Kauno technologijos universitetas

Informatikos fakultetas

Inžinerinio projekto ataskaita

individuali užduotis

Aistis Jakutonis

Studentas

Dalius Makackas
Andrius Kriščiūnas
Tadas Kraujalis
Vidmantas Rimavičius

Dėstytojai

Kaunas, 2025

Turinys

1. Pirma dalis

1.1. Asimptotinis programos vykdymo laiko sudėtingumas

Nr.	Kodas	Laikas	Kartai
1	`static Place FindNearest(Place from,`		
2	`List<Place> unvisited)`		
3	`    {`		
4	`        Place nearest = null;`	C1	1
5	`        double minDist = double.MaxValue;`	C2	1
6			
7	`        foreach (var place in unvisited)`	C3	$unvisited$
8	`        {`		
9	`            double dist =`	C4	$unvisited$
10	`from.DistanceTo(place);`		
11	`            if (dist < minDist)`	C5	$unvisited$
12	`            {`		
13	`                minDist = dist;`	C6	$unvisited$
14	`                nearest = place;`	C7	$unvisited$
15	`            }`		
16	`        }`		
17			
18	`        return nearest;`	C8	1
19	`    }`		

$$T_{FindNearest}(from, unvisited) = C + unvisited = O(unvisited) = O(n)$$

$$T_{FindNearest}(from, unvisited) = O(n)$$

Nr.	Kodas	Laikas	Kartai
1	`static void Nuoseklus(List<Place> unvisited,`		
2	`Place current1,`		
3	`        Place current2, Place current3,`		
4	`List<double> route1,`		
5	`        List<double> route2, List<double>`		
6	`route3, Place start)`		
7	`    {`		
8	`        while (unvisited.Count > 0)`	C1	$\frac{unvisited}{3}$
9	`        {`		
10	`            Place nearest1 =`	C2	$\frac{unvisited}{3} * O(unvisited)$
11	`FindNearest(current1, unvisited);`		
12	`            if (nearest1 != null)`	C3	$\frac{unvisited}{3}$
13	`            {`		
14			
15	`route1.Add(current1.DistanceTo(nearest1));`	C4	$\frac{unvisited}{3}$
16	`                current1 = nearest1;`	C5	
17	`                unvisited.Remove(nearest1);`	C6	$\frac{unvisited}{3}$
18	`            }`		$\frac{unvisited}{3}$
19			
20	`            if (unvisited.Count == 0)`	C7	$\frac{unvisited}{3}$
21	`            {`		
22	`                break;`	C8	$\frac{unvisited}{3}$
23	`            }`		
24			
25	`            Place nearest2 =`	C9	$\frac{unvisited}{3} * O(unvisited)$
26	`FindNearest(current2, unvisited);`		

27	`if (nearest2 != null)`	C10	$\frac{unvisited}{3}$
28	`{`		
29			
30	`route2.Add(current2.DistanceTo(nearest2));`	C11	$\frac{unvisited}{3}$
31	`current2 = nearest2;`	C12	
32	`unvisited.Remove(nearest2);`	C13	$\frac{unvisited}{3}$
33	`}`		$\frac{unvisited}{3}$
34			
35	`if (unvisited.Count == 0)`	C14	$\frac{unvisited}{3}$
36	`{`		
37	`break;`	C15	$\frac{unvisited}{3}$
38	`}`		
39			
40	`Place nearest3 =`	C16	$\frac{unvisited}{3} * O(unvisited)$
41	`FindNearest(current3, unvisited);`		
42	`if (nearest3 != null)`	C17	$\frac{unvisited}{3}$
43	`{`		
44			
45	`route3.Add(current3.DistanceTo(nearest3));`	C18	$\frac{unvisited}{3}$
46	`current3 = nearest3;`	C19	
47	`unvisited.Remove(nearest3);`	C20	$\frac{unvisited}{3}$
48	`}`		$\frac{unvisited}{3}$
49	`}`		
50			
51	`route1.Add(current1.DistanceTo(start));`	C21	1
52	`route2.Add(current2.DistanceTo(start));`	C22	1
53	`route3.Add(current3.DistanceTo(start));`	C23	1
54	`}`		

$$T_{Nuoseklus}(unvisited, cur1, cur2, cur3, r1, r2, r3, start) = C + \frac{unvisited}{3} + \frac{unvisited}{3} * O(unvisited) = O\left(\frac{unvisited^2}{3}\right)$$
$$= O(unvisited^2) = O(n^2)$$

$$T_{Nuoseklus}(unvisited, cur1, cur2, cur3, r1, r2, r3, start) = O(unvisited^2)$$

Galutinis kodo sudėtingumas: $O(n^2)$

1.2. Programos abstraktus aprašas – pseudo kodas

Funkcija Nuoseklus(unvisited, current1, current2, current3, route1, route2, route3, start):

Kol yra neaplankytų vietų:

// 1 autobusas ieško artimiausios vietos

Rask artimiausią vietą iš current1 tarp neaplankytų

Jei tokia vieta rasta:

Įrašyk atstumą į route1

current1 = ta vieta

Pašalink vietą iš neaplankytų

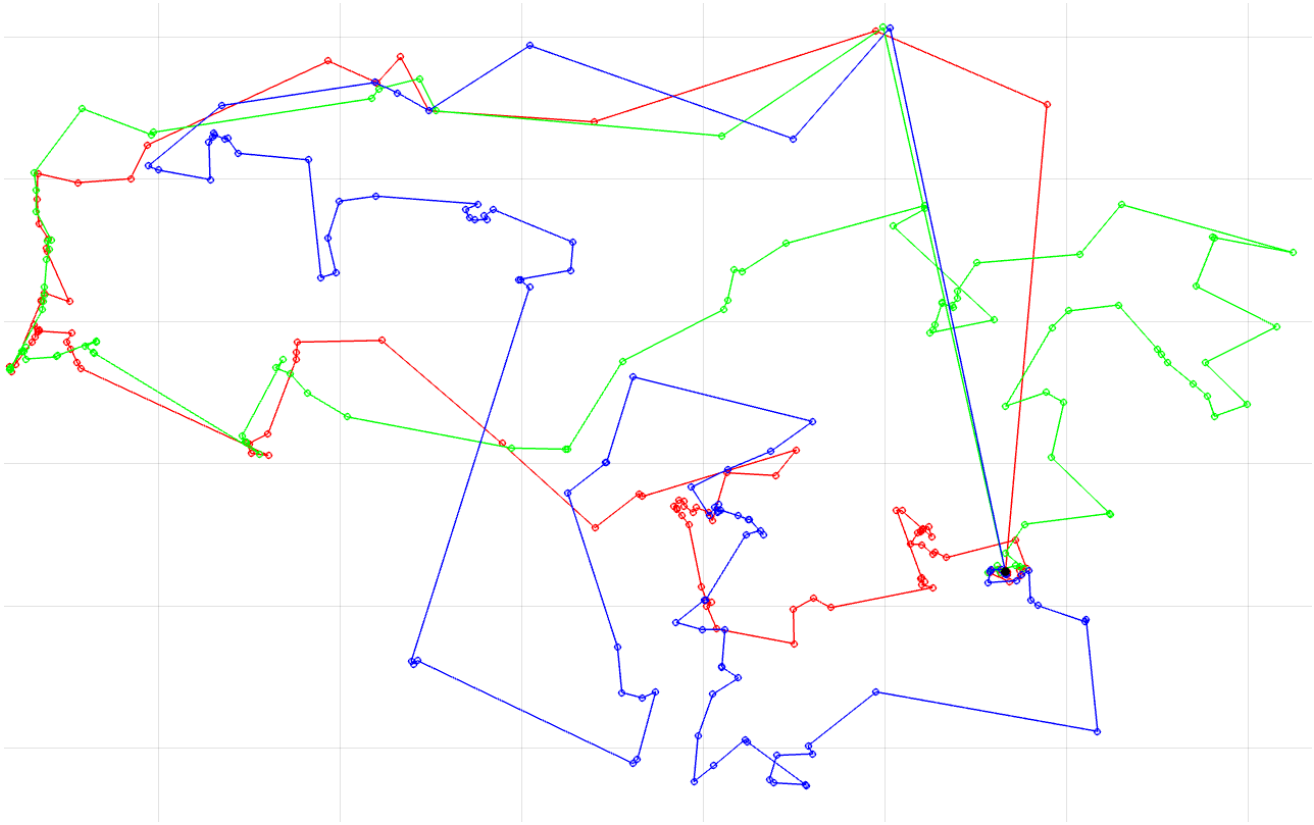
```
    Jei nėra daugiau neaplankytų, nutrauk ciklą


    // 2 autobusas ieško artimiausios vietos
    Rask artimiausią vietą iš current2 tarp neaplankytų
    Jei tokia vieta rasta:
        Įrašyk atstumą į route2
        current2 = ta vieta
        Pašalink vietą iš neaplankytų


    Jei nėra daugiau neaplankytų, nutrauk ciklą


    // 3 autobusas ieško artimiausios vietos
    Rask artimiausią vietą iš current3 tarp neaplankytų
    Jei tokia vieta rasta:
        Įrašyk atstumą į route3
        current3 = ta vieta
        Pašalink vietą iš neaplankytų


// Kai visos vietos aplankytos, grįžtama į pradinį tašką
Pridėk atstumą nuo current1 iki start į route1
Pridėk atstumą nuo current2 iki start į route2
Pridėk atstumą nuo current3 iki start į route3
```

1.3. Metodų paaiškinimas ir analizė

Šiame nuosekliame sprendime yra naudojamas artimiausio kaimyno heuristinis metodas. Kol yra neaplankytų vietų algoritmas eina per kiekvieną autobusą paeiliui ir ieško tarp visų likusių vietų pačios artimiausios ir į ją važiuoja. Taip tęsiamas algoritmas iki kol nebelieka nepalankytų vietų. Šis metodas nėra visada optimalus, jis nesiekia rasti geriausio įmanomo sprendimo, o renkasi tiesiog artimiausią vietą.

1.4. Grafikai pavaizduotas sprendimas

1 grafikas Artimiausio kaimyno metodu važiuojančių autobusų maršrutų grafikas



1.5. Algoritmo išlygiagretinimas

Nr.	Kodas	Laikas	Kartai
1	`static void Islygiagretintas(List<Place>`		
2	`sharedUnvisited, List<double> route1,`		
3	`List<double> route2, List<double> route3, Place`		
4	`start, Place current1, Place current2, Place`		
5	`current3)`		
6	`{`		
7	`object lockObj = new object();`	C1	1
8			
9	`Parallel.Invoke(`	C2	$\frac{\frac{unvisited}{3} * (Ounvisited)}{3}$
10	`() =>`		
11	`{`		
12	`while (true)`	C3	
13	`{`		
14	`Place next = null;`	C4	$\frac{unvisited}{3}$
15			
16	`lock (lockObj)`	C5	$\frac{unvisited}{3}$
17	`{`		
18	`if`	C6	$\frac{unvisited}{3}$
19	`(sharedUnvisited.Count == 0)`		$\frac{unvisited}{3}$
20	`{`		
21	`break;`	C7	
22	`}`		$\frac{unvisited}{3}$
23			
24	`next =`	C8	
25	`FindNearest(current1, sharedUnvisited);`		

26			
27	`if (next != null)`	C9	$\frac{unvisited}{3} * O(unvisited)$
28	`{`		
29			$\frac{unvisited}{3}$
30	`sharedUnvisited.Remove(next);`	C10	
31	`}`		$\frac{unvisited}{3}$
32	`}`		
33			
34	`if (next != null)`	C11	
35	`{`		$\frac{unvisited}{3}$
36			
37	`route1.Add(current1.DistanceTo(next));`	C12	
38	`current1 = next;`	C13	$\frac{unvisited}{3}$
39	`}`		
40	`}`		$\frac{unvisited}{3}$
41			
42	`route1.Add(current1.DistanceTo(start));`	C14	
43	`},`		1
44	`() =>`		
45	`{`		
46	`while (true)`	C15	$\frac{unvisited}{3}$
47	`{`		
48	`Place next = null;`	C16	
49			$\frac{unvisited}{3}$
50	`lock (lockObj)`	C17	
51	`{`		$\frac{unvisited}{3}$
52	`if`	C18	
53	`(sharedUnvisited.Count == 0)`		$\frac{unvisited}{3}$
54	`{`		
55	`break;`	C19	$\frac{unvisited}{3}$
56	`}`		
57			
58	`next =`	C20	$\frac{unvisited}{3} * O(unvisited)$
59	`FindNearest(current2, sharedUnvisited);`		
60			
61	`if (next != null)`	C21	$\frac{unvisited}{3}$
62	`{`		
63			
64	`sharedUnvisited.Remove(next);`	C22	$\frac{unvisited}{3}$
65	`}`		
66	`}`		
67			
68	`if (next != null)`	C23	$\frac{unvisited}{3}$
69	`{`		
70			
71	`route2.Add(current2.DistanceTo(next));`	C24	$\frac{unvisited}{3}$
72	`current2 = next;`	C25	
73	`}`		$\frac{unvisited}{3}$
74	`}`		
75			
76	`route2.Add(current2.DistanceTo(start));`	C26	
77	`},`		1
78	`() =>`		
79	`{`		
80	`while (true)`	C27	$\frac{unvisited}{3}$
81	`{`		
82	`Place next = null;`	C28	
83			$\frac{unvisited}{3}$
84	`lock (lockObj)`	C29	

85	{		$\frac{unvisited}{3}$
86	if	C30	$\frac{unvisited}{3}$
87	(sharedUnvisited.Count == 0)		$\frac{unvisited}{3}$
88	{		
89	break;	C31	$\frac{unvisited}{3}$
90	}		
91			
92	next =	C32	
93	FindNearest(current3, sharedUnvisited);		$\frac{unvisited}{3} * O(unvisited)$
94			
95	if (next != null)	C33	$\frac{unvisited}{3}$
96	{		
97			$\frac{unvisited}{3}$
98	sharedUnvisited.Remove(next);	C34	
99	}		$\frac{unvisited}{3}$
100	}		
101			
102	if (next != null)	C35	
103	{		$\frac{unvisited}{3}$
104			
105	route3.Add(current3.DistanceTo(next));	C36	
106	current3 = next;	C37	$\frac{unvisited}{3}$
107	}		
108	}		$\frac{unvisited}{3}$
109			
110	route3.Add(current3.DistanceTo(start));	C38	1
111	}		
112	);		
113	}		

$$T_{Islygiagretinimas}(unvisited) = \frac{\frac{unvisited}{3} * O(unvisited)}{3} = \frac{unvisited^2}{9} = O(unvisited^2) = O(n^2)$$

Galutinis kodo sudėtingumas: $O(n^2)$. Veikia šiek tiek greičiau nei prieš tai buvęs nuoseklus.

1.6. Išlygiagretinimo paaiškinimas

Visi trys autobusai dalijasi vienu vietų sąrašu. Todėl kiekvienas autobusas vykdo savo atskiras paralelines užduotis ir ieško artimiausių kelių iki kitų vietų. Kadangi visi autobusai turi prieigą prie bendro sąrašo ir renkasi duomenis iš jo, tai tam yra naudojamas užraktas, kad kol vienas autobusas renkasi duomenis kiti tų duomenų negalėtų pamodifikuoti. Tokiu metodu maršrutų skaičiavimas vyksta kiek greičiau, bet dėl autobusų lygiagrečių skaičiavimų rezultatai nesutampa su nuosekliais.

2. Antra dalis

2.1. Asimptotinis programos vykdymo laiko sudėtingumas

Nr.	Kodas	Laikas	Kartai
1	`static void Branch(State state, List<Place>`		
2	`remaining, Place start, ref double bestTime,`		
3	`ref State bestState, Stopwatch timer)`		
4	`    {`		
5	`        if (timer.Elapsed.TotalSeconds > 10)`	C1	1
6	`        {`		
7	`            Console.WriteLine("timeend");`	C2	1
8	`            return;`	C3	1
9	`        }`		
10			
11	`        if (state.Visited.Count ==`	C4	1
12	`remaining.Count)`		
13	`        {`		
14	`            var cloned = state.Clone();`	C5	1
15	`            for (int i = 0; i < 3; i++)`	C6	4
16	`            {`		
17	`                var last =`	C7	3
18	`cloned.Routes[i].Last();`		
19	`                cloned.Distances[i] +=`	C8	3
20	`last.DistanceTo(start);`		
21	`                cloned.Routes[i].Add(start);`	C9	3
22	`            }`		
23			
24	`            double maxTime =`	C10	1
25	`cloned.Distances.Max();`		
26	`            if (maxTime < bestTime)`	C11	1
27	`            {`		
28	`                bestTime = maxTime;`	C12	1
29	`                bestState = cloned;`	C13	1
30	`            }`		
31	`            return;`	C14	1
32	`        }`		
33			
34	`        foreach (var place in remaining)`	C15	$remaining$
35	`        {`		
36	`            if`	C16	$remaining$
37	`(state.Visited.Contains(place.Id))`		
38	`            {`		
39	`                continue;`	C17	$remaining$
40	`            }`		
41			
42	`            for (int bus = 0; bus < 3; bus++)`	C18	$remaining * (3 + 1)$
43	`            {`		
44	`                var newState = state.Clone();`	C19	$remaining * 3$
45	`                var lastPlace =`	C20	$remaining * 3$
46	`newState.Routes[bus].Last();`		
47			
48	`newState.Routes[bus].Add(place);`	C21	$remaining * 3$
49	`                newState.Distances[bus] +=`	C22	$remaining * 3$
50	`lastPlace.DistanceTo(place);`		
51	`                newState.Visited.Add(place.Id);`	C23	$remaining * 3$
52			
53			

54
55
56
57
58
59
60
	`if (newState.Bound() < bestTime)`
	`Branch(newState, remaining, start, ref bestTime, ref bestState, timer);`
	`}`
	`}`
	`}`
		C24
		
		C25
			$remaining * 3$
			
			$remaining * 3 * T_{Branch}(newState, remaining, start, bestTime, bestState, timer)$

$$T_{Branch}(n) = C + remaining * 3 * T_{Branch}(remaining - 1) + 4 * remaining = 3n * T(n-1) + 4n = O(3^n * n!)$$

54

55 Galutinis kodo sudėtingumas: $O(3^n * n!)$

56

57 **2.2. Programos abstraktus aprašas – pseudo kodas**

Funkcija Branch(būsena, likusiosVietos, startoVieta, geriausiasLaikas, geriausiaBūsena, laikmatis):

Jei praeita daugiau nei 10 sekundžių:

Baigti (laikas baigėsi)

Jei visos vietos jau aplankytos:

Sukurti kopiją iš esamos būsenos

Kiekvienam autobusui:

Pridėti kelią nuo paskutinės vietos iki starto

Jei maksimalus maršruto ilgis < geriausiasLaikas:

Atnaujinti geriausią laiką

Išsaugoti esamą būseną kaip geriausią

Grįžti

Kiekvienai neaplankytai vietai:

Jei vieta jau aplankyta – praleisti

Kiekvienam autobusui:

Sukurti naują būsenos kopiją

Gauti paskutinę autobuso maršruto vietą

Pridėti pasirinktą vietą į autobuso maršrutą

Atnaujinti maršruto atstumą

Pažymėti vietą kaip aplankytą

Jei naujos būsenos maksimalus atstumas (Bound) < geriausiasLaikas:

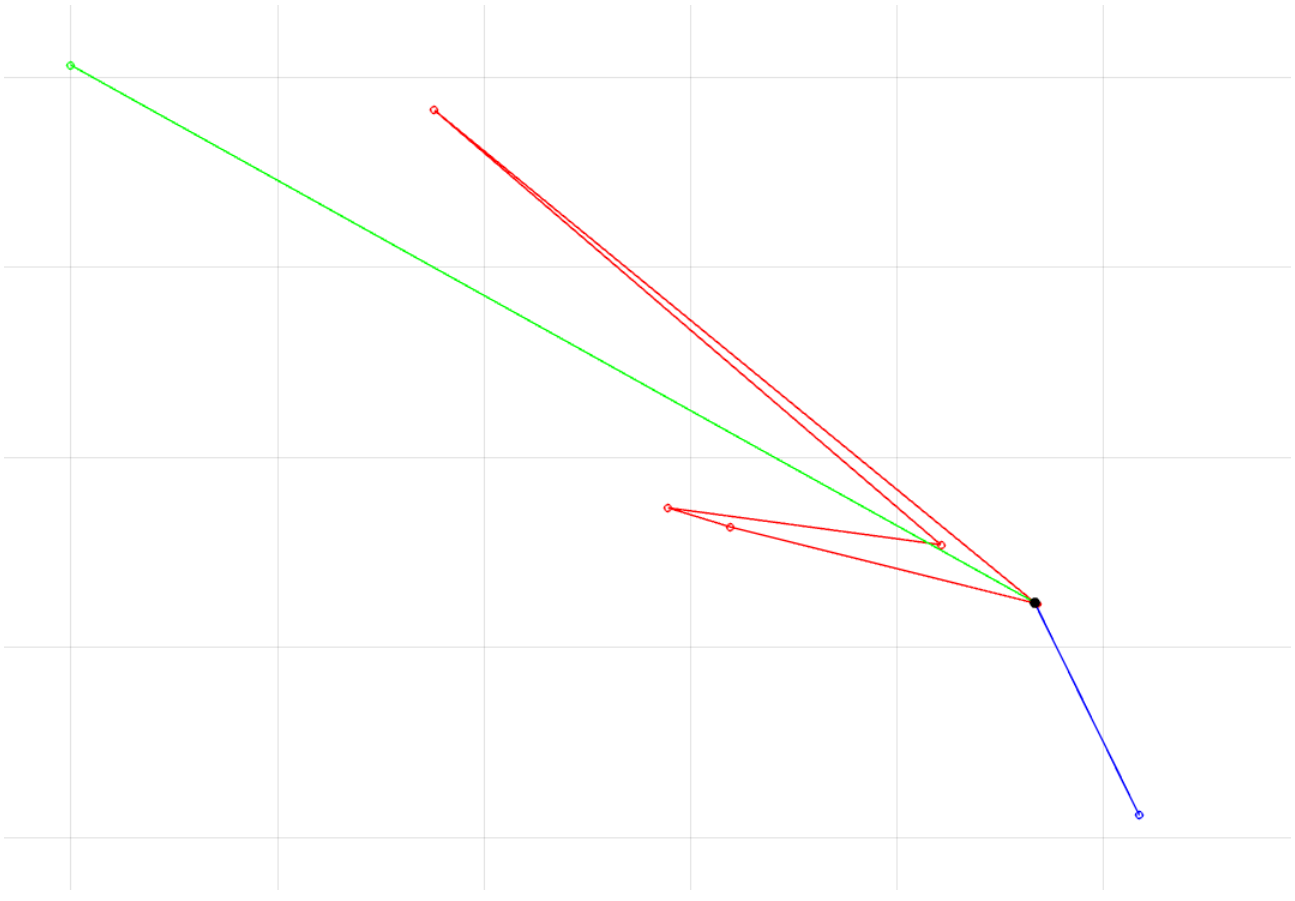
Rekursyviai iškviesti Branch su nauja būsena

2.3. Metodų paaiškinimas ir analizė

Čia yra naudojamas šakų ir rėžių metodas. Jis ieško geriausio maršruto visiems trims autobusams kartu. Kiekviename žingsnyje bandoma priskirti naują neaplankytą vietą vienam iš autobusų ir taip eina gilyn rekursiškai. Tuomet, jei dabartinė būsena jau yra blogesnė nei geriausias rastas sprendinys, tai tada ta šaka yra nutraukiama. Grįžtama šiek tiek atgal ir bandoma naudoti kitą vietą ir taip toliau. Galiausiai yra randamas optimalus sprendinys. Tačiau šis algoritmas tikrindamas labai daug kombinacijų užtrunka gan ilgai.

2.4. Grafikai pavaizduotas sprendimas

2 grafikas Šakų ir rėžių metodu važiuojančių autobusų optimaliausi maršrutai



Nr.	Kodas	Laikas	Kartai
1	`static (double bestTime, State bestState)`		
2	`BranchParallel(State state, List<Place>`		
3	`remaining, Place start, Stopwatch timer, int`		
4	`depth = 0)`		
5	`    {`		
6	`        if (timer.Elapsed.TotalSeconds > 10)`	C1	1
7	`            return (double.MaxValue, null);`	C2	1
8			
9	`        if (state.Visited.Count ==`	C3	1
10	`remaining.Count)`		
11	`        {`		
12	`            var cloned = state.Clone();`	C4	1
13	`            for (int i = 0; i < 3; i++)`	C5	4
14	`            {`		
15	`                var last =`	C6	3
16	`cloned.Routes[i].Last();`		
17	`                cloned.Distances[i] +=`	C7	3
18	`last.DistanceTo(start);`		
19	`                cloned.Routes[i].Add(start);`	C8	3
20	`            }`		
21			
22	`            double maxTime =`	C9	3
23	`cloned.Distances.Max();`		
24	`            return (maxTime, cloned);`	C10	3
25	`        }`		
26			
27	`        double bestTime = double.MaxValue;`	C11	1
28	`        State bestState = null;`	C12	1
29			
30	`        var tasks = new List<Task<(double,`	C13	1
31	`State)>>();`		
32			
33	`        foreach (var place in remaining)`	C14	$remaining$
34	`        {`		
35	`            if`	C15	$remaining$
36	`(state.Visited.Contains(place.Id))`		
37	`                continue;`	C16	$remaining$
38			
39	`            for (int bus = 0; bus < 3; bus++)`	C17	$remaining * 4$
40	`            {`		
41	`                var newState = state.Clone();`	C18	$remaining * 3$
42	`                var lastPlace =`	C19	$remaining * 3$
43	`newState.Routes[bus].Last();`		
44			
45	`newState.Routes[bus].Add(place);`	C20	
46	`                newState.Distances[bus] +=`	C21	$remaining * 3$
47	`lastPlace.DistanceTo(place);`		$remaining * 3$
48	`                newState.Visited.Add(place.Id);`	C22	
49			$remaining * 3$
50	`                if (newState.Bound() >=`	C23	
51	`bestTime)`		$remaining * 3$
52	`                    continue;`	C24	
53			$remaining * 3$
54	`                if (depth < 2)`	C25	
55	`                {`		$remaining * 3$
56	`                    tasks.Add(Task.Run(() =>`	C26	

57	{
	return
	BranchParallel(newState, remaining, start,
	timer, depth + 1);
	}));
	}
	else
	{
	var (bTime, bState) =
	BranchParallel(newState, remaining, start,
	timer, depth + 1);
	if (bState != null && bTime
	< bestTime)
	{
	bestTime = bTime;
	bestState = bState;
	}
	}
	}
	}
	
	if (tasks.Count > 0)
	{
	Task.WaitAll(tasks.ToArray());
	
	foreach (var t in tasks)
	{
	var (tTime, tState) = t.Result;
	if (tState != null && tTime <
	bestTime)
	{
	bestTime = tTime;
	bestState = tState;
	}
	}
	}
	
	return (bestTime, bestState);
	}
		C27
		C28
		C29
		C30
		C31
		C32
		C33
		C34
		C35
		C36
		C37
		C38
		C39
		C40
			$remaining * 3$
			$remaining * 3 * T_{BranchParallel}$
			$remaining * 3$
			$remaining * 3$
			$remaining * 3$
			$remaining * 3$
			$remaining * 3$
			1
			1
			$tasks$
			$tasks$
			$tasks$
			$tasks$
			$tasks$
			1

$$T_{BranchParallel}(n) = C + n * \left(4 * \left(3 * T_{BranchParallel}(n-1)\right)\right) = O(3^n * n!)$$

Galutinis kodo sudėtingumas: $O(3^n * n!)$

2.6. Išlygiagretinimo paaiškinimas

Šiame išlygiagretinime pradžioje kiekviena vieta bandoma priskirti visiems trims autobusams lygiagrečiai. Vėliau, jei gylis nėra didelis, kiekviena šaka taip pat yra vykdoma atskirai. Čia kaip ir praeitame atvejyje duomenys yra apsaugojami juos užrakinant.

3. Trečia dalis

3.1. Asimptotinis programos vykdymo laiko sudėtingumas

Nr.	Kodas	Laikas	Kartai
1	`static double RouteDistance(List<Place> route,`		
2	`Place start)`		
3	`    {`		
4	`        double dist = 0.0;`	C1	1
5	`        var current = start;`	C2	1
6			
7	`        foreach (var p in route)`	C3	*route*
8	`        {`		
9	`            dist += current.DistanceTo(p);`	C4	*route*
10	`            current = p;`	C5	*route*
11	`        }`		
12			
13	`        dist += current.DistanceTo(start);`	C6	1
14			
15	`        return dist;`	C7	1
16	`    }`		

$$T_{RouteDistance}(n) = C + n = O(n)$$

79

Nr.	Kodas	Laikas	Kartai
1	`static BusSolution`		
2	`MutateBusSolution(BusSolution solution)`		
3	`    {`		
4	`        var newSol = solution.Clone();`	C1	1
5			
6	`        if (rand.NextDouble() < 0.5)`	C2	1
7	`        {`		
8	`            int from = rand.Next(3);`	C3	1
9	`            int to = rand.Next(3);`	C4	1
10	`            if (from != to &&`	C5	1
11	`newSol.Routes[from].Count > 0)`		
12	`            {`		
13	`                int index =`	C6	1
14	`rand.Next(newSol.Routes[from].Count);`		
15	`                var place =`	C7	1
16	`newSol.Routes[from][index];`		
17			
18	`newSol.Routes[from].RemoveAt(index);`	C8	1
19	`                newSol.Routes[to].Add(place);`	C9	1
20	`            }`		
21	`        }`		
22	`        else`	C10	1
23	`        {`		
24	`            int bus = rand.Next(3);`	C11	1
25	`            if (newSol.Routes[bus].Count > 1)`	C12	1
26	`            {`		
27	`                int i =`	C13	1
28	`rand.Next(newSol.Routes[bus].Count);`		
29	`                int j =`	C14	1
30	`rand.Next(newSol.Routes[bus].Count);`		
31			

32 33 34 35 36 37 38 39	(newSol.Routes[bus][i], newSol.Routes[bus][j]) = (newSol.Routes[bus][j], newSol.Routes[bus][i]); } } return newSol; }	C15 C16	1 1 1
$T_{MutateBusSolution}(n) = C = O(C)$			

80

Nr.	Kodas	Laikas	Kartai

1
2
3
4
5
6
7
8
9
10
11
12
13
14
	static BusSolution
	CreateInitialBusSolution(List<Place> places)
	{
	var solution = new BusSolution();
	var shuffled = places.OrderBy(x =>
	rand.Next()).ToList();
	
	for (int i = 0; i < shuffled.Count;
	i++)
	solution.Routes[i %
	3].Add(shuffled[i]);
	
	return solution;
	}
		
		C1
		C2
		
		C3
		
		C4
		
		C5
			
			1
			1
			
			$places + 1$
			
			$places$
			
			1

$T_{CreateInitialBusSolution}(n) = n + C = O(n)$			

81

Nr.	Kodas	Laikas	Kartai

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
	static BusSolution
	SimulatedAnnealingNuoseklus(List<Place>
	allPlaces, Place start, Stopwatch timer)
	{
	double temp = 1000;
	double coolingRate = 0.99999;
	double minTemp = 1e-4;
	int noImprovement = 0;
	int maxNoImprovement = 10000;
	
	var current =
	CreateInitialBusSolution(allPlaces);
	var best = current.Clone();
	double bestDist = best.Routes.Max(r =>
	RouteDistance(r, start));
	
	int i = 0;
	
	while (temp > minTemp && noImprovement
	< maxNoImprovement)
	{
	if (timer.Elapsed.TotalSeconds >
	10)
		
		C1
		C2
		C2
		C4
		C5
		
		C6
		
		C7
		C8
		
		C9
		
		C10
		
		C11
			
			1
			1
			1
			1
			1
			
			n
			
			1
			n
			
			1
			
			1

24	{		
25	Console.WriteLine("timeend");	C12	1
26	break;	C13	1
27	}		
28			
29	var next =	C14	1
30	MutateBusSolution(current);		
31	double nextDist = next.Routes.Max(r	C15	n
32	=> RouteDistance(r, start));		
33	double delta = nextDist - bestDist;	C16	1
34	i++;	C17	1
35			
36	if (delta < 0 || Math.Exp(-delta /	C18	1
37	temp) > rand.NextDouble())		
38	{		
39	current = next;	C19	1
40	if (nextDist < bestDist)	C20	1
41	{		
42	best = next.Clone();	C21	1
43	bestDist = nextDist;	C22	1
44	noImprovement = 0;	C23	1
45	Console.WriteLine($"{i} =	C24	1
46	{bestDist}");		
47	}		
48	else noImprovement++;	C25	1
49	}		
50			
51	temp *= coolingRate;	C26	1
52	}		
53			
54	return best;	C27	1
55	}		

$$T_{SimulatedAnnealingNuoseklus}(n) = C + n * \log\left(\frac{temp}{minTemp}\right) = O\left(n * \log\left(\frac{temp}{minTemp}\right)\right)$$

82

83 ### 3.2. Programos abstraktus aprašas – pseudo kodas

Funkcija SimulatedAnnealingNuoseklus(vietos, starto_taskas, laikmatis):

Inicializuoti pradine temperatura = 1000

Nustatyti vėsinimo tempą (pvz., 0.99999)

Nustatyti minimalią temperatūrą ir maksimalių iteracijų be pagerėjimo skaičių

Sukurti pradine atsitiktinę sprendimo versiją (trys autobusų maršrutai)

Apskaičiuoti maksimalų vieno iš trijų autobusų maršruto ilgį – tai bus "geriausias iki šiol"

Kol temperatūra > minimali ir nepagerinta per daug iteracijų:

Jei praejo daugiau nei 10 sekundžių – išeiti

// Sukuriamas naujas kandidatinis sprendimas:

Kopijuoti dabartinį sprendimą kaip naują

Atsitiktinai pasirinkti vieną iš dviejų mutacijų:

1) Perkelti atsitiktinę vietą iš vieno autobuso į kitą:

- Pasirinkti atsitiktinius du skirtingus autobusus (FROM ir TO)
- Jei FROM turi bent vieną vietą:
  - Pašalinti atsitiktinę vietą iš FROM
  - Pridėti ją į TO

ARBA

2) Sukeisti dvi vietas vieno autobuso maršrute:

- Pasirinkti atsitiktinį autobusą
- Jei jis turi bent 2 vietas:
  - Atsitiktinai parinkti dvi pozicijas ir sukeisti jas vietomis

Apskaičiuoti šio kandidato maksimalų vieno autobuso maršruto ilgį

Skirtumas = kandidato ilgis - geriausias ilgis

Jei skirtumas < 0 (geresnis) ARBA priimamas pagal tikimybę (Math.Exp(-delta / temp)):

Priimti kandidatą kaip naują dabartinį sprendimą

Jei jis taip pat yra geresnis nei geriausias iki šiol:

Atnaujinti geriausią sprendimą

Nustatyti pagerėjimų skaitiklį į 0

Kitu atveju:

Padidinti be pagerėjimo skaitiklį

Temperatūra mažinama (temp *= coolingRate)

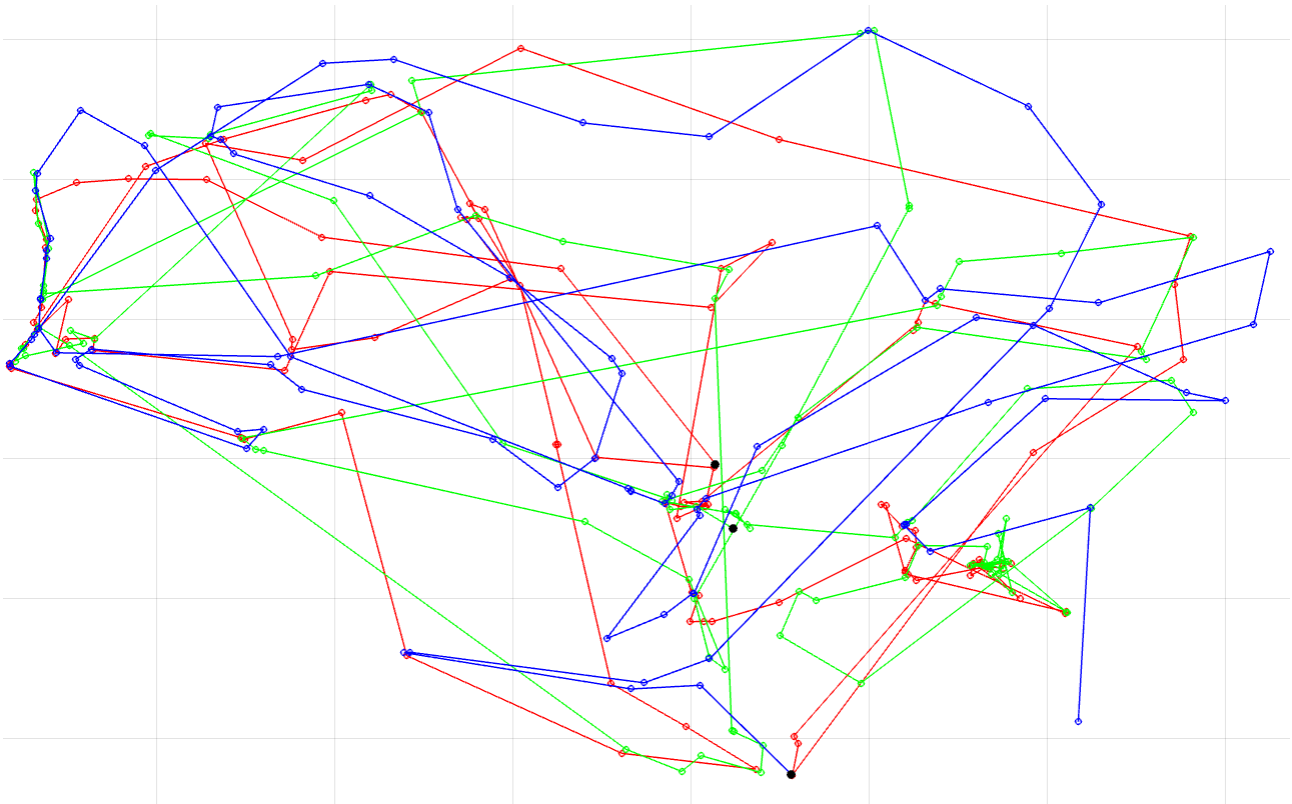
Grąžinti geriausią rastą sprendimą

3.3. Metodų paaiškinimas ir analizė

Čia buvo naudojamas atvėsinimo algoritmas, kuris leidžia ieškoti optimalaus sprendimo. Šiame algoritme kiekvienas galimas būsimos būsenos variantas yra gaunamas taikant atsitiktinę mutaciją – tai gali būti viename autobuse vietų sukeitimas ar tarp dviejų autobusų vietų keitimas. Kiekviena nauja būsena yra įvertinama ir jei ji yra geresnė tai ji yra priimama. Jei blogesnė ji taip pat gali būti priimama su tam tikra tikimybe, priklausančia nuo temperatūros ir sprendinių skirtumo. Temperatūra vis mažinama, taip ribojant blogesnių sprendinių priėmimą vėlesnėse iteracijose.

3.4. Grafikai pavaizduotas sprendimas

3 grafikas Atkaitinimo metodu važiuojančių autobusų grafikas



3.5. Diskretinio optimizavimo parametrai užduoties sprendimui

Sprendžiant užduotį buvo taikytas diskretinis optimizavimas, kurio paieškos erdvę sudarė visos įmanomos vietų kombinacijos paskirstant jas į 3 autobusus. Sprendinys – tai konkretus vietų išdėstymas tarp autobusų, o tikslo funkcija buvo maksimalaus vieno autobuso maršruto ilgis, kurį siekta minimizuoti. Apribojimų tiesiogiai nebuvo taikoma, išskyrus tai, kad kiekviena vieta gali būti paskirta tik vienam autobusui. Pagrindiniai optimizavimo algoritmo parametrai:

- **Pradinė temperatūra**: 1000 – leido pradžioje laisvai priimti ir blogesnius sprendimus.
- **Temperatūros mažinimo koeficientas**: 0.99999 – užtikrino laipsnišką perėjimą prie mažesnės sprendimų priėmimo rizikos.
- **Minimalios temperatūros riba**: 0.0001 – sustabdymo kriterijus.
- **Maksimalus negerėjančių iteracijų skaičius**: 10 000 – taip pat naudotas kaip sustabdymo sąlyga.

3.6. Algoritmo išlygiagretinimas

Nr.	Kodas	Laikas	Kartai
1	`static BusSolution`		
2	`SimulatedAnnealingLygiagretus(List<Place>`		
3	`allPlaces, Place start, Stopwatch timer)`		
4	`    {`		
5	`        double temp = 1000;`	C1	1
6	`        double coolingRate = 0.99999;`	C2	1
7	`        double minTemp = 1e-4;`	C3	1
8	`        int noImprovement = 0;`	C4	1
9	`        int maxNoImprovement = 10000;`	C5	1
10	`        int candidatesPerIter =`	C6	1
11	`Environment.ProcessorCount;`		
12			
13	`        var current =`	C7	n
14	`CreateInitialBusSolution(allPlaces);`		
15	`        var best = current.Clone();`	C8	1
16	`        double bestDist = best.Routes.Max(r =>`	C9	n
17	`RouteDistance(r, start));`		
18			
19	`        while (temp > minTemp && noImprovement`	C10	
20	`< maxNoImprovement)`		
21	`        {`		
22	`            if (timer.Elapsed.TotalSeconds >`	C11	1
23	`10)`		
24	`            {`		
25	`                Console.WriteLine("timeend");`	C12	1
26	`                break;`	C13	1
27	`            }`		
28			
29	`            var candidates = new`	C14	1
30	`BusSolution[candidatesPerIter];`		
31	`            Parallel.For(0, candidatesPerIter,`	C15	1
32	`i =>`		
33	`            {`		
34	`                candidates[i] =`	C16	1
35	`MutateBusSolution(current);`		
36	`            });`		
37			
38	`            BusSolution bestCandidate = null;`	C17	1
39		C18	1

40	`double bestCandidateDist =`		
41	`double.MaxValue;`		
42		C19	1
43	`foreach (var candidate in`		
44	`candidates)`		
45	`{`	C20	n
46	`double dist =`		
47	`candidate.Routes.Max(r => RouteDistance(r,`		
48	`start));`	C21	1
49	`if (dist < bestCandidateDist)`		
50	`{`	C22	1
51	`bestCandidate = candidate;`	C23	1
52	`bestCandidateDist = dist;`		
53	`}`		
54	`}`		
55		C24	1
56	`double delta = bestCandidateDist -`		
57	`bestDist;`	C25	1
58	`if (delta < 0 || Math.Exp(-delta /`		
59	`temp) > rand.NextDouble())`		
60	`{`	C26	1
61	`current = bestCandidate;`	C27	1
62	`if (bestCandidateDist <`		
63	`bestDist)`		
64	`{`	C28	1
65	`best =`		
66	`bestCandidate.Clone();`	C29	1
67	`bestDist =`		
68	`bestCandidateDist;`	C30	1
69	`noImprovement = 0;`		
70	`}`	C31	1
71	`else`		
72	`{`	C32	1
73	`noImprovement++;`		
74	`}`		
75	`}`		
76		C33	1
77	`temp *= coolingRate;`		
78	`}`		
79		C34	1
80	`return best;`		
	`}`		

$$T_{SimulatedAnnealingLygiagretus}(n) = C + n * p = O(n * p)$$

$$p - procesų\ skaičius$$

Galutinis kodo sudėtingumas: $O(n * p)$

3.7. Išlygiagretinimo paaiškinimas

Kiekvienoje iteracijoje vietoje vieno sprendinio mutacijos, lygiagrečiai sukuriama tiek kandidatų sprendinių, kiek yra procesorių branduolių. Šie sprendiniai generuojami naudojant tą pačią MutateBusSolution funkciją, bet vykdomi lygiagrečiai. Iš sugeneruotų variantų išrenkamas geriausias

ir jis lyginamas su dabartiniu sprendimu pagal atvėsinimo kriterijų. Šis metodas leidžia vienu metu išbandyti kelis galimus patobulinimus.