DAFTAR PUSTAKA

Ayu, F., Sitanggang, A., & Napitupulu, N. (2023). Implementation of Branch and Bound Algorithm to Solve the Travelling Salesman Problem at PT Jasa Harapan Barat. *Journal of Mathematics Education and Application (JMEA)*, *2*(3), 152–158.

Carlisle, A., & Dozier, G. (2001). An Off-The-Shelf PSO. *Proceedings of the workshop on particle swarm optimization (Indianapolis, IN)*.

Elloumi, W., El Abed, H., Abraham, A., & Alimi, A. M. (2014). A comparative study of the improvement of performance using a PSO modified by ACO applied to TSP. *Applied Soft Computing*, *25*, 234–241.

Engelbrecht, A. P. (2007). *Computational intelligence: an introduction*. John Wiley & Sons.

Greco, F. (2008). *Travelling salesman problem*. In-Tech.

Hoffmann, M., Mühlenthaler, M., Helwig, S., & Wanka, R. (2011). Discrete particle swarm optimization for TSP: theoretical results and experimental evaluations. *Lecture Notes in Computer Science*, *6943 LNAI*, 416–427.

Kalczynski, P. (2005). A Java implementation of the branch and bound algorithm: The asymmetric traveling salesman problem. *Journal of Object Technology*, *4*(1), 155–163. https://doi.org/10.5381/jot.2005.4.1.a5

Kennedy, J., & Eberhart, R. C. (1995). Particle Swarm Optimization. *in Proceedings of the 1995 IEEE International Conference on Neural Network*, 1942–1948.

Madona, E., Irmansyah, M., Pengajar, S., Teknik, J., Politeknik, E., & Padang, N. (2013). Aplikasi Metode Nearest Neighbor Pada Penentuan. *Jurnal Elektron*, *5* no.*2*, 45–53.

Mataija, M., Rakamarić Šegić, M., & Jozić, F. (2016). Solving the Travelling Salesman Problem Using the Branch and Bound Method. *Zbornik Veleučilišta u Rijeci*, *4*(1), 259–270.

Panda, M. (2018). Performance Comparison of Genetic Algorithm, Particle Swarm Optimization and Simulated Annealing Applied to TSP. *International Journal of Applied Engineering Research*, *13*(9), 6808–6816.

Piotrowski, A. P., Napiorkowski, J. J., & Piotrowska, A. E. (2020). Population size in Particle Swarm Optimization. *Swarm and Evolutionary Computation*, *58*.

Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle Swarm Optimization: An Overview. *Swarm Intelligence*, *1*(1), 33–57.

Rao, S. S. (2019). Engineering optimization: Theory and practice. In *Engineering Optimization: Theory and Practice*. https://doi.org/10.1002/9781119454816

Sachin Desale, Akhtar Rasool, Sushil Andhale, & Priti Rane. (2015). Heuristic and Meta-Heuristic Algorithms and Their Relevance to the Real World: A Survey. *International Journal of Computer Engineering in Research Trends*, *2*(5), 296–304.

Said, M. S., Soesanti, I., Kusumawardani, S. S., Grafika, J., & Yogyakarta, N. (2016). Analisis Integrasi Elastic Net pada Self-Organizing Map untuk penyelesaian Travelling Salesman Problem. *Prosiding SNMIPA*, *2*, 78–86.

Shi, Y., Eberhart, R. C., Shi, Y., Clerc, M., Kaveh, A., Zolghadr, A., Wang, Y., Li, B., Weise, T., Wang, J., Yuan, B., Tian, Q., Krink, T., Vesterstrom, J. S., Riget, J., & Glover, F. (1998). A modified particle swarm optimizer. *Artificial Evolution*, *1363*(February 1998), 1951–1957.

Suyudi, M., Sukono, Mamat, M., & Bon, A. T. (2019). Solving traveling salesman problems using branch and bound methods. *Proceedings of the International Conference on Industrial Engineering and Operations Management*, *July*, 1606–1614.

Yan, X., Zhang, C., & Luo, W. (2012). Solve Traveling Salesman Problem Using Particle Swarm Optimization Algorithm. *International Journal of Computer Science Issues*, *9*(6), 264–271.

Yi, J., Yang, G., Zhang, Z., & Tang, Z. (2009). An improved elastic net method for traveling salesman problem. *Neurocomputing*, 1329–1335.

LAMPIRAN

LAMPIRAN

Lampiran 1. Data penelitian TSP Simetri

Data TSP Simetri untuk 5 kota

	kota 1	**kota 2**	**kota 3**	**kota 4**	**kota 5**
kota 1	0	20	72	37	75
kota 2	20	0	35	12	30
kota 3	72	35	0	66	35
kota 4	37	12	66	0	59
kota 5	75	30	35	59	0

Data TSP Simetri ukuran 10 kota

	kota 1	**kota 2**	**kota 3**	**kota 4**	**kota 5**	**kota 6**	**kota 7**	**kota 8**	**kota 9**	**kota 10**
kota 1	0	94	16	75	92	46	89	54	21	40
kota 2	94	0	3	65	8	71	14	56	62	39
kota 3	16	3	0	47	78	22	9	31	14	40
kota 4	75	65	47	0	63	45	53	72	82	38
kota 5	92	8	78	63	0	50	59	19	11	89
kota 6	46	71	22	45	50	0	95	87	17	15
kota 7	89	14	9	53	59	95	0	36	27	96
kota 8	54	56	31	72	19	87	36	0	64	1
kota 9	21	62	14	82	11	17	27	64	0	95
kota 10	40	39	40	38	89	15	96	1	95	0

Data TSP Simetri ukuran 15 kota

	kota 1	**kota 2**	**kota 3**	**kota 4**	**kota 5**	**kota 6**	**kota 7**	**kota 8**	**kota 9**	**kota 10**	**kota 11**	**kota 12**	**kota 13**	**kota 14**	**kota 15**
kota 1	0	212	931	481	943	601	720	220	530	510	541	213	973	109	815
kota 2	212	0	893	954	961	631	349	138	620	109	295	275	208	993	396
kota 3	931	893	0	435	329	107	560	290	935	429	142	282	230	151	861
kota 4	481	954	435	0	260	427	135	920	670	975	888	389	937	581	882
kota 5	943	961	329	260	0	276	919	325	716	505	965	277	154	343	259
kota 6	601	631	107	427	276	0	310	985	914	323	531	825	693	892	552
kota 7	720	349	560	135	919	310	0	603	845	694	755	653	267	629	658
kota 8	220	138	290	920	325	985	603	0	321	733	733	584	676	223	564
kota 9	530	620	935	670	716	914	845	321	0	373	785	368	919	513	851
kota 10	510	109	429	975	505	323	694	733	373	0	656	739	574	493	218
kota 11	541	295	142	888	965	531	755	733	785	656	0	476	309	863	430
kota 12	213	275	282	389	277	825	653	584	368	739	476	0	427	122	408
kota 13	973	208	230	937	154	693	267	676	919	574	309	427	0	764	706
kota 14	109	993	151	581	343	892	629	223	513	493	863	122	764	0	188
kota 15	815	396	861	882	259	552	658	564	851	218	430	408	706	188	0

Data TSP simetri ukuran 20 kota

	Kota 1	**Kota 2**	**Kota 3**	**Kota 4**	**Kota 5**	**Kota 6**	**Kota 7**	**Kota 8**	**Kota 9**	**Kota 10**	**kota 11**	**kota 12**	**kota 13**	**kota 14**	**kota 15**	**kota 16**	**kota 17**	**kota 18**	**kota 19**	**kota 20**
kota 1	0	331	406	254	925	168	805	583	428	462	723	555	353	598	525	300	585	279	315	751
kota 2	331	0	526	418	994	389	446	648	358	529	466	532	599	259	899	538	262	121	599	284
kota 3	406	526	0	869	255	869	924	638	915	761	830	681	618	964	722	210	379	232	526	190
kota 4	254	418	869	0	579	131	113	832	673	578	319	427	675	661	636	887	456	944	569	782
kota 5	925	994	255	579	0	152	116	283	799	467	581	582	408	910	375	295	284	651	893	801
kota 6	168	389	869	131	152	0	429	416	888	954	717	692	321	567	675	353	562	415	500	338
kota 7	805	446	924	113	116	429	0	933	912	143	313	671	652	496	445	239	499	932	326	450
kota 8	583	648	638	832	283	416	933	0	150	825	541	347	493	904	713	113	911	364	174	345
kota 9	428	358	915	673	799	888	912	150	0	969	499	707	505	564	453	771	493	956	241	462
kota 10	462	529	761	578	467	954	143	825	969	0	868	156	978	268	803	428	796	519	354	110
kota 11	723	466	830	319	581	717	313	541	499	868	0	415	300	733	412	677	914	943	452	659
kota 12	555	532	681	427	582	692	671	347	707	156	415	0	278	525	424	414	699	216	463	106
kota 13	353	599	618	675	408	321	652	493	505	978	300	278	0	140	155	589	378	792	563	125
kota 14	598	259	964	661	910	567	496	904	564	268	733	525	140	0	824	907	653	978	498	744
kota 15	525	899	722	636	375	675	445	713	453	803	412	424	155	824	0	706	134	721	565	282
kota 16	300	538	210	887	295	353	239	113	771	428	677	414	589	907	706	0	591	813	651	973
kota 17	585	262	379	456	284	562	499	911	493	796	914	699	378	653	134	591	0	986	570	647
kota 18	279	121	232	944	651	415	932	364	956	519	943	216	792	978	721	813	986	0	644	285
kota 19	315	599	526	569	893	500	326	174	241	354	452	463	563	498	565	651	570	644	0	755
kota 20	751	284	190	782	801	338	450	345	462	110	659	106	125	744	282	973	647	285	755	0

Lampiran 2. Data Peneliian TSP Asimetri

Data TSP Asimetri ukuran 5 kota

	kota 1	**kota 2**	**kota 3**	**kota 4**	**kota 5**
kota 1	0	89	87	78	97
kota 2	88	0	101	82	91
kota 3	96	109	0	110	93
kota 4	75	96	102	0	100
kota 5	89	90	70	101	0

Data TSP Asimetri ukuran 10 kota

	kota 1	**kota 2**	**kota 3**	**kota 4**	**kota 5**	**kota 6**	**kota 7**	**kota 8**	**kota 9**	**kota 10**
kota 1	0	103	75	88	74	99	79	83	96	73
kota 2	87	0	108	110	105	95	77	100	87	79
kota 3	82	82	0	92	98	71	73	104	110	96
kota 4	72	108	95	0	107	86	110	91	76	89
kota 5	109	110	94	108	0	93	101	88	70	90
kota 6	75	109	91	97	85	0	84	81	108	99
kota 7	94	96	99	108	70	105	0	96	83	80
kota 8	104	96	88	109	70	76	102	0	96	83
kota 9	78	95	73	97	104	94	97	93	0	95
kota 10	78	109	103	89	85	70	79	72	85	0

Data TSP Asimetri ukuran 15 kota

	kota 1	**kota 2**	**kota 3**	**kota 4**	**kota 5**	**kota 6**	**kota 7**	**kota 8**	**kota 9**	**kota 10**	**kota 11**	**kota 12**	**kota 13**	**kota 14**	**kota 15**
kota 1	0	549	673	933	539	311	618	885	673	260	751	407	670	595	250
kota 2	272	0	931	459	838	714	894	157	910	604	397	494	814	226	147
kota 3	100	279	0	177	156	652	239	389	916	777	257	955	265	664	134
kota 4	623	823	553	0	387	814	756	564	942	457	521	553	227	542	515
kota 5	776	673	259	306	0	604	913	866	236	308	336	381	198	287	504
kota 6	665	705	123	480	476	0	385	346	110	655	235	695	572	898	402
kota 7	408	642	698	305	716	648	0	831	837	853	960	181	617	602	675
kota 8	351	315	443	479	468	940	725	0	351	661	808	844	654	759	732
kota 9	546	426	783	496	924	320	555	584	0	116	135	162	653	956	568
kota 10	400	664	708	167	202	225	918	134	375	0	694	159	127	807	190
kota 11	781	447	997	342	451	473	870	554	780	924	0	686	106	252	540
kota 12	346	252	114	352	304	425	111	300	311	891	975	0	679	674	853
kota 13	554	179	632	166	225	793	762	179	942	499	260	322	0	368	667
kota 14	804	989	147	326	858	896	492	796	359	404	701	154	523	0	893
kota 15	175	284	512	268	191	882	277	744	416	216	771	617	853	672	0

Data TSP Asimetri ukuran 20 kota

	kota 1	**kota 2**	**kota 3**	**kota 4**	**kota 5**	**kota 6**	**kota 7**	**kota 8**	**kota 9**	**kota 10**	**kota 11**	**kota 12**	**kota 13**	**kota 14**	**kota 15**	**kota 16**	**kota 17**	**kota 18**	**kota 19**	**kota 20**
kota 1	0	100	241	916	600	679	672	473	322	463	983	198	621	135	905	578	622	550	496	591
kota 2	238	0	619	191	672	833	675	129	607	227	214	691	430	859	244	387	836	280	802	737
kota 3	680	125	0	109	800	563	157	546	804	711	822	985	965	219	737	848	355	808	670	511
kota 4	451	446	120	0	532	399	886	514	905	820	914	716	805	778	334	651	841	285	480	669
kota 5	1000	912	421	309	0	884	865	562	505	896	555	836	170	726	210	577	475	852	921	226
kota 6	371	523	157	490	256	0	971	184	568	756	136	407	475	826	257	270	148	665	655	721
kota 7	710	339	962	708	588	692	0	622	794	292	944	543	619	267	278	638	967	190	126	712
kota 8	254	663	108	612	686	661	413	0	824	825	576	830	712	832	551	450	309	452	499	753
kota 9	845	749	444	880	264	488	604	558	0	971	852	397	759	646	174	199	389	479	599	109
kota 10	148	841	676	310	588	132	499	248	491	0	215	571	521	649	373	643	642	829	874	285
kota 11	519	681	254	496	655	308	547	870	283	680	0	600	781	345	262	279	190	426	746	863
kota 12	995	119	411	250	762	901	182	823	592	791	853	0	720	483	864	202	348	630	450	772
kota 13	729	725	944	179	778	782	901	856	449	815	232	142	0	775	552	850	343	734	484	643
kota 14	799	921	183	827	408	519	287	362	292	210	266	693	330	0	296	203	260	723	316	255
kota 15	705	221	869	684	805	473	183	679	604	797	935	652	763	388	0	131	212	820	271	109
kota 16	180	312	925	135	321	497	372	511	532	319	679	486	150	880	278	0	840	291	597	457
kota 17	256	506	627	355	879	872	819	202	206	365	956	879	609	216	843	827	0	433	248	773
kota 18	475	687	375	471	860	633	386	743	619	927	110	760	981	632	831	910	182	0	564	249
kota 19	101	927	330	317	232	398	776	816	912	528	379	620	365	141	354	916	168	409	0	165
kota 20	979	619	852	571	874	942	105	893	255	560	638	813	622	487	739	590	270	780	950	0

Lampiran 3. Program Python untuk *Generate* data TSP Simetri

```
import random
import pandas as pd

# Fungsi untuk membuat data TSP simetris dengan ukuran
tertentu
def generate_symmetric_tsp_data(size):
    tsp_data = {}

    cities = ["kota " + str(i+1) for i in range(size)]
    for i, city in enumerate(cities):
        connections = {}
        for j, other_city in enumerate(cities):
            if i != j:
                if j < i:
                    connections[other_city] =
tsp_data[other_city][city]
                else:
                    connections[other_city] =
random.randint(100, 1000)
        tsp_data[city] = connections

    return tsp_data

# Membuat data TSP simetris dengan ukuran
sizedata = int(input("Ukuran Data :"))
data_tsp_symmetric = generate_symmetric_tsp_data(sizedata)

# Cetak data TSP simetris
for city, connections in data_tsp_symmetric.items():
    print(f"{city}: {connections}")


# tabel_tsp_symmetric = pd.DataFrame(data_tsp_symmetric,
index=["kota " + str(i+1) for i in range(sizedata)])
# tabel_tsp_symmetric = tabel_tsp_symmetric.fillna(0)
# tabel_tsp_symmetric = tabel_tsp_symmetric.astype(int)
# print(tabel_tsp_symmetri
```

Lampiran 4. Progam Python *Generate* data TSP Asimetri

```
import random
import pandas as pd

Fungsi untuk membuat data TSP dengan ukuran tertentu
def generate_tsp_data(size):
    tsp_data = {}
    cities = [chr(65 + i) for i in range(size)]  # Membuat
daftar nama kota
    cities = [f"kota {i+1}" for i in range(size)]
    cities = ["kota {}".format(i+1) for i in range(size)]
    cities = ["kota " + str(i+1) for i in range(size)]
    for city in cities:
        connections = {}
        for other_city in cities:
            if other_city != city:
                connections[other_city] = random.randint(100,
1000)  # Jarak antara kota acak antara 1 dan 100
        tsp_data[city] = connections

    return tsp_data

# Membuat data TSP dengan ukuran
sizedata = int(input("Ukuran Data :"))
data_tsp = generate_tsp_data(sizedata)

# Cetak data TSP
for city, connections in data_tsp.items():
    #print(f"{city}: {connections}")
tabel_tsp = pd.DataFrame(data_tsp, index = ["kota " + str(i+1)
for i in range(sizedata)])
tabel_tsp = tabel_tsp.fillna(0)
tabel_tsp = tabel_tsp.astype(int)
print(tabel_tsp)
```

Lampiran 5: Progam Implementasi Branch and Bound

```
import numpy as np
import sys

def tsp_branch_and_bound(distances):
    num_cities = len(distances)
    start_city = 'kota 3'  # Start city is now 'kota 1'
    unvisited = set(distances.keys())  # Use keys to
initialize unvisited set
    unvisited.remove(start_city)
    min_cost = sys.maxsize
    optimal_path = []
    min_costs = []  # List untuk menyimpan minimum cost pada
setiap iterasi
    def dfs(current_city, path, total_cost):
        nonlocal min_cost, optimal_path

        if len(path) == num_cities:
            total_cost += distances[current_city][start_city]
            if total_cost < min_cost:
                min_cost = total_cost
                optimal_path = path + [start_city]
                min_costs.append(min_cost)  # Menyimpan
minimum cost pada setiap iterasi
            return

        for next_city in unvisited.copy():  # Make a copy to
avoid modifying the original set
            new_path = path + [next_city]
            new_cost = total_cost +
distances[current_city][next_city]
            if new_cost < min_cost:
                unvisited.remove(next_city)
                dfs(next_city, new_path, new_cost)
                unvisited.add(next_city)

    dfs(start_city, [start_city], 0)

    return min_cost, optimal_path, min_costs
```

```
min_cost, path, min_costs = tsp_branch_and_bound(data_tsp)
print("Jarak Minimum:", min_cost)
print("Rute Optimal:", path)
```

Lampiran 6: Program Implementasi Particle Swarm Optimization

```
class Particle:
    def __init__(self, cities):
        self.position = np.random.permutation(len(cities))
        self.velocity = np.random.rand(len(cities))
        self.best_position = self.position.copy()
        self.best_cost = float('inf')
```

```
def calculate_cost(path, data_tsp):
    cost = 0
    for i in range(len(path) - 1):
        current_city = path[i]
        next_city = path[i + 1]
        cost += data_tsp[current_city][next_city]
    last_city = path[-1]
    first_city = path[0]
    cost += data_tsp[last_city][first_city]  # Complete the
loop
    return cost
```

```
# Parameters for PSO
cities = list(data_tsp.keys())
num_particles = 500
max_iter = 10000
w_max = 0.9
w_min = 0.4
c1 = 2  # Cognitive coefficient
c2 = 2  # Social coefficient
```

```
import numpy as np

def update_velocity(particle, global_best_position,
inertia_weight, c1, c2):
    inertia_term = w * particle.velocity
    cognitive_term = c1 * np.random.rand() *
(particle.best_position - particle.position)
    social_term = c2 * np.random.rand() *
(global_best_position - particle.position)
    new_velocity = inertia_term + cognitive_term + social_term
    return new_velocity

def update_position(particle, new_velocity):
```

```
    new_position = np.argsort(np.argsort(particle.position +
new_velocity)) % len(particle.position)
    return new_position

particles = [Particle(cities) for _ in range(num_particles)]

# Main PSO loop
global_best_particle = min(particles, key=lambda x:
x.best_cost)

for iteration in range(max_iter):
    w = (w_max - w_min) * iteration / max_iter
    for particle in particles:
        # Evaluate current position
        current_cost = calculate_cost([cities[i] for i in
particle.position], data_tsp)

        # Update personal best if necessary
        if current_cost < particle.best_cost:
            particle.best_cost = current_cost
            particle.best_position = particle.position.copy()

        # Update global best if necessary
        if current_cost < global_best_particle.best_cost:
            global_best_particle = particle

    for particle in particles:
        # Update velocity and position
        new_velocity = update_velocity(particle,
global_best_particle.best_position, w, c1, c2)
        particle.velocity = new_velocity
        particle.position = update_position(particle,
new_velocity)
```

```
# Print the best path found
best_path = [cities[i] for i in
global_best_particle.best_position]
best_path.append(best_path[0])
print("Best Path:", best_path)
print("Best Cost:", global_best_particle.best_cost)
```