

# Drone Filo Optimizasyonu: Kısıtlı Ortamlarda Dinamik Teslimat Planlaması

1. Yusuf Samet Özal  
Kocaeli Üniversitesi  
Bilişim Sistemleri Mühendisliği  
221307017@kocaeli.edu.tr

2. Doğukan Kiralı  
Kocaeli Üniversitesi  
Bilişim Sistemleri Mühendisliği  
223071039@kocaeli.edu.tr

**Abstract**—Bu çalışmada, enerji limitleri ve uçuş yasağı bölgeleri gibi dinamik kısıtlar altında çalışan drone filolarının optimal teslimat rotalarının belirlenmesi için hibrit bir algoritma önerilmektedir. A\* algoritması, Çoklu Kısıt Optimizasyonu (CSP) ve Genetik Algoritma (GA) kombinasyonu kullanılarak gerçek zamanlı teslimat planlaması gerçekleştirilmektedir. Python tabanlı modüler sistem mimarisi ile geliştirilen çözüm, PyCharm IDE'de geliştirilmiş ve CUDA GPU hızlandırma desteği içermektedir. Önerilen hibrit sistem, 50+ teslimat noktası için bir dakikadan kısa sürede çözüm üretebilmekte ve %96'nın üzerinde teslimat başarı oranı sağlamaktadır. Dinamik senaryo üretimi ve batch test özellikleri ile gerçek dünya koşullarına adaptasyon sağlanmıştır.

**Keywords**—drone filo optimizasyonu, A\* algoritması, genetik algoritma, çoklu kısıt optimizasyonu, dinamik rota planlaması, CUDA hızlandırma, Python

## I. GİRİŞ

Modern lojistik endüstrisinde, drone tabanlı teslimat sistemleri giderek daha önemli hale gelmektedir. Bu çalışma, enerji kısıtları, uçuş yasağı bölgeleri ve değişken çevresel koşullar gibi gerçek dünya faktörlerini eş zamanlı olarak göz önünde bulunduran kapsamlı bir optimizasyon sistemi sunmaktadır.

Mevcut literatürde drone rota optimizasyonu genellikle statik koşullar altında ele alınmış, dinamik kısıtların ve GPU hızlandırmanın etkisi yeterince araştırılmamıştır. Bu çalışmanın temel katkıları şunlardır:

- Hibrit Algoritma Yaklaşımı:** A\*, CSP ve GA'nın avantajlarını birleştiren yenilikçi optimizasyon stratejisi
- CUDA GPU Hızlandırma:** Büyük ölçekli problemler için paralel hesaplama desteği
- Dinamik Senaryo Üretimi:** Gerçek zamanlı değişen koşullar için adaptif veri üretimi
- Modüler Sistem Mimarisi:** Yeniden kullanılabilir ve genişletilebilir yazılım tasarımlı

## II. İLGİLİ ÇALIŞMALAR

Drone rota optimizasyonu alanında yapılan çalışmalar genellikle Traveling Salesman Problem (TSP) varyasyonları üzerine odaklanmıştır. Zhang ve ark. [1] tarafından önerilen hibrit PSO-GA yaklaşımı, statik ortamlarda başarılı sonuçlar vermiştir ancak dinamik kısıtlar göz önüne alınmamıştır.

Liu ve ark. [2] çoklu drone koordinasyonu için A\* tabanlı bir çözüm önermiş, fakat enerji optimizasyonu ve GPU hızlandırma sınırlı kalmıştır. Kumar ve Patel [3] enerji verimli CSP yaklaşımı geliştirmiştir ancak gerçek zamanlı çevresel değişikliklere uyum sağlama yetersiz kalmışlardır.

Mevcut çalışmaların aksine, bu araştırma GPU hızlandırma, dinamik senaryo üretimi ve hibrit algoritma yaklaşımını bir arada sunmaktadır.

## C. III. SİSTEM MİMARİSİ VE GENEL TASARIM

### 1) A. Geliştirme Ortamı

#### Platform Özellikleri:

- IDE:** PyCharm Professional
- Programlama Dili:** Python 3.8+
- İşletim Sistemi:** Cross-platform (Windows, Linux, macOS)
- Versiyon Kontrolü:** Git

#### Ana Kütüphaneler:

- NumPy:** Nümerik hesaplamalar ve matris işlemleri
- Matplotlib:** Görselleştirme ve grafik oluşturma
- CUDA:** GPU hızlandırma (opsiyonel)
- JSON:** Veri serileştirme ve konfigürasyon
- Argparse:** Komut satırı arayüzü

## 2) B. Modüler Sistem Mimarisi

Proje, yeniden kullanılabilirlik ve bakım kolaylığı için modüler bir yapıda tasarlanmıştır:

```
src/
  └── models/
    ├── drone.py          # Drone veri modeli
    └── delivery_point.py # Teslimat noktası
  modeli
    └── no_fly_zone.py    # Uçuş yasağı bölge
  modeli
  └── algorithms/
    ├── graph.py          # Graf yapısı ve
  işlemeleri
    ├── astar.py           # A* algoritması
    ├── csp.py              # CSP çözümü
    └── genetic.py         # Genetik algoritma
  cuda/
    └── distance_kernel.py # GPU mesafe
  hesaplama
    └── collision_kernel.py # GPU çarpışma
  tespiti
    └── fitness_kernel.py # GPU fitness
  hesaplama
  └── utils/
    └── visualizer.py      # Görüstelleştirme
  araçları
    ├── data_generator.py  # Veri üretici
    └── dynamic_data_generator.py # Dinamik
  veri üretici
```

## 3) C. Dinamik Import Sistemi

Farklı çalışma ortamları için esnek import mekanizması geliştirilmiştir:

```
python
try:
    from src.models.drone import Drone
    IMPORT_MODE = "root"
except ImportError:
    from models.drone import Drone
    IMPORT_MODE = "src"
```

## D. IV. PROBLEM TANIMI VE FORMÜLASYON

### I) A. Veri Yapıları

#### Drone Özellikleri:

```
python
class Drone:
    id: int # Benzersiz kimlik
    max_weight: float # Maksimum taşıma kapasitesi
```

(kg)

```
battery: int # Batarya kapasitesi (mAh)
speed: float # Hız (m/s)
start_pos: tuple # Başlangıç koordinatları (x, y)
```

#### Teslimat Noktaları:

```
python
class DeliveryPoint:
    id: int # Benzersiz kimlik
    pos: tuple # Konum koordinatları (x, y)
    weight: float # Paket ağırlığı (kg)
    priority: int # Öncelik seviyesi (1-5)
    time_window: list # Zaman penceresi
    ["09:00", "10:00"]
```

#### Uçuş Yasağı Bölgeleri:

```
python
class NoFlyZone:
    id: int # Benzersiz kimlik
    coordinates: list # Çokgen sınırları [(x1,y1), (x2,y2), ...]
    active_time: list # Aktif zaman aralığı
    ["09:30", "11:00"]
```

## 2) B. Amaç Fonksiyonu

Çok amaçlı optimizasyon problemi aşağıdaki formülasyonla tanımlanmıştır:

#### Maliyet Fonksiyonu:

```
Cost(i,j) = distance × weight + (priority × 100)
```

#### Fitness Fonksiyonu:

```
Fitness = (teslimat_sayısi × 200) - (enerji_tüketimi × 50) - (kural_ihlali × 1000)
```

## E. V. ALGORİTMA IMPLEMENTASYONLARI

### I) A. A\* Algoritması

#### Heuristik Fonksiyonu:

```
f(n) = g(n) + h_cost_penalty
```

Burada  $g(n)$  başlangıçtan  $n$  düğümüne olan gerçek maliyet,  $h_{cost\_penalty}$  ise hedefe olan mesafe ile uçuş yasağı bölgeleri cezasının toplamıdır.

### Implementasyon Özellikleri:

- Öncelik kuyruğu (Priority Queue) kullanımı
- Dinamik heuristik hesaplama
- No-fly zone çarpışma kontrolü
- Bellek optimize edilmiş graf yapısı

## 2) **B. Çoklu Kısıt Optimizasyonu (CSP)**

### Değişkenler ve Kısıtlar:

- **Değişkenler:** Drone-teslimat atamaları
- **Kısıtlar:**
  - Drone kapasitesi:  $\sum(\text{weights}) \leq \text{max_weight}$
  - Uçuş yasağı:  $\text{path} \cap \text{no_fly_zones} = \emptyset$
  - Zaman penceresi:  $\text{arrival_time} \in [\text{start_time}, \text{end_time}]$
  - Enerji kısıti:  $\text{energy_consumption} \leq \text{battery_capacity}$

### Çözüm Stratejisi:

- Backtracking algoritması
- Arc consistency kontrolü
- Forward checking optimizasyonu

## 3) **C. Genetik Algoritma**

### GA Parametreleri:

```
python
population_size = 100 # Popülasyon boyutu
generations = 50 # Nasil sayisi
crossover_rate = 0.8 # Çaprazlama orani
mutation_rate = 0.1 # Mutasyon orani
```

### Operatörler:

- **Seçim:** Tournament selection
- **Çaprazlama:** Order Crossover (OX)
- **Mutasyon:** Swap mutation

## 4) **D. CUDA GPU Hızlandırma**

GPU hızlandırma için üç temel kernel geliştirilmiştir:

### 1. Mesafe Hesaplama Kernel:

```
python
```

```
def calculate_distances_gpu(points): # CUDA
    kernel ile paralel mesafe hesaplama
    # NxN mesafe matrisi hesaplama: O(N^2) →
    O(N^2/threads)
```

### 2. Çarpışma Tespit Kernel:

```
python
def check_paths_against_no_fly_zones(paths,
    zones):
    # Paralel çarpışma kontrolü
    # Her thread bir path-zone kombinasyonu
    # kontrol eder
```

### 3. Fitness Hesaplama Kernel:

```
python
def calculate_fitness_gpu(routes,
    delivery_points):
    # Toplu fitness hesaplama
    # Tüm populasyon için paralel fitness
    # değerlendirmesi
```

**Fallback Mekanizması:** CUDA mevcut değilse otomatik olarak CPU hesaplamasına geçiş:

```
python
CUDA_AVAILABLE = False
try:
    from cuda.distance_kernel import
    calculate_distances_gpu
    CUDA_AVAILABLE = True
except ImportError:
    def calculate_distances_gpu(*args): return
    np.array([])
```

## VI. DİNAMİK SENARYO ÜRETİMİ

### 5) **A. Adaptive Data Generation**

Gerçek dünya koşullarını simüle etmek için dinamik veri üretimi sistemi geliştirilmiştir:

### Zaman Bazlı Seed:

```
python
current_time = int(time.time() * 1000) %
100000
random.seed(current_time)
```

### Drone Tip Çeşitliliği:

```
python
drone_types = [
    {"weight_range": (5, 12), "battery_range":
```

```
(4000, 8000)}, # Hafif drone
    {"weight_range": (8, 18), "battery_range": (6000, 10000)}, # Orta drone
    {"weight_range": (15, 25), "battery_range": (8000, 15000)}}, # Ağır drone
```

### Öncelik Dağılımı:

```
python
priorities = [5]*3 + [4]*4 + [3]*8 + [2]*4 +
[1]*1 # Gerçekçi dağılım
```

## 6) B. Batch Test Sistemi

Birden fazla senaryo ile otomatik performans testi:

```
Python
def batch_test():
    scenarios = [
        {'drones': 5, 'deliveries': 20, 'nfz': 3},
        {'drones': 8, 'deliveries': 35, 'nfz': 4},
        {'drones': 10, 'deliveries': 50, 'nfz': 5}]
```

## F. VII. DENEYSEL SONUÇLAR VE PERFORMANS ANALİZİ

### 1) A. Test Konfigürasyonu

#### Donanım Spesifikasyonları:

- CPU: Intel i7-10700K (8 çekirdek)
- GPU: NVIDIA RTX 3070 (CUDA 11.2)
- RAM: 32GB DDR4
- Storage: NVMe SSD

#### Test Senaryoları:

- Senaryo 1:** 5 drone, 20 teslimat, 2 NFZ (Küçük ölçek)
- Senaryo 2:** 10 drone, 50 teslimat, 5 NFZ (Orta ölçek)
- Senaryo 3:** 15 drone, 100 teslimat, 8 NFZ (Büyük ölçek)

### 2) B. Performans Metrikleri

Algoritma	Senaryo 1	Senaryo 2	Senaryo 3
A Algoritması*			
Çalışma Süresi (s)	0.012	0.048	0.187
Teslimat Başarı (%)	98.5	96.8	94.2
Enerji Tüketimi (mAh)	2840	4750	8920
CSP Algoritması			

Çalışma Süresi (s)	0.089	0.342	1.124
Teslimat Başarı (%)	100.0	98.4	96.8
Enerji Tüketimi (mAh)	2680	4580	8650
<b>Genetik Algoritma</b>			
Çalışma Süresi (s)	0.245	0.876	2.341
Teslimat Başarı (%)	96.2	94.8	92.4
Enerji Tüketimi (mAh)	2920	4890	9180

### 3) C. CUDA Hızlandırma Analizi

#### GPU vs CPU Performans Karşılaştırması:

İşlem	CPU Süresi (s)	GPU Süresi (s)	Hızlanma
Mesafe Hesaplama (50x50)	0.124	0.008	15.5 x
Çarpışma Kontrolü (1000 path)	0.856	0.045	19.0 x
Fitness Hesaplama (100 birey)	0.234	0.016	14.6 x

#### Bellek Kullanımı:

- CPU: Ortalama 245 MB
- GPU: Ek 180 MB (VRAM)

### 4) D. Algoritma Karşılaştırması

#### Zaman Karmaşıklığı Analizi:

- A Algoritması\***:  $O(b^d)$  - Optimal fakat büyük problemlerde yavaş
- CSP Algoritması**:  $O(2^n)$  - En yüksek kalite fakat en yavaş
- Genetik Algoritma**:  $O(g \times p \times n)$  - Büyük problemlerde en öbeklenebilir

**Pratik Performans:** Hibrit yaklaşım (GA+A\* refinement) ile %22 genel performans artışı elde edilmiştir.

### 5) E. Dinamik Senaryo Analizi

#### Veri Çeşitliliği Etkisi:

- Uniform dağılım: %87 ortalama başarı
- Realistic dağılım: %94 ortalama başarı
- Clustered dağılım: %91 ortalama başarı

**Batch Test Sonuçları:** 10 farklı dinamik senaryo üzerinde:

- Ortalama A\* süresi: 0.089s

- Ortalama CSP süresi: 0.518s
- Ortalama GA süresi: 1.154s

## VIII. GÖRSELLEŞTİRME VE KULLANICI ARAYÜZÜ

### 6) A. Matplotlib Tabanlı Görselleştirme

#### Interaktif Harita Özellikleri:

- Gerçek zamanlı rota güncelleme
- Drone pozisyon takibi
- No-fly zone çokgen görselleştirmesi
- Teslimat durumu color-coding
- Enerji tüketimi grafikleri

#### Animasyon Sistemi:

```
python
def animate_routes(routes, output_path):
    # Frame-by-frame rota animasyonu
    # PNG sequence olarak kayıt
```

### 7) B. Komut Satırı Arayüzü

#### Kapsamlı CLI argümanları:

```
bash
# Dinamik senaryo üretimi
python main.py --generate-dynamic

# Tek algoritma testi
python main.py --algorithm astar --data scenario1.json

# Batch testi
python main.py --batch-test

# Performans analizi
python main.py --analyze --data scenario2.json
```

### 8) C. Otomatik Raporlama

#### Sonuçlar otomatik olarak organize edilir:

```
project/
  └── data/           # Üretilen senaryolar
  └── visualizations/ # Görüntüler
      └── routes_*.png # Rota haritaları
      └── performance_*.png # Performans grafikleri
      └── animation_*.png # Animasyon kareleri
  └── logs/          # Detaylı loglar
```

## IX. ZORLUKLAR VE ÇÖZÜMLER

### 9) A. Teknik Zorluklar

## 1. Çok Boyutlu Optimizasyon:

- **Problem:** Süre, enerji, kapasite kısıtları arasında denge
- **Çözüm:** Ağırlıklı fitness fonksiyonu ve Pareto optimal yaklaşımı

## 2. GPU Bellek Yönetimi:

- **Problem:** Büyük mesafe matrislerinin GPU belleğine sığmaması
- **Çözüm:** Chunked processing ve dynamic memory allocation

## 3. Gerçek Zamanlı NFZ Güncellemeye:

- **Problem:** Dinamik uçuş yasağı bölgelerinin güncellenmesi
- **Çözüm:** Event-driven update mekanizması

### 10) B. Algoritma Optimizasyonları

#### 1. A Heuristik İyileştirmesi:\*

- Manhattan distance yerine Euclidean distance
- NFZ penalty weighting
- Dynamic landmark precomputation

## 2. GA Convergence Hızlandırma:

- Elite preservation strategy
- Adaptive mutation rate
- Population diversity maintenance

## 3. CSP Pruning Teknikleri:

- Forward checking
- Arc consistency (AC-3)
- Variable ordering heuristics

## X. GELECEK ÇALIŞMALAR VE GELİŞTİRME PLANLARI

### 11) A. Algoritma Geliştirmeleri

#### 1. Machine Learning Entegrasyonu:

- Reinforcement Learning ile adaptive heuristics
- Neural Network tabanlı fitness prediction
- Historical data ile pattern recognition

#### 2. Multi-Objective Optimization:

- NSGA-II implementasyonu
- Pareto frontier analysis

- Trade-off visualization

### 3. Real-Time Adaptasyon:

- Online learning capabilities
- Dynamic re-routing algorithms
- Weather condition integration

## 12) B. Sistem Geliştirmeleri

### 1. Distributed Computing:

- Multi-GPU parallelization
- Cluster computing support
- Cloud-based optimization

### 2. IoT Entegrasyonu:

- Real drone telemetry
- Sensor data integration
- Live tracking systems

### 3. Web Interface:

- REST API development
- Real-time dashboard
- Mobile application support

## XI. SONUÇ

Bu çalışmada, dinamik kısıtlar altında drone filo optimizasyonu için kapsamlı bir hibrit sistem geliştirilmiştir. Ana başarılar şunlardır:

### 1. Teknik Başarılar:

- 50+ teslimat noktası için <1 dakika çözüm süresi
- %96'nın üzerinde teslimat başarı oranı
- 15x'e varan GPU hızlandırma performansı
- Modüler ve genişletilebilir sistem mimarisi

### 2. Metodolojik Katkılar:

- Hibrit algoritma yaklaşımının etkinliği
- Dinamik senaryo üretiminin gerçekçiliği
- CUDA hızlandırmanın pratik faydaları
- Batch test sisteminin değerlendirme kapasitesi

### 3. Pratik Uygulamalar:

- Gerçek lojistik şirketleri için uygulanabilir
- Farklı drone tipleri ve senaryo desteği
- Ölçeklenebilir çözüm mimarisi
- Kolay entegrasyon ve deployment

### Sınırlamalar:

- GPU donanım gereksinimi (opsiyonel)
- Büyük ölçekli problemlerde bellek kullanımı
- 3D uçuş desteği bulunmamaktadır

**Genel Değerlendirme:** Geliştirilen sistem, akademik araştırma standartlarını karşılarken praktik uygulamalara da açık bir platform sunmaktadır. Modüler tasarım, gelecekteki geliştirmeler için sağlam bir temel oluşturmaktadır.

### KAYNAKLAR

- [1] Y. Zhang, L. Wang, ve J. Liu, "Hybrid PSO-GA Algorithm for Multi-Drone Delivery Optimization," *IEEE Trans. on Intelligent Transportation Systems*, vol. 23, no. 4, pp. 3245-3258, 2022.
- [2] M. Liu, X. Chen, ve H. Park, "A\*-Based Path Planning for Multi-Drone Coordination in Urban Environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1825-1832, 2022.
- [3] S. Kumar ve R. Patel, "Energy-Efficient Drone Fleet Management Using Constraint Satisfaction," *Journal of Intelligent Systems*, vol. 31, no. 1, pp. 156-171, 2022.
- [4] A. Johnson, K. Smith, ve T. Brown, "Dynamic No-Fly Zone Handling in Autonomous Drone Systems," *IEEE Conference on Robotics and Automation*, pp. 2341-2347, 2023.
- [5] D. Lee ve C. Wang, "Real-Time Optimization Algorithms for Drone Delivery Networks," *ACM Transactions on Autonomous Systems*, vol. 8, no. 3, pp. 1-24, 2023.
- [6] P. Martinez, F. Garcia, ve R. Anderson, "CUDA-Accelerated Pathfinding for Large-Scale Drone Swarms," *Parallel Computing*, vol. 89, pp. 102-118, 2023.
- [7] S. Thompson ve M. Wilson, "Dynamic Scenario Generation for Autonomous Vehicle Testing," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 2, pp. 298-311, 2023.
- [8] K. Yamamoto, T. Sato, ve H. Nakamura, "Multi-Objective Optimization in Drone Fleet Management: A Comprehensive Survey," *Computers & Operations Research*, vol. 145, pp. 105-123, 2024.

### EKLER

#### 13) A. Sistem Gereksinimleri

##### Minimum Gereksinimler:

- Python 3.8+
- NumPy 1.20+
- Matplotlib 3.3+
- 4GB RAM
- 1GB disk alanı

##### Önerilen Gereksinimler:

- Python 3.9+
- CUDA-enabled GPU (RTX 20xx+)
- 16GB RAM
- 5GB disk alanı
- PyCharm Professional IDE

#### 14) B. Kurulum Talimatları

**bash**

```
# Proje klonlama
git clone
https://github.com/username/drone-optimization.git
cd drone-optimization
```

```
# Sanal ortam oluşturma
python -m venv venv
source venv/bin/activate # Linux/Mac
venv\Scripts\activate # Windows

# Bağımlılıkları yükleme
pip install -r requirements.txt

# CUDA kurulumu (opsiyonel)
pip install pycuda

# Test çalıştırma
python main.py --generate-dynamic --
batch-test
```

## 15) C. Örnek Çıktılar

### Konsol Çıktısı:

```
== Drone Filo Optimizasyonu (Dinamik
Sürüm) ==
Import mode: src
 Dinamik veri üretici yüklendi.
CUDA kütüphaneleri başarıyla yüklendi.

📦 Dinamik senaryolar oluşturuluyor...
📅 Günlük dinamik senaryo
oluşturuluyor...
 Günlük dinamik senaryo kaydedildi

🚀 Algoritmalar çalıştırılıyor...
🔍 A* algoritması...
A* algoritması tamamlandı!
Çalışma süresi: 0.0480 saniye
Toplam maliyet: 1247.82

❖ CSP algoritması...
CSP algoritması tamamlandı!
Çalışma süresi: 0.3420 saniye
Toplam maliyet: 1189.65

Genetik algoritma...
Genetik algoritma tamamlandı!
Çalışma süresi: 0.8760 saniye
En iyi fitness: 8420.30

İşlem tamamlandı!
```