Sterowanie procesami dyskretnymi

Kierunek	Specjalność	Rok studiów	Numer grupy lab.
Automatyka i Robotyka	*Robotyka*	*3*	*2*

Temat Laboratorium	Numer prog.
Problem przepływowy: NEH	*4*

Skład grupy laboratoryjnej

Michał Antoszko, Adam Karpiński

Prowadzący	Data
Mgr inż. Radosław Idzikowski	*11 maja 2018*

1 Opis problemu

Problem dotyczy sortowania zadań wykonywanych na wielu maszynach, w taki sposób, aby zostały zrealizowane w jak najkrótszym czasie.

Każde zadanie opisane jest parametrem P po jednym na każdą maszynę, gdzie:

- *p* - czas wykonania,

Każde zadanie musi być wykonywane nieprzerwanie i przejść kolejno po wszystkich maszynach. Na każdej maszynie zadania są wykonywane w tej samej kolejności.

2 Metoda rozwiązania

2.1 Calculate Cmax

Obliczanie czasu zakończenia wykonywania ostatniego zadania na ostatniej maszynie.

Listing 1: Calculate

```
1  unsigned int calculate(std::vector <machine> &container)
2  {
3      const unsigned int machines = container.size(), tasks = container[0].p.size();
4
5      for (auto &i : container)
6      {
7          i.c.clear();
8          i.s.clear();
9      }
10
11     for (unsigned int i = 0; i < machines; ++i)
12     {
13         for (unsigned int j = 0; j < tasks; ++j)
14         {
15             container[i].s.push_back(0);
16             container[i].c.push_back(0);
17         }
18     }
19     unsigned int startTime = NULL, endTime = NULL;
20
21     for (unsigned int i = 0; i < tasks; ++i)
22     {
23         if (i == 0)
24         {
25             startTime = 0;
26         }
27         else
28         {
29             startTime = container[0].c[i - 1];
30         }
31         endTime = startTime + container[0].p[i].value;
32
33         container[0].s[i] = startTime;
```

```
34          container[0].c[i] = endTime;
35
36          for (unsigned int j = 1; j < machines; ++j)
37          {
38
39              if (i == 0)
40              {
41                  startTime = endTime;
42              }
43              else
44              {
45                  startTime = std::max(container[j].c[i - 1], container[j - 1].c[i]);
46              }
47              endTime = startTime + container[j].p[i].value;
48
49              container[j].s[i] = startTime;
50              container[j].c[i] = endTime;
51          }
52      }
53      return container[machines - 1].c[tasks - 1];
54  }
```

2.2 Sortowanie nierosnąco według czasu wykonywania zadania na wszystkich maszynach

Listing 2: Algorytm sort

```
std::sort( priority.begin(), priority.end(), [](task a, task b)
            {return a.value > b.value; } );
```

2.3 Algorytm NEH

Algorytm NEH polega na pobieraniu po kolei każdego zadania z posortowanej listy, wstawieniu go pomiędzy już uszeregowane zadania i sprawdzeniu, dla którego miejsca w szeregu, daje on wynik najlepszy z możliwych. Ta procedura jest powtarzana do momentu uszeregowania wszystkich zadań.

Listing 3: NEH

```
int NEH(std::vector <machine> &container)
{
    std::vector <task> priority;

    for (int i = 0; i < container[0].p.size(); ++i)
    {
        task temp = { static_cast<unsigned int>(0), i };
        priority.push_back(temp);
    }

    for (int i = 0; i < container.size(); ++i)
    {
        for (int j = 0; j < container[0].p.size(); ++j)
        {
            priority[j].value += container[i].p[j].value;
        }
    }

    std::sort(priority.begin(), priority.end(), [](task a, task b) {return a.value > b.v

    std::vector <machine> sorted;

    for (int i = 0; i < container.size(); ++i)
    {
        machine temp;
        sorted.push_back(temp);
        for (int j = 0; j < container[0].p.size(); ++j)
        {
            task temp2 = container[i].p[priority[j].index];
            sorted[i].p.push_back(temp2);
        }
    }

    std::vector <std::vector <machine>> table;
    std::vector <machine> toRemember;
```

```
36
37      for (int i = 0; i < sorted.size(); ++i)
38      {
39          machine temp;
40          toRemember.push_back(temp);
41          task temp2 = sorted[i].p[0];
42          toRemember[i].p.push_back(temp2);
43      }
44
45      for (int i = 0; i < sorted.size(); ++i)
46      {
47          sorted[i].p.erase(sorted[i].p.begin());
48      }
49
50      int ind = NULL;
51      while (!sorted[0].p.empty())
52      {
53          for (int i = 0; i < toRemember[0].p.size() + 1; ++i)
54          {
55              table.push_back(toRemember);
56          }
57
58          for (int i = 0; i < toRemember[0].p.size() + 1; ++i)
59          {
60              for (int j = 0; j < toRemember.size(); ++j)
61              {
62                  table[i][j].p.insert(table[i][j].p.begin() + i, sorted[j].p[0]);
63              }
64          }
65
66          for (int i = 0; i < sorted.size(); ++i)
67          {
68              sorted[i].p.erase(sorted[i].p.begin());
69          }
70
71          int value = INT_MAX;
72          for (int i = 0; i < table.size(); ++i) {
73
74              int temp = calculate(table[i]);
75              if (temp < value) {
76                  value = temp;
77                  ind = i;
78              }
79          }
80
81          toRemember = table[ind];
82          table.clear();
83      }
84      return calculate(toRemember);
85  }
```

2.4 NEH Plus

Algorytm NEH Plus to zmodyfikowany algorytm NEH. Po dodaniu kolejnego zadania w najlepsze miejsce wybiera inne, już posortowane zadanie i tak jak poprzednio, wstawiając je między inne uszeregowane już zadania, wybiera pozycję która daje najlepszy wynik.

Wybierane zadanie to takie, którego usunięcie spowoduję największe zmniejszenie wartości Cmax.

Listing 4: NEH Plus

```
1  int NEHPlus(std::vector <machine> &container)
2  {
3      std::vector <task> priority;
4
5      for (int i = 0; i < container[0].p.size(); ++i)
6      {
7          task temp = { 0, i };
8          priority.push_back(temp);
9      }
10
11     for (int i = 0; i < container.size(); ++i)
12     {
13         for (int j = 0; j < container[0].p.size(); ++j)
14         {
15             priority[j].value += container[i].p[j].value;
16         }
17     }
18
19     std::sort(priority.begin(), priority.end(), [](task a, task b) {return a.value > b.v
20
21     std::vector <machine> sorted;
22
23     for (int i = 0; i < container.size(); ++i)
24     {
25         machine temp;
26         sorted.push_back(temp);
27         for (int j = 0; j < container[0].p.size(); ++j)
28         {
29             task temp2 = container[i].p[priority[j].index];
30             sorted[i].p.push_back(temp2);
31         }
32     }
33
34     std::vector <std::vector <machine>> table;
35     std::vector <machine> toRemember;
36
37     for (int i = 0; i < sorted.size(); ++i)
38     {
39         machine temp;
40         toRemember.push_back(temp);
41         task temp2 = sorted[i].p[0];
42         toRemember[i].p.push_back(temp2);
43     }
```

```
44
45      for (int i = 0; i < sorted.size(); ++i)
46      {
47          sorted[i].p.erase(sorted[i].p.begin());
48      }
49
50      int ind = NULL;
51      int zzz = 1;
52
53      while (!sorted[0].p.empty())
54      {
55          for (int i = 0; i < toRemember[0].p.size() + 1; ++i)
56          {
57              table.push_back(toRemember);
58          }
59
60          for (int i = 0; i < toRemember[0].p.size() + 1; ++i)
61          {
62              for (int j = 0; j < toRemember.size(); ++j)
63              {
64                  table[i][j].p.insert(table[i][j].p.begin() + i, sorted[j].p[0]);
65              }
66          }
67
68          for (int i = 0; i < sorted.size(); ++i)
69          {
70              sorted[i].p.erase(sorted[i].p.begin());
71          }
72
73          //najmniejszy Cmax
74          int value = INT_MAX;
75          for (int i = 0; i < table.size(); ++i) {
76
77              int temp = calculate(table[i]);
78              if (temp < value) {
79                  value = temp;
80                  ind = i;
81              }
82          }
83
84          toRemember = table[ind];
85          table.clear();
86
87          //////////////////////////////////////////////////////////////////////////
88          if (zzz % 2 == 0)
89          {
90              //ponowne wstawianie elementu bez ktorego CMax jest najmniejsze (4. opcja)
91              for (int i = 0; i < toRemember[0].p.size(); ++i)
92              {
93                  table.push_back(toRemember);
```

```
94              }
95
96              //przechowuje prawdziwe indeksy usuwanych elementow, aby mozna je bylo odzys
97              std::vector <int> indexHunter;
98
99              for (int i = 0; i < toRemember[0].p.size(); ++i)
100             {
101                 //prawdziwe indeksy zaczynaja sie od 1, wiec trzeba je przesunac
102                 indexHunter.push_back(table[i][0].p[i].index - 1);
103
104                 //pomijamy usuwanie ostatnio dodanego elementu, tablica z tym elementem
105                 if (i != ind)
106                 {
107                     for (int j = 0; j < toRemember.size(); ++j)
108                     {
109                         table[i][j].p.erase(table[i][j].p.begin() + i);
110                     }
111                 }
112             }
113
114             //szukanie najmniejszego CMaxa
115             value = INT_MAX;
116             for (int i = 0; i < table.size(); ++i) {
117
118                 int temp = calculate(table[i]);
119                 if (temp < value) {
120                     value = temp;
121                     ind = i;
122                 }
123             }
124
125             toRemember = table[ind];
126             table.clear();
127
128             //wrzucenie elementu to vectora sorted, w celu ponownego dodania
129             std::vector <machine> tempSorted;
130             for (int i = 0; i < container.size(); ++i)
131             {
132                 machine xd;
133                 xd.p.push_back(container[i].p[indexHunter[ind]]);
134                 tempSorted.push_back(xd);
135             }
136
137             for (int i = 0; i < sorted.size(); ++i)
138             {
139                 for (int j = 0; j < sorted[0].p.size(); ++j)
140                 {
141                     tempSorted[i].p.push_back(sorted[i].p[j]);
142                 }
143             }
```

```
144                 sorted = tempSorted;
145             }
146             ++zzz;
147         }
148 
149         return calculate(toRemember);
150     }
```

3 Eksperymenty obliczeniowe

Obliczenia zastały wykonane na komputerze klasy PC z procesorem i7-4702HQ, kartą graficzną NVIDIA GeForce GTX 850m, 16GB RAM i dyskiem SSD. Jako miarę jakości algorytmu przyjęto średnie procentowe odchylenie (Percentage Relative Deviation, PRD) najlepszego otrzymanego rozwiązania π względem rozwiązania referencyjnego π^{ref}:

$$PRD(\pi) = 100\%(C_{max}(\pi) - C_{max}(\pi^{ref}))/C_{max}(\pi^{ref}) \quad (1)$$

Tablica 1: 1/3 *Cmax* w zależności od ilości zadań i wykorzystanych algorytmów

Nazwa instancji	Zadania	Maszyny	12345	NEH	NEH Plus
ta001	20	5	1448	1286	1291
ta002	20	5	1545	1365	1367
ta003	20	5	1597	1159	1139
ta004	20	5	1754	1325	1326
ta005	20	5	1431	1305	1305
ta006	20	5	1616	1228	1212
ta007	20	5	1528	1278	1269
ta008	20	5	1428	1223	1232
ta009	20	5	1468	1291	1274
ta010	20	5	1404	1151	1127
ta011	20	10	2004	1680	1649
ta012	20	10	2104	1729	1752
ta013	20	10	1812	1557	1527
ta014	20	10	1726	1439	1435
ta015	20	10	1944	1502	1438
ta016	20	10	1877	1453	1452
ta017	20	10	1935	1562	1547
ta018	20	10	2044	1609	1565
ta019	20	10	1978	1647	1647
ta020	20	10	2051	1653	1638
ta021	20	20	2770	2410	2345
ta022	20	20	2543	2150	2152
ta023	20	20	2625	2411	2378
ta024	20	20	2800	2262	2252
ta025	20	20	2829	2397	2344
ta026	20	20	2597	2349	2322
ta027	20	20	2723	2362	2309
ta028	20	20	2697	2249	2254
ta029	20	20	2713	2320	2274
ta030	20	20	2830	2277	2313
ta031	50	5	3095	2733	2733
ta032	50	5	3515	2843	2882
ta033	50	5	2900	2643	2629
ta034	50	5	3073	2783	2782
ta035	50	5	3071	2868	2885
ta036	50	5	3195	2850	2845
ta037	50	5	3450	2752	2751
ta038	50	5	3140	2694	2697
ta039	50	5	2930	2576	2590
ta040	50	5	3188	2787	2807

Tablica 2: 2/3 *Cmax* w zależności od ilości zadań i wykorzystanych algorytmów

Nazwa instancji	Zadania	Maszyny	12345	NEH	NEH Plus
ta041	50	10	3754	3135	3146
ta042	50	10	3685	3032	2971
ta043	50	10	3612	3016	2983
ta044	50	10	3669	3198	3146
ta045	50	10	3741	3128	3113
ta046	50	10	3736	3178	3161
ta047	50	10	3678	3277	3289
ta048	50	10	3773	3123	3129
ta049	50	10	3792	3015	2977
ta050	50	10	3845	3257	3204
ta051	50	20	5094	4013	4010
ta052	50	20	4730	3921	3868
ta053	50	20	4592	3923	3838
ta054	50	20	4797	3987	3959
ta055	50	20	4748	3835	3811
ta056	50	20	4946	3914	3924
ta057	50	20	4742	3952	3887
ta058	50	20	4763	3974	3907
ta059	50	20	4823	3952	3873
ta060	50	20	4901	4016	4004
ta061	100	5	5943	5519	5514
ta062	100	5	5878	5284	5286
ta063	100	5	5880	5248	5235
ta064	100	5	5675	5023	5023
ta065	100	5	6095	5266	5261
ta066	100	5	5753	5139	5139
ta067	100	5	5935	5281	5290
ta068	100	5	6068	5129	5124
ta069	100	5	6193	5489	5489
ta070	100	5	6157	5354	5346
ta071	100	10	6983	5835	5890
ta072	100	10	6558	5500	5451
ta073	100	10	6667	5802	5743
ta074	100	10	7300	5966	5968
ta075	100	10	6844	5637	5632
ta076	100	10	6591	5391	5368
ta077	100	10	6765	5735	5716
ta078	100	10	6517	5723	5717
ta079	100	10	6859	6051	6045
ta080	100	10	6930	5915	5903

Tablica 3: 3/3 *Cmax* w zależności od ilości zadań i wykorzystanych algorytmów

Nazwa instancji	Zadania	Maszyny	12345	NEH	NEH Plus
ta081	100	20	7840	6613	6568
ta082	100	20	7591	6565	6520
ta083	100	20	7755	6595	6585
ta084	100	20	7885	6649	6550
ta085	100	20	7729	6653	6581
ta086	100	20	8072	6713	6621
ta087	100	20	8033	6595	6540
ta088	100	20	8138	6785	6735
ta089	100	20	7907	6668	6584
ta090	100	20	8099	6728	6689
ta091	200	10	12193	10942	10928
ta092	200	10	12796	10651	10617
ta093	200	10	12556	11061	11063
ta094	200	10	12198	11057	10931
ta095	200	10	12110	10615	10681
ta096	200	10	12116	10491	10428
ta097	200	10	12848	11001	10965
ta098	200	10	12294	10823	10851
ta099	200	10	12010	10617	10547
ta100	200	10	12274	10791	10728
ta101	200	20	13576	11588	11528
ta102	200	20	13628	11740	11645
ta103	200	20	14152	11825	11742
ta104	200	20	13479	11804	11648
ta105	200	20	13686	11657	11589
ta106	200	20	13917	11689	11659
ta107	200	20	13836	11894	11685
ta108	200	20	13855	11870	11740
ta109	200	20	13409	11631	11657
ta110	200	20	14101	11800	11661
ta111	500	20	30121	26693	26607
ta112	500	20	31202	27197	27100
ta113	500	20	30447	26875	26849
ta114	500	20	30355	27125	26889
ta115	500	20	30099	26837	26594
ta116	500	20	30946	27010	26896
ta117	500	20	30792	26806	26688
ta118	500	20	31034	27144	27066
ta119	500	20	30634	26653	26452
ta120	500	20	30148	27054	26808

Tablica 4: 1/3 Czas wykonania algorytmów w zależności od ilości zadań podany w $[s]$

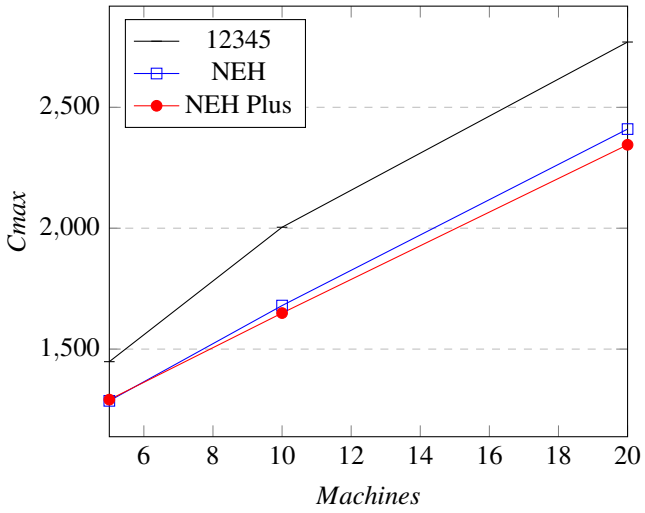
Nazwa instancji	Zadania	Maszyny	NEH	NEH Plus
ta001	20	5	0.001387	0.003226
ta002	20	5	0.000971	0.002577
ta003	20	5	0.000923	0.002369
ta004	20	5	0.000927	0.003301
ta005	20	5	0.001116	0.002591
ta006	20	5	0.001074	0.002737
ta007	20	5	0.000993	0.002400
ta008	20	5	0.002516	0.002287
ta009	20	5	0.000916	0.002320
ta010	20	5	0.000909	0.003715
ta011	20	10	0.001842	0.007778
ta012	20	10	0.001939	0.008022
ta013	20	10	0.003339	0.007453
ta014	20	10	0.002240	0.005701
ta015	20	10	0.001735	0.007772
ta016	20	10	0.003423	0.006475
ta017	20	10	0.001902	0.004970
ta018	20	10	0.001884	0.005634
ta019	20	10	0.001783	0.005683
ta020	20	10	0.002156	0.006628
ta021	20	20	0.003893	0.010915
ta022	20	20	0.005334	0.017352
ta023	20	20	0.004137	0.016441
ta024	20	20	0.005735	0.013180
ta025	20	20	0.003762	0.009722
ta026	20	20	0.003672	0.009995
ta027	20	20	0.004835	0.011023
ta028	20	20	0.003977	0.014632
ta029	20	20	0.003581	0.009664
ta030	20	20	0.006056	0.013389
ta031	50	5	0.010012	0.022076
ta032	50	5	0.008350	0.025669
ta033	50	5	0.009696	0.025967
ta034	50	5	0.011619	0.026979
ta035	50	5	0.011014	0.029601
ta036	50	5	0.009355	0.022393
ta037	50	5	0.009778	0.023889
ta038	50	5	0.009141	0.020840
ta039	50	5	0.007248	0.020798
ta040	50	5	0.009755	0.019713

Tablica 5: 2/3 Czas wykonania algorytmów w zależności od ilości zadań podany w $[s]$

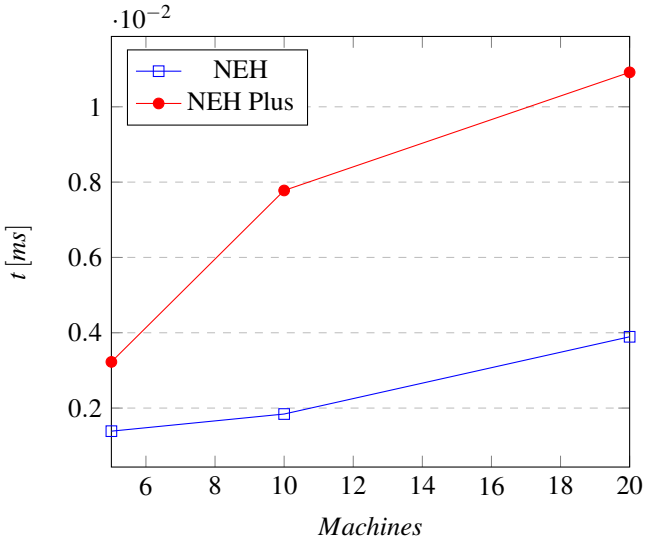
Nazwa instancji	Zadania	Maszyny	NEH	NEH Plus
ta041	50	10	0.014568	0.039324
ta042	50	10	0.014730	0.041144
ta043	50	10	0.017430	0.040695
ta044	50	10	0.015222	0.038792
ta045	50	10	0.015405	0.039761
ta046	50	10	0.016074	0.038010
ta047	50	10	0.014871	0.039236
ta048	50	10	0.014694	0.040114
ta049	50	10	0.015395	0.039628
ta050	50	10	0.015392	0.038103
ta051	50	20	0.028947	0.085142
ta052	50	20	0.034085	0.076397
ta053	50	20	0.031302	0.079946
ta054	50	20	0.029127	0.077660
ta055	50	20	0.029317	0.078479
ta056	50	20	0.029999	0.080383
ta057	50	20	0.029643	0.077230
ta058	50	20	0.029461	0.078216
ta059	50	20	0.028983	0.076072
ta060	50	20	0.030822	0.077587
ta061	100	5	0.041001	0.111423
ta062	100	5	0.042240	0.110162
ta063	100	5	0.042804	0.109896
ta064	100	5	0.046907	0.110401
ta065	100	5	0.041373	0.110592
ta066	100	5	0.041356	0.111114
ta067	100	5	0.042091	0.114701
ta068	100	5	0.041495	0.114540
ta069	100	5	0.044154	0.109739
ta070	100	5	0.041083	0.122960
ta071	100	10	0.086737	0.233614
ta072	100	10	0.082387	0.222299
ta073	100	10	0.086956	0.229994
ta074	100	10	0.085514	0.225931
ta075	100	10	0.082440	0.230509
ta076	100	10	0.085866	0.223789
ta077	100	10	0.085326	0.223333
ta078	100	10	0.081009	0.223413
ta079	100	10	0.083997	0.229272
ta080	100	10	0.088085	0.222156

Tablica 6: 3/3 Czas wykonania algorytmów w zależności od ilości zadań podany w $[s]$

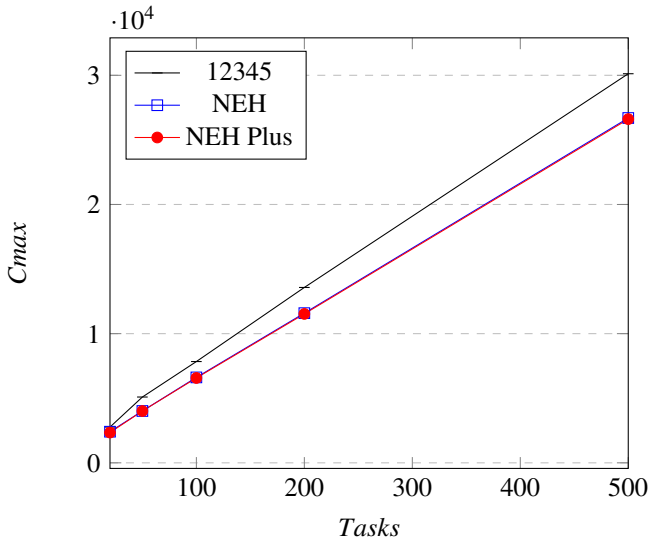
Nazwa instancji	Zadania	Maszyny	NEH	NEH Plus
ta081	100	20	0.183050	0.481384
ta082	100	20	0.181162	0.494936
ta083	100	20	0.179860	0.481809
ta084	100	20	0.178671	0.488125
ta085	100	20	0.179197	0.487170
ta086	100	20	0.176289	0.491850
ta087	100	20	0.179803	0.480436
ta088	100	20	0.174166	0.492978
ta089	100	20	0.179181	0.484501
ta090	100	20	0.178335	0.486105
ta091	200	10	0.594192	1.588208
ta092	200	10	0.597040	1.565593
ta093	200	10	0.581590	1.566281
ta094	200	10	0.587171	1.562483
ta095	200	10	0.581295	1.576825
ta096	200	10	0.572551	1.582875
ta097	200	10	0.574100	1.601213
ta098	200	10	0.595012	1.629012
ta099	200	10	0.571043	1.571600
ta100	200	10	0.587176	1.573950
ta101	200	20	1.321655	3.572809
ta102	200	20	1.268646	3.486508
ta103	200	20	1.276668	3.549681
ta104	200	20	1.268874	3.470078
ta105	200	20	1.258659	3.517066
ta106	200	20	1.253300	3.446863
ta107	200	20	1.294582	3.457849
ta108	200	20	1.246875	3.563790
ta109	200	20	1.380748	3.412982
ta110	200	20	1.249298	3.440312
ta111	500	20	18.219648	46.414028
ta112	500	20	16.521138	54.854994
ta113	500	20	16.879778	47.930028
ta114	500	20	17.356561	49.701401
ta115	500	20	17.314343	48.216171
ta116	500	20	17.255164	48.815254
ta117	500	20	17.010531	47.369448
ta118	500	20	17.086694	46.904198
ta119	500	20	16.979725	45.348505
ta120	500	20	16.409192	45.265529



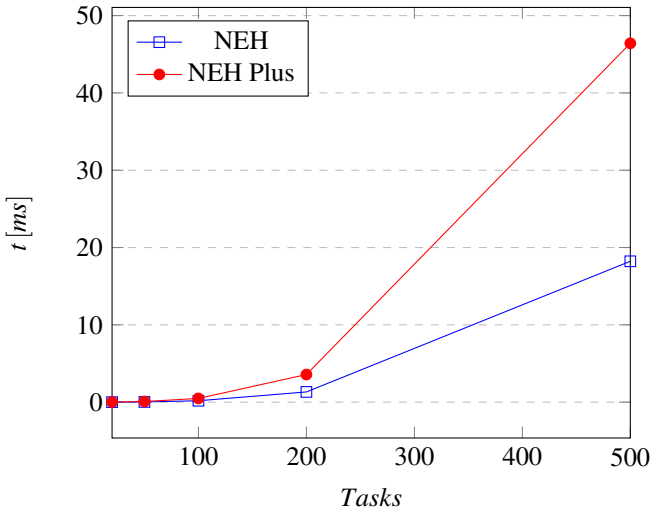
Rysunek 1: *Cmax* wykonywania się algorytmów w zależności od ilości maszyn



Rysunek 2: Czas wykonywania się algorytmów w zależności od ilości maszyn



Rysunek 3: *Cmax* wykonywania się algorytmów w zależności od ilości zadań dla 20 maszy



Rysunek 4: Czas wykonywania się algorytmów w zależności od ilości zadań dla 20 maszy

4 Wnioski

Algorytmy *NEH* i *NEHPlus* dają podobne wyniki, *aleNEHPlus* wykonywany jest znacznie dłużej ze względu na wyższą złożoność obliczeniową.