Bilgisayar Oyunlarında Yapay Zekâ Ödev-1 Raporu

Metehan SÖZENLİ 25435004037

Ödev-1 kapsamında Gezgin Satıcı Problemi (TSP), Nearest Neighbor (En Yakın Komşu) heuristiği kullanılarak çözülmüştür. Nearest Neighbor yöntemi, TSP'nin çözümünde kullanılan basit ve hızlı bir yaklaşımdır. Bu yöntemde satıcı, rastgele seçilen bir başlangıç noktasından yola çıkarak her adımda henüz ziyaret edilmemiş şehirler arasından kendisine en yakın olanı seçer ve bu şekilde tüm şehirleri dolaştıktan sonra başlangıç noktasına geri döner. Optimum çözümü garanti etmese de hesaplama açısından oldukça verimli olan bu algoritma, özellikle şehir sayısının fazla olduğu durumlarda kısa sürede yaklaşık bir çözüm elde edilmesini sağlar.

Kod Açıklamaları

create_random_tsp_graph() fonksiyonu, verilen nokta sayısına göre iki boyutlu düzlemde rastgele şehirler oluşturur ve bu şehirler arasında Öklid mesafesine dayalı kenarlar ekleyerek tam bir graf yapısı oluşturur.

```python
def create_random_tsp_graph(n: int, seed: int, area=(0.0, 100.0)):
    """Random noktalardan TSP graf'ı oluşturur"""
    random.seed(seed)
    G = nx.Graph()

    # Random noktalar oluştur
    for i in range(n):
        x = random.uniform(*area)
        y = random.uniform(*area)
        G.add_node(i, x=x, y=y)

    # Tam graf (complete graph) kenarları ekle - Euclidean mesafelerle
    for i in range(n):
        for j in range(i+1, n):
            xi, yi = G.nodes[i]['x'], G.nodes[i]['y']
            xj, yj = G.nodes[j]['x'], G.nodes[j]['y']
            distance = float(np.hypot(xi - xj, yi - yj))
            G.add_edge(i, j, length=distance)

    return G, list(range(n))
```

NearestNeighborSolver sınıfı, Gezgin Satıcı Problemi'nin çözümünde kullanılan Nearest Neighbor heuristiğini uygular. Bu sınıfın solve() metodu, başlangıç noktasından başlayarak her adımda henüz ziyaret edilmemiş şehirler arasından en yakında olanı seçer ve tüm şehirler gezildikten sonra başlangıç noktasına dönerek turun sırasını döndürür.

```python
class NearestNeighborSolver:
    def solve(self, G: Any, start_idx: int = 0) -> List[int]:
        """TSP Solver"""
        nodes = list(G.nodes())
        unvisited = set(nodes)
        tour = [start_idx]
        unvisited.remove(start_idx)

        while unvisited:
            last = tour[-1]
            # Graf edge'lerinden direkt mesafe al
            nxt = min(unvisited, key=lambda j: G[last][j]['length'])
            tour.append(nxt)
            unvisited.remove(nxt)

        tour.append(start_idx)
        return tour
```

tour_length() fonksiyonu, elde edilen turun toplam uzunluğunu hesaplar.

```python
def tour_length(G: Any, tour: List[int]) -> float:
    """Graf edge'lerinden direkt tour uzunluğu hesapla"""
    return float(sum(G[tour[i]][tour[i+1]]['length'] for i in range(len(tour)-1)))
```

MatplotlibPlotter sınıfı ise sonuçların görselleştirilmesinden sorumludur. Bu sınıfın render() metodu, tüm şehirleri, bağlantıları ve elde edilen turu farklı renklerle çizerek anlaşılır bir grafik oluşturur ve çıktıyı dosya olarak kaydeder.

```python
class MatplotlibPlotter:
    def __init__(self):
        # Modern ve güzel renkler
        self.colors = {
            'background': '#f8f9fa',
            'grid': '#e9ecef',
            'nodes': '#4a90e2',
            'start_node': '#e74c3c',
            'tour_edge': '#e74c3c',
            'all_edges': '#bdc3c7',
            'text': '#2c3e50',
            'title': '#34495e'
        }

    def render(self, G: Any, nodes: Sequence[int], tour_idx: Sequence[int], out_path: str = "plot.png", total_length: float = 0):
        """Graf görselleştirmesi"""
        pos = {u: (G.nodes[u]['x'], G.nodes[u]['y']) for u in nodes}

        fig, ax = plt.subplots(figsize=(12, 10))
        fig.patch.set_facecolor(self.colors['background'])
        ax.set_facecolor(self.colors['background'])

        # Grid ekle
        ax.grid(True, color=self.colors['grid'], linestyle='-', linewidth=0.5, alpha=0.7)
        ax.set_axisbelow(True)

        # Tüm kenarları çiz
        all_edges = list(G.edges())
        nx.draw_networkx_edges(G, pos, edgelist=all_edges,
                               edge_color=self.colors['all_edges'],
                               width=0.8, alpha=0.3, ax=ax)

        # Tour kenarlarını çiz
        tour_edges = [(nodes[tour_idx[i]], nodes[tour_idx[i+1]]) for i in range(len(tour_idx)-1)]
        nx.draw_networkx_edges(G, pos, edgelist=tour_edges,
```

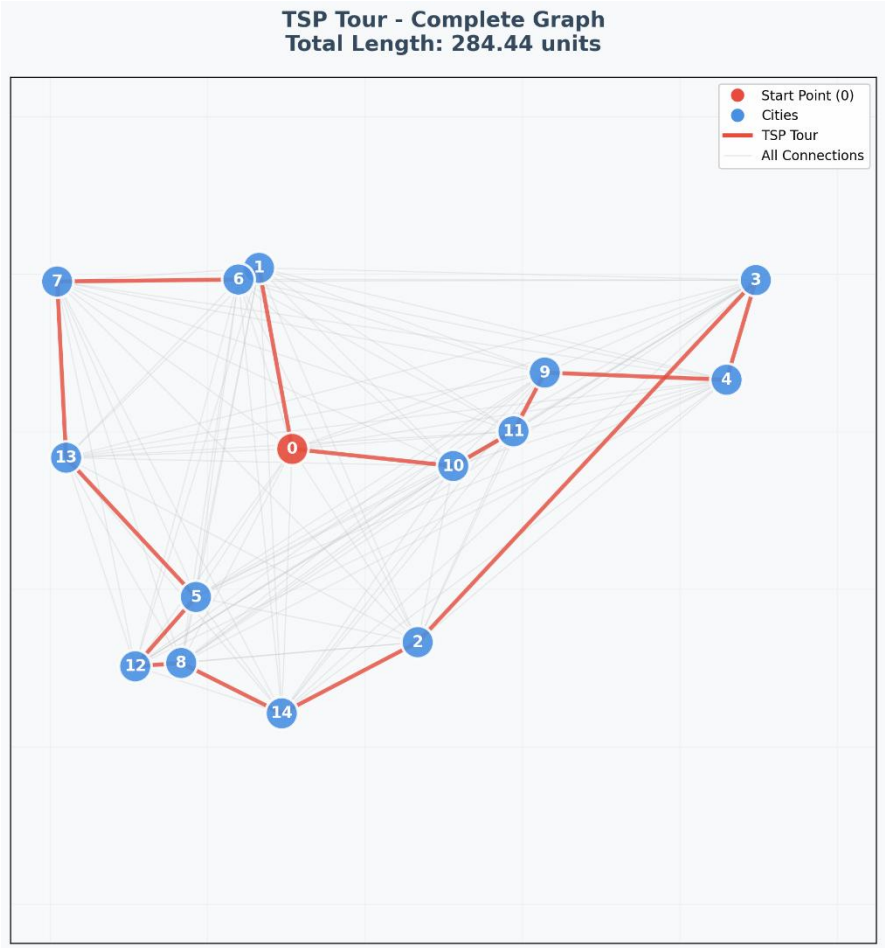
```python
    # Noktaları çiz
    node_colors = [self.colors['start_node'] if i == 0 else self.colors['nodes'] for i in range(len(nodes))]
    nx.draw_networkx_nodes(G, pos, nodelist=nodes,
                           node_color=node_colors,
                           node_size=600,
                           alpha=0.9, ax=ax,
                           edgecolors='white', linewidths=2)

    # Numara etiketleri
    labels = {nodes[i]: str(i) for i in range(len(nodes))}
    nx.draw_networkx_labels(G, pos, labels=labels,
                            font_size=12, font_weight='bold',
                            font_color='white', ax=ax)

    # Başlık ve bilgiler
    ax.set_title(f'TSP Tour - Complete Graph\nTotal Length: {total_length:.2f} units',
                 fontsize=16, fontweight='bold', color=self.colors['title'], pad=20)

    # Eksenleri eşitle ve sınırları ayarla
    ax.set_aspect('equal')
    margin = 5
    ax.set_xlim(-margin, 105)
    ax.set_ylim(-margin, 105)

    # Legend ekle
    legend_elements = [
        plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=self.colors['start_node'],
                   markersize=12, label='Start Point (0)', markeredgecolor='white', markeredgewidth=2),
        plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=self.colors['nodes'],
                   markersize=12, label='Cities', markeredgecolor='white', markeredgewidth=2),
        plt.Line2D([0], [0], color=self.colors['tour_edge'], linewidth=3, label='TSP Tour'),
        plt.Line2D([0], [0], color=self.colors['all_edges'], linewidth=1, alpha=0.3, label='All Connections')
    ]
    ax.legend(handles=legend_elements, loc='upper right', framealpha=0.9)

    plt.tight_layout()
    fig.savefig(out_path, dpi=200, bbox_inches='tight', facecolor=self.colors['background'])
    plt.close(fig)
    return out_path
```

Programın yürütüldüğü ***main()*** fonksiyonu, bu bileşenleri bir araya getirerek rastgele bir TSP grafı oluşturarak Nearest Neighbor algoritmasıyla çözüm üretir. Sonuç olarak da turun uzunluğunu hesaplayarak ve sonucu görselleştirir.

```python
def main():
    parser = argparse.ArgumentParser(description="Task 1: Random Points TSP with Nearest Neighbor")
    parser.add_argument("--n", type=int, default=15, help="Number of random points")
    parser.add_argument("--seed", type=int, default=42, help="Random seed")
    parser.add_argument("--out", type=str, default="plot_task1.png", help="Output plot filename")
    args = parser.parse_args()

    cfg = TaskConfig(n=args.n, seed=args.seed)

    # Rastgele TSP graf'ı oluştur
    G, nodes = create_random_tsp_graph(cfg.n, cfg.seed, area=(0.0, 100.0))

    # Nearest Neighbor algoritması ile çöz
    solver = NearestNeighborSolver()
    tour_idx = solver.solve(G, start_idx=0)
    total = tour_length(G, tour_idx)

    print(f"Tour (indices): {tour_idx}")
    print(f"Tour length ≈ {total:.2f} units")

    # Visualize
    plotter = MatplotlibPlotter()
    out_full = plotter.render(G, nodes, tour_idx, out_path=args.out, total_length=total)
    print(f"Saved plot: {out_full}")


if __name__ == "__main__":
    main()
```

Program çalıştırıldığında, belirtilen parametrelere göre (örneğin 15 şehir ve seed = 60) rastgele konumlara sahip noktalar üretilmiş ve bu noktalar arasında Nearest Neighbor heuristiği ile bir tur oluşturulmuştur. Çözüm sonucunda, satıcı 0. şehirden başlayarak her adımda en yakın komşuya geçip tüm şehirleri dolaşmış ve 284.44 birim yol kat ederek başlangıç noktasına geri dönmüştür.

```
PS D:\Yüksek Lisans Dersler\Bilgisayar Oyunlarında Yapay Zeka\Assignments> python task1.py --n 15 --seed 60
Tour (indices): [0, 10, 11, 9, 4, 3, 2, 14, 8, 12, 5, 13, 7, 6, 1, 0]
Tour length ≈ 284.44 units
Saved plot: plot_task1.png
```



Github Linki: https://github.com/BLM5026-AI-in-Computer-Games/hw1-metehansozenli