیراً انجام شده است، نگهداری می‌شود.
   - حرکات جدید نمی‌توانند به طور مکرر انتخاب شوند مگر اینکه از معیار آرزو عبور کنند.
3. تولید همسایگی:

- در هر تکرار، همسایه‌ها به وسیله تغییر انتخاب دوربین‌ها ایجاد می‌شوند. به این صورت که برای هر دوربین:
  - اگر دوربین در راه‌حل فعلی وجود دارد، آن را حذف می‌کنیم.
  - اگر دوربین در راه‌حل فعلی وجود ندارد، آن را اضافه می‌کنیم.
- همسایگان تولید شده باید تمام نقاط را پوشش دهند.

4. انتخاب بهترین همسایه:

- هر همسایه ارزیابی می‌شود (شامل تعداد دوربین‌های استفاده‌شده و تعداد نقاط تحت پوشش).
- اگر همسایه‌ای تابو نباشد یا از معیار آرزو عبور کند (یعنی تعداد دوربین‌ها کم‌تر از بهترین راه‌حل باشد)، انتخاب می‌شود.

5. به روز رسانی تابو لیست:

- حرکات جدید به تابو لیست اضافه می‌شوند و اگر تابو لیست پر شود، حرکت‌های قدیمی حذف می‌شوند.

6. به روز رسانی بهترین راه‌حل:

- اگر همسایه جدید بهتر از بهترین راه‌حل پیدا شده باشد، آن را به عنوان بهترین راه‌حل به روز می‌کنیم.

7. شرایط پایان:

- الگوریتم پس از رسیدن به حداکثر تعداد تکرارها متوقف می‌شود.

توضیح شبه کد

```
تابو_Search(cameras, points, max_iter, tabu_size, aspiration_criteria):
  current_solution = generate_initial_solution(cameras, points)
  best_solution = current_solution
  best_cost, uncovered_points = evaluate_solution(best_solution, cameras, points)

  tabu_list = [] // لیست تابو که حرکت‌های اخیر را ذخیره می‌کند
  iteration = 0

  while iteration < max_iter:
    neighborhood = [] // لیست همسایگان (حل‌های جدید)

    // تولید همسایگان از راه‌حل فعلی
    برای هر دوربین در cameras:
      neighbor = تغییر دوربین‌ها در current_solution (اضافه یا حذف کردن)
      اگر neighbor تمام نقاط را پوشش می‌دهد:
        neighborhood به neighbor اضافه کن

    بهترین همسایه = None
    بهترین ارزیابی = بی‌نهایت
    تابو violations = 0

    // ارزیابی هر همسایه
    برای هر neighbor در neighborhood:
      ارزیابی_solution = ارزیابی (neighbor)

      // بررسی تابو لیست و معیار آرزو
      اگر neighbor در تابو لیست نیست یا (aspiration_criteria و ارزیابی_solution بهتر از بهترین_cost باشد):
        اگر ارزیابی_solution بهتر از بهترین ارزیابی باشد:
          بهترین_solution = ارزیابی
          بهترین همسایه = neighbor

    اگر بهترین همسایه پیدا شد:
      current_solution = بهترین همسایه
      current_cost, uncovered_points = ارزیابی_solution(best_neighbor)

      اگر current_cost کمتر از بهترین_cost باشد:
        best_solution = current_solution
        بهترین_cost = current_cost
      else:
        تابو violations += 1

    // به روز رسانی تابو لیست
    به تابو لیست current_solution اضافه کردن
    اگر طول تابو لیست بیشتر از tabu_size باشد:
      حذف اولین حرکت از تابو لیست
```

```
    iteration += 1
برگرداندن best_solution و best_cost
```

مدل سازی عدد صحیح

مدل برنامه‌ریزی عدد صحیح برای مسئله نصب دوربین‌های نظارتی

تعریف مجموعه‌ها

P: مجموعه نقاط یا مناطقی که باید پوشش داده شوند

C: مجموعه مکان‌های بالقوه برای نصب دوربین‌ها.

متغیرهای تصمیم‌گیری

$$x_i \in \{0,1\}$$

متغیر تصمیم‌گیری که نشان می‌دهد آیا دوربین C_i نصب شده است یا خیر. اگر دوربین نصب شود مقدار آن 1 و در غیر این صورت 0 است.

تابع هدف

هدف، کمینه‌سازی تعداد یا هزینه دوربین‌های نصب‌شده است:

$$Z = \sum_{i=1}^{m} w_i \cdot x_i$$

که در آن:

w_i هزینه نصب دوربین c_i است

اگر هزینه نصب تمام دوربین ها یکسان باشد، رابطه به صورت زیر ساده میشود.

$$Z = \sum_{i=1}^{m} x_i$$

این فرمول تعداد دوربین‌های نصب‌شده را کمینه می‌کند.

محدودیت‌ها

محدودیت پوشش کامل

هر نقطه باید حداقل توسط یک دوربین پوشش داده شود:

$$\sum_{i \in C} A[j,i] \cdot x_i \geq 1 \quad \forall j \in P$$

که در آن:

$A[i,j]$ مقدار 1 را دارد اگر نقطه P_j توسط دوربین c_i پوشش داده شود. در غیر اینصورت مقدار 0 را دارد.

محدودیت متغیرهای تصمیم‌گیری

متغیرهای تصمیم‌گیری باید دودویی باشند:

$$x_i \in \{0,1\} \quad \forall i \in C$$

این مدل یک فرمول‌بندی برنامه‌ریزی عدد صحیح است که هدف آن کمینه‌سازی تعداد و در نهایت هزینه دوربین‌ها است، در حالی که پوشش کامل تمام نقاط مورد نیاز را تضمین می‌کند.

مدل ترکیبیاتی

تعریف مجموعه‌ها:

- P: مجموعه نقاط یا مناطقی که باید پوشش داده شوند.

$$S: Set\ of\ all\ cameras(c_1, c_2, \dots, c_m).$$

- U: مجموعه‌ای از نقاط که باید پوشش داده شوند.

$$U: Set\ of\ all\ points(p_1, p_2, \dots, p_n)\ that\ need\ coverage.$$

- $A[j,i]$: ماتریس پوشش که $A[j,i] = 1$ اگر دوربین c_i نقطه P_j را پوشش دهد، در غیر این صورت $A[j,i] = 0$

متغیرهای تصمیم‌گیری:

- $\lambda \subseteq S$: زیرمجموعه‌ای از دوربین‌ها که انتخاب می‌شود (یعنی دوربین‌هایی که به پوشش نقاط کمک می‌کنند).
- $c(\lambda)$: هزینه مجموعه λ ، که ممکن است شامل هزینه نصب دوربین‌ها یا تعداد دوربین‌های انتخابی باشد.

مجموعه بهینه‌سازی:

$$\Omega = \{\lambda \mid \lambda \subseteq S,$$

$$\Omega = \left\{\lambda \mid \lambda \subseteq S,\ \bigcup_{c\in\lambda} \text{coverage}(c) = U\right\}$$

که در آن:

$\lambda \subseteq S$ مجموعه‌ای از دوربین‌های انتخاب شده است.

$\bigcup_{c\in\lambda} \text{coverage}(c) = U$ تضمین می‌کند که مجموعه انتخاب شده تمام نقاط موجود در U را پوشش دهد.

تابع هدف (هزینه انتخاب):

$$c(\lambda) = \sum_{i\in\lambda} w_i$$

که در آن w_i هزینه مربوط به نصب دوربین c_i است. اگر هزینه‌ها برای تمام دوربین‌ها یکسان باشد، تابع هدف فقط به کمینه‌سازی تعداد دوربین‌ها می‌پردازد.

مسئله بهینه‌سازی:

$$\min_{\lambda\in\Omega} c\ (\lambda)$$

$$\bigcup_{c\in\lambda} \text{coverage}(c) = U$$

هدف کمینه‌سازی هزینه مجموعه انتخابی λ است که تمام نقاط در U را پوشش دهد.

توضیحات:

مجموعه Ω : این مجموعه شامل تمام زیرمجموعه‌های ممکن از دوربین‌ها λ است که قادر به پوشش تمام نقاط در U هستند. شرط این است که اتحاد پوشش دوربین‌ها در λ باید تمام نقاط U را پوشش دهد.

تابع هزینه $c(\lambda)$: هزینه مجموعه دوربین‌ها برابر با مجموع هزینه‌های نصب دوربین‌های موجود در λ است. اگر هزینه‌ها یکسان باشند، تابع هدف فقط تعداد دوربین‌های انتخاب شده را کمینه می‌کند.

هدف :هدف ما کمینه‌سازی هزینه (که می‌تواند تعداد دوربین‌ها یا مجموع هزینه‌های نصب باشد) با رعایت محدودیت پوشش تمام نقاط U است.

توضیحات کد پایتون

1. جواب اولیه(Initial Solution)

هدف: تولید یک جواب اولیه به‌طور تصادفی که تمام نقاط پوشش داده شوند.

عملکرد: دوربین‌های تصادفی انتخاب می‌شوند تا همه نقاط پوشش داده شوند.

```
def generate_initial_solution(cameras, points):
    solution = []
    covered_points = set()
    while len(covered_points) < len(points):
            random_camera = random.choice(list(cameras.keys()))
        solution.append(random_camera)
        covered_points.update(cameras[random_camera])
    return solution
```

2. تابع ارزیاب(Evaluate Solution)

هدف: ارزیابی کیفیت یک جواب با بررسی اینکه آیا تمام نقاط پوشش داده شده‌اند یا نه و محاسبه تعداد دوربین‌های انتخاب‌شده.

عملکرد: بررسی پوشش نقاط توسط دوربین‌ها و محاسبه هزینه (تعداد دوربین‌ها).

```
def evaluate_solution(solution, cameras, points):
    covered_points = set()
    unique_cameras = set(solution)
    for camera in unique_cameras:
            covered_points.update(cameras[camera])
    uncovered_points = set(points) - covered_points
    return len(unique_cameras), len(uncovered_points)
```

3. همسایگی‌ها(Neighborhood)

هدف: تولید همسایگانی برای جواب جاری (با اضافه یا حذف دوربین).

عملکرد: برای هر دوربین در جواب فعلی، دوربین اضافه یا حذف می‌شود و بررسی می‌شود که آیا جواب جدید تمام نقاط را پوشش می‌دهد یا نه.

```
def generate_neighborhood(current_solution, cameras, points):
    neighborhood = []
    for camera in cameras.keys():
            neighbor = set(current_solution)
            if camera in neighbor:
                    neighbor.remove(camera)
            else:
                    neighbor.add(camera)
```

```
            if evaluate_solution(list(neighbor), cameras, points)[1] == 0: # اگر تمام نقاط پوشش داده شوند
                neighborhood.append(list(neighbor))
    return neighborhood
```

4. تابع تابوسرچ (Tabu_Search)

۱ .تعریف تابع

```
def tabu_search(cameras, points, max_iter=100, tabu_size=10, aspiration_criteria=True):
    """Perform the Tabu Search algorithm to optimize the camera selection for covering all points."""
```

این خط یک تابع به نام tabu_search تعریف می‌کند که ورودی‌های زیر را می‌گیرد:

- Cameras :مجموعه‌ای از دوربین‌ها و محدوده دیدشان.
- Points :نقاطی که باید پوشش داده شوند.
- max_iter :حداکثر تعداد تکرارهای الگوریتم (پیش‌فرض: ۱۰۰).
- tabu_size :اندازه لیست تابو برای محدود کردن بازدیدهای مجدد (پیش‌فرض: ۱۰).
- aspiration_criteria :یک معیار برای لغو محدودیت‌های لیست تابو در صورت بهبود هزینه (پیش‌فرض : True).

۲ .مقداردهی اولیه

```
current_solution = generate_initial_solution(cameras, points)
```

- این خط یک راه‌حل اولیه (مجموعه‌ای از دوربین‌های فعال) تولید می‌کند.

```
best_solution = current_solution[:]
```

- بهترین راه‌حل تا کنون با راه‌حل فعلی مقداردهی می‌شود.

```
best_cost, uncovered_points = evaluate_solution(best_solution, cameras, points)
```

- هزینه (مثل تعداد دوربین‌ها) و نقاط پوشش داده نشده در بهترین راه‌حل محاسبه می‌شود.

```
tabu_list = []
```

- لیست تابو (برای ذخیره راه‌حل‌های ممنوعه) به‌صورت خالی مقداردهی می‌شود.

```
aspiration_uses = 0
```

- شمارش تعداد دفعات استفاده از معیار اشتیاق مقداردهی می‌شود.

۳ .ایجاد فایل لاگ

```
output_file = "tabu_search_log.txt"
```

```
with open(output_file, 'w') as file:
  file.write("Iteration\tBest Cost\tTabu Violations\tAspiration Uses\n")
```

- یک فایل لاگ با نام tabu_search_log.txt ایجاد می‌شود و سرستون‌های آن نوشته می‌شوند.

۴ .حلقه اصلی الگوریتم

```
for iteration in range(1, max_iter + 1):
```

- حلقه‌ای که الگوریتم را برای حداکثر تعداد max_iter اجرا می‌کند.

۵ .ایجاد همسایگی‌ها

```
neighborhood = []
```

- لیستی برای نگهداری همسایگی‌های راه‌حل فعلی تعریف می‌شود.

```
for camera in cameras.keys():
  neighbor = set(current_solution)
  if camera in neighbor:
    neighbor.remove(camera)
  else:
    neighbor.add(camera)
```

- برای هر دوربین:
  - اگر در راه‌حل فعلی است، حذف می‌شود.
  - در غیر این صورت، اضافه می‌شود.

```
if evaluate_solution(list(neighbor), cameras, points)[1] == 0:
  neighborhood.append(list(neighbor))
```

- اگر همسایه تمام نقاط را پوشش دهد (یعنی نقاط پوشش داده نشده صفر شود)، به لیست همسایگی اضافه می‌شود.

۶ .انتخاب بهترین همسایه

```
best_neighbor = None
best_eval = float('inf')
tabu_violations = 0
```

- مقداردهی اولیه برای:
  - بهترین همسایه.
  - بهترین ارزیابی هزینه (با مقدار بی‌نهایت).
  - شمارش نقض‌های لیست تابو.

۷ .بررسی همسایگی‌ها

```
for neighbor in neighborhood:
  eval_solution = evaluate_solution(neighbor, cameras, points)
```

- برای هر همسایه، هزینه و تعداد نقاط پوشش داده نشده محاسبه می‌شود.

```
if neighbor not in tabu_list or (aspiration_criteria and eval_solution[0] < best_cost):
```

- همسایه انتخاب می‌شود اگر:
    - در لیست تابو نباشد.
    - یا معیار اشتیاق اجازه دهد (یعنی هزینه کمتر از بهترین هزینه باشد).

```
if eval_solution[0] < best_eval:
    best_eval = eval_solution[0]
    best_neighbor = neighbor
```

- اگر هزینه همسایه از بهترین ارزیابی فعلی بهتر باشد، بهترین همسایه به‌روزرسانی می‌شود.

۸ .به‌روزرسانی راه‌حل‌ها

```
if best_neighbor:
    current_solution = best_neighbor[:]
    current_cost, _ = evaluate_solution(current_solution, cameras, points)
```

- بهترین همسایه به‌عنوان راه‌حل فعلی تنظیم می‌شود و هزینه آن محاسبه می‌شود.

```
if current_cost < best_cost:
    best_solution = current_solution[:]
    best_cost = current_cost
else:
    tabu_violations += 1
```

- اگر راه‌حل فعلی بهتر از بهترین راه‌حل باشد:
    - بهترین راه‌حل و هزینه آن به‌روزرسانی می‌شود.
- در غیر این صورت، نقض تابو افزایش می‌یابد.

۹ .مدیریت لیست تابو

```
tabu_list.append(current_solution)
if len(tabu_list) > tabu_size:
    tabu_list.pop(0)
```

- راه‌حل فعلی به لیست تابو اضافه می‌شود.
- اگر لیست از اندازه مشخص‌شده بزرگ‌تر شود، قدیمی‌ترین راه‌حل حذف می‌شود.

۱۰ .ثبت اطلاعات در لاگ

```
with open(output_file, 'a') as file:
    file.write(f"{iteration}\t{best_cost}\t{tabu_violations}\t{aspiration_uses}\n")
```

- اطلاعات مربوط به تکرار فعلی، شامل هزینه بهترین راه‌حل، نقض‌های تابو، و استفاده از معیار اشتیاق، در فایل لاگ ذخیره می‌شود.

۱۱ .خروجی نهایی

```
return best_solution, best_cost, aspiration_uses
```

- بهترین راه‌حل، هزینه آن، و تعداد استفاده از معیار اشتیاق بازگردانده می‌شوند.

توضیحات $cplex$

```
    ایجاد مدل بهینه‌سازی#
    mdl = Model(name='Camera Placement Optimization')
    متغیرهای تصمیم (دوربین‌ها) #
    x = {camera: mdl.binary_var(name=f'x_{camera}') for camera in cameras}
    تابع هدف: کمینه کردن تعداد دوربین‌ها #
    mdl.minimize(mdl.sum(x[camera] for camera in cameras))
    محدودیت‌ها: پوشش دادن هر نقطه #
    for point in points:
    coverage_cameras = [camera for camera in cameras if point in
cameras[camera]['coverage']]
    mdl.add_constraint(mdl.sum(x[camera] for camera in coverage_cameras) >= 1)
    حل مدل #
    solution = mdl.solve()
```

هدف: استفاده از کتابخانه $docplex$ برای حل مسأله بهینه‌سازی که هدف آن کمینه کردن تعداد دوربین‌های انتخاب‌شده برای پوشش همه نقاط است .عملکرد :تعریف متغیرهای تصمیم: هر دوربین یک متغیر دودویی (0 یا 1) دارد که نشان می‌دهد آیا دوربین انتخاب می‌شود یا نه .تابع هدف: تعداد دوربین‌های انتخاب‌شده باید کمینه شود .محدودیت‌ها: هر نقطه باید حداقل توسط یک دوربین پوشش داده شود .شبه کد:

تحلیل جواب‌ها و نمودارها

نمونه اول: 12 نقطه – 6 دوربین

نمونه دوم: 200 نقطه – 100 دوربین

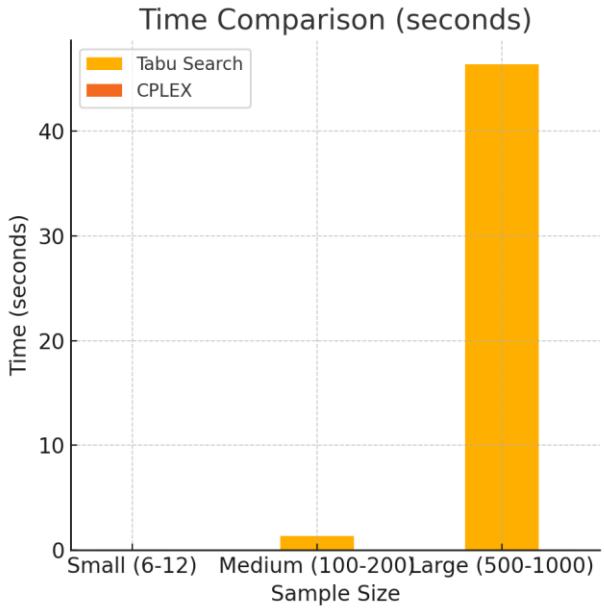
نمونه سوم: 1000 نقطه – 500 دوربین

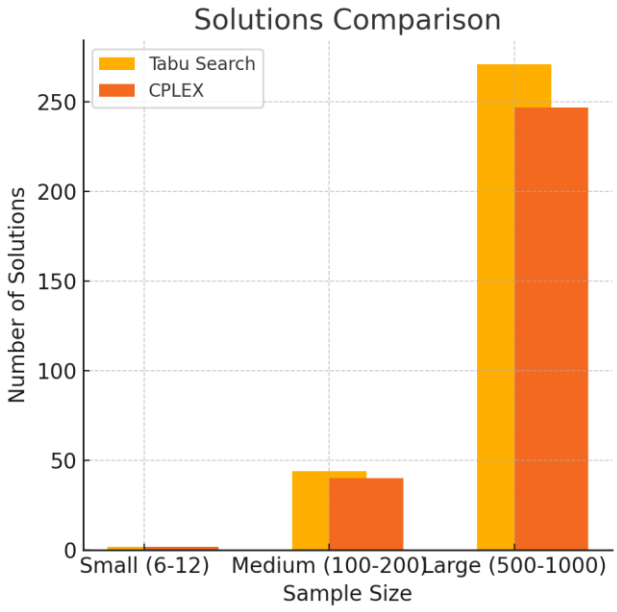
نام روش	نمونه کوچک (6–12)	نمونه متوسط (100–200)	نمونه بزرگ (500–1000)
TABU SEARCH	0.09seconds	1.35seconds	46.38seconds
CPLEX	0.0005 seconds	0.01 seconds	0.0425 seconds

نمودار1- زمان اجرا

نام روش	نمونه کوچک (6–12)	نمونه متوسط (100–200)	نمونه بزرگ (500–1000)
TABU SEARCH	2	44	271
CPLEX	2	40	247

نمودار2- جواب الگوریتم‌ها





منابع:

[1]3D Optimal Surveillance Trajectory Planning for Multiple UAVs by Using Particle Swarm Optimization With Surveillance Area Priority; HU TENG, ISHTIAQ AHMAD, ALAMGIR MSM

[2]An Intelligent Traffic Surveillance System Using Integrated Wireless Sensor Network and Improved Phase Timing Optimization; Quadri Noorulhasan Naveed, Hamed Alqahtani

[3]Optimisation of surveillance camera site locations and viewing angles using a novel multi-attribute, multi-objective genetic algorithm: A day/night anti-poaching application; Andries M. Heyns

[4]Computational Intelligence-Based Harmony Search Algorithm for Real-Time Object Detection and Tracking in Video Surveillance Systems; Maged Faihan Alotaibi