Sprawozdanie 2
Problemy spełniania ograniczeń

Szymon Woźniak, 235040

12.04.2019

1 Wstęp teoretyczny

1.1 Problem spełniania ograniczeń

Problemy spełniania ograniczeń można zdefiniować poprzez trzy terminy:

- zmienne: $V_1, V_2, ..., V_k$
- dziedziny zmiennych: $D_1, D_2, ..., D_k$
- ograniczenia: $C_1, C_2, ..., C_L$

W każdym z ograniczeń jest może być uwikłanych od 1 do n zmiennych. Celem zadania spełniania ograniczeń, jest znalezienie takiego przypisania wartości v_i każdej zmiennej V_i, że $v_i \in D_i$ i żadne z ograniczeń C_j nie jest złamane. Bardziej ogólnie, celem takiego zadania może być również znaleznienie wszystkich tego typu przypisań i to właśnie zostanie rozpatrzone w tej pracy.

1.2 Rozwiązywane problemy

1.2.1 Wspólne cechy

Zmienne
Oba rozwiązywane w tej pracy problemy, są łamigłówkami logicznymi przedstawianymi jako kwadratowa plansza, składająca się z $n \times n$ pól. W terminach problemów spełniania ograniczeń każde pole jest zmienną o dziedzinie składającej się z liczb całkowitych $1..n$.

Ograniczenia
Dla obu problemów definiuje się również takie samo ograniczenie, mówiące że w żadnej z kolumn i w żadnym z wierszy planszy nie mogą wystąpić powtarzające się wartości. W każdym tego typu problemie występuje zatem co najmniej $2n$ ograniczeń, a w każdym z nich uwikłanych jest n zmiennych.

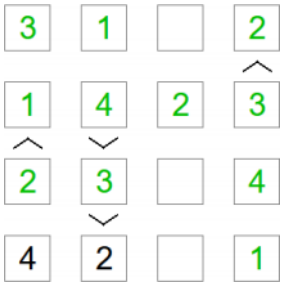
Rozmiar przestrzeni rozwiązań
Tak jak już wcześniej wspomniano, w obu problemach występuje $n \times n$ zmiennych, a dziedzina każdej z nich składa się z n wartości. Oznacza to więc, że rozmiar pełnej przestrzeni rozwiązań wynosi:

$$S = n^{n^2} \tag{1}$$

Rozmiar przestrzeni rośnie w zastraszającym tempie. Dla $n = 5$ jest to już $S \approx 2,98 \cdot 10^{17}$. W oczywisty sposób świadczy to o tym, że szukanie rozwiązań metodą przeglądu zupełnego nie może się sprawdzić dla tej klasy problemów.

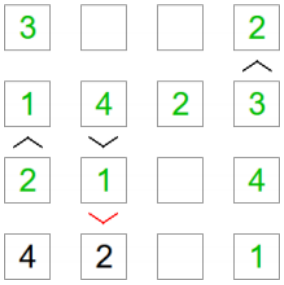
1.2.2 Futoshiki

Futoshiki jest łamigłówką logiczną, w której przykładowa plansza może wyglądać następująco:



Rysunek 1: Przykładowa plansza Futoshiki

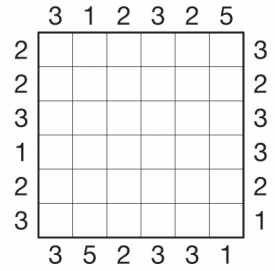
Oprócz wspomnianego już wcześniej ograniczenia nakładanego na kolumny i wiersze, w tym problemie definiuje się jeszcze jedno. Na obrazku 1 jest pokazane jako znaki nierówności pomiędzy dwoma sąsiadującymi polami. Oznacza to, że wartości przypisane tym zmiennym muszą spełniać daną nierówność. Przykład niespełnienia tego typu ograniczenia jest został przedstawiony na rysunku 2.



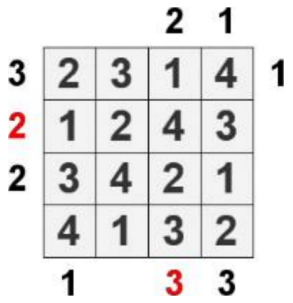
Rysunek 2: Niepoprawnie uzupełniona plansza Futoshiki

1.2.3 Skyscrapper

Skyscrapper jest łamigłówką logiczną, w której wartości przypisywane zmiennym reprezentują wysokość wieżowca stawianego na danym polu. W tym problemie definiuje się dodatkowe ograniczenie, poza tym nakładanym na kolumny i wiersze. Na każdym z boków planszy, obok każdego pola tego boku, może być napisana liczba, mówiąca ile wieżowców powinno być widać w wierszu lub kolumnie, jeżeli spojrzeć na planszę z tego miejsca. Jeżeli wyższy budynek znajduje się przed niższym, to go zasłania. Przykładowa plansza jest przedstawiona na rysunku 3, a przykład niespełnienia ograniczenia na rysunku 4.



Rysunek 3: Przykładowa plansza do łamigłówki Skyscrapper



Rysunek 4: Przykład niespełnienia ograniczenia w łamigłówce Skyscrapper

1.3 Proponowane rozwiązania

1.3.1 Algorytm przeszukiwania przyrostowego z powracaniem (*Backtracking*)

Algorytm ten ma na celu ograniczyć liczbę sprawdzanych rozwiązań. Wybierana jest zmienna, a następnie jest jej przypisywana wartość z dziedziny. Jeżeli po takim przypisaniu żadne ograniczenie nie zostało złamane, to algorytm przechodzi do przypisania wartości następnej zmiennej. Jeżeli w dziedzinie aktualnie rozpatrywanej zmiennej nie istnieje wartość niełamiąca żadnego z ograniczeń, następuje powrót do poprzedniej zmiennej i próba przypisania kolejnej wartości z jej dziedziny.
Zadanie jest rozwiązanie w momencie kiedy wszystkie zmienne mają przypisane wartości i żadne ograniczenie nie jest złamane.

1.3.2 Algorytm sprawdzania w przód (*Forward checking*)

Algorytm ten postępuje podobnie do wspomnianego wcześniej. Różnica jest taka, że po każdym przypisaniu wartości do zmiennej, jeżeli ograniczenia są spełnione, to wykonywany jest dodatkowy krok. Z dziedziny wszystkich zmiennych uwikłanych w ograniczenia z aktualnie rozpatrywaną zmienną, usuwane są wartości, które powodowałyby złamanie jakiegoś ograniczenia. Jeżeli po tym pro-

cesie dziedzina którejkolwiek ze zmiennych jest pusta, to aktualnej zmiennej przypisuje się kolejną wartość z dziedziny.
W znaczący sposób ogranicza to liczbę przeglądanych rozwiązań, ale procedura usuwania wartości z dziedziny zmiennych jest dość złożona, więc nie zawsze przekłada się to na zysk czasowy.

1.3.3 Heurstyki wyboru zmiennych i wartości

Wybór zmiennej do wartościowania
Do wyboru zmiennej do wartościowania została w tej pracy wybrana heurystyka *najmniejszej liczby pozostałych wartości* (*Minimum Remaining Values*). Jako następna zmienna do wartościowania wybierana będzie ta, której pozostało najmniej dostępnych wartości w dziedzinie. Heurystyka ta jest naturalnym uzupełnieniem algorytmu sprawdzania w przód. W przypadku algorytmu przeszukiwania przyrostowego z powrotami generuje ona spory narzut obliczeniowy, ponieważ wymaga wykonania obliczeń, których ten algorytm normalnie nie wykonuje.
Do rozwiązywania remisów (sytuacji gdy zmienne mają taką samą liczbę dostępnych wartości) zostanie użyta heurystyka wybierająca zmienną uwikłaną w najwięcej ograniczeń. Para ta będzie w dalszej części pracy nazywana w skrócie heurystyką *MRV*.
Niestatyczny wybór zmiennych do wartościowania potencjalnie oferuje duży zysk, ponieważ pozwala ustawić zmienne w kolejności generującej zdecydowanie mniejsze drzewo poszukiwań.

Wybór wartości dla zmiennej
Do wyboru następnej wartości dla zmiennej zostanie użyta heurystyka *najmniej ograniczającej wartości*. Wybiera ona dla zmiennej taką wartość, która eliminuje najmniej dopuszczalnych wartości z dziedzin powiązanych zmiennych.
Sformułowanie celu jako znalezienia wszystkich rozwiązań problemu, sprawia że heurystyka ta nie powinna mieć wpływu na liczbę sprawdzonych rozwiązań, ponieważ i tak należy sprawdzić wszystkie dopuszczalne wartości dla każdej ze zmiennych. Rzeczą którą można w tym przypadku badać jest czas i liczba rozwiązań sprawdzonych do momentu znalezienia pierwszego rozwiązania.

1.3.4 Usprawnienia wprowadzone do ograniczeń

Aby zmniejszyć liczbę sprawdzanych rozwiązań i potencjalnie przyspieszyć działanie algorytmów, do sposobu rozpatrywania spełnienia ograniczeń zostały wprowadzone niewielkie zmiany, wynikające wprost z ich logiki.

Futoshiki
Ograniczenie mówiące o nierówności wartości dwóch zmiennych można by rozpatrywać dopiero w momencie przypisania obu z nich. Znając jednak pełną dziedzinę zmiennych, można wyciągnąć następujące wnioski:

- jeżeli do "mniejszej" zmiennej została przypisana największa wartość z dziedziny, to ograniczenie nie jest spełnione,
- jeżeli do "większej" zmiennej została przypisana najmniejsza wartość z dziedziny, to ograniczenie nie jest spełnione.

Skyscrapper
Ograniczenie "widzialności" wieżowców można rozszerzyć w następujący sposób:

- jeżeli do pierwszej zmiennej widzianej, z którejś ze stron została przypisana największa wartość z dziedziny, a z tej strony powinno być widać więcej niż 1 wieżowiec, to ograniczenie nie jest spełnione,
- jeżeli do pierwszej zmiennej widzianej, z którejś ze stron została przypisana najmniejsza wartość z dziedziny, a z tej strony powinno być widać tylko 1 wieżowiec, to ograniczenie nie jest spełnione.

2 Badania

W sekcji badań wykorzystywane będą następujące oznaczenia:

- BT - algorytm przeszukiwania przyrostowego z powrotami (*Backtracking*),
- FC - algorytm sprawdzania w przód (*Forward checking*),
- MRV - wspomniana w sekcji 1.3.3 heurystyka wyboru zmiennej do wartościowania
- VH - wspomniana w sekcji 1.3.3 heurystyka wyboru wartości dla zmiennej
- BTM, FCM - odpowiednio algorytm BT i FC z zastosowaną heurystyką MRV

2.1 Czas działania i liczba węzłów drzewa rozwiązań

2.1.1 Futoshiki

Tabela 1: Liczba węzłów drzewa rozwiązań przegladanych przez algorytmy dla różnych instancji problemu Futoshiki

	BT	**FC**	**BT+MRV**	**FC+MRV**
test_futo_4_0	295	125	27	25
test_futo_4_1	327	161	35	31
test_futo_4_2	172	80	56	49
test_futo_5_0	1361	355	59	45
test_futo_5_1	4664	1711	87	67
test_futo_5_2	352	159	96	82
test_futo_6_0	21120	3811	166	142
test_futo_6_1	86685	23489	214	**187**
test_futo_6_2	161079	56986	679	**532**
test_futo_7_0	35648494	10309677	52583	**40525**
test_futo_7_1	9367837	529143	33591	24181
test_futo_7_2	7616386	2399217	40573	27054
test_futo_8_0	3026907845	1117	**1**	**1**
test_futo_8_1	n/a	458028557	820475	**591495**
test_futo_8_2	77198485	17540373	16445380	12106524
test_futo_9_0	n/a	269559	35787	31290
test_futo_9_1	n/a	n/a	6811876	4977377
test_futo_9_2	n/a	n/a	6541040	4472736

W tabeli 1 przedstawione zostały liczby węzłów drzewa rozwiązań, odwiedzanych przez poszczególne algorytmy, dla różnych instancji problemu Futoshiki. Każdy z opisywanych algorytmów został tam przedstawiony w 2 wersjach. Jedna z nich korzystała z heurystyki MRV do wyboru zmiennej do wartościowania, a druga nie. Pierwsza rzecz, która rzuca się w oczy, to dużo mniejsza liczba przeglądanych przez algorytm FC w stosunku do algorytmu BT. Bardzo ważne jest, aby zauważyć, jak dużą różnicę w liczbie przeglądanych węzłów wprowadza zastosowanie heurystyki MRV. W tabeli pogrubione zostały te miejsca, dla których zysk był największy. Najlepiej widać to najprawdopodobniej na przykładzie pliku test_futo_8_0, gdzie zastosowanie tej heurystyki pozwoliło już w pierwszym kroku stwierdzić, że ten problem nie posiada rozwiązania. Drugim ciekawy przypadek widać dla pliku test_futo_8_1, gdzie zysk był ponad 700-krotny.

Tabela 2: Czasy przetwarzania poszczególnych algorytmów dla różnych instancji problemu Futoshiki

	BT (s)	**FC (s)**	**BT + MRV (s)**	**FC + MRV (s)**
test_futo_4_0	0,0015	0,0007	0,0016	0,0139
test_futo_4_1	0,0004	0,0010	0,0016	0,0051
test_futo_4_2	0,0002	0,0004	0,0020	0,0072
test_futo_5_0	0,0019	0,0020	0,0056	0,0135
test_futo_5_1	0,0053	0,0069	0,0071	0,0049
test_futo_5_2	0,0005	0,0009	0,0073	0,0037
test_futo_6_0	0,0313	0,0201	0,0227	0,0117
test_futo_6_1	0,1050	0,1551	0,0263	0,0067
test_futo_6_2	0,1798	0,3180	0,0899	0,0961
test_futo_7_0	48,5966	62,1622	9,9277	0,6738
test_futo_7_1	12,8833	4,2271	6,7674	0,3085
test_futo_7_2	10,3744	17,0219	8,6324	0,2909
test_futo_8_0	4491,2230	0,1625	0,0010	0,0002
test_futo_8_1	n/a	2813,9495	254,7720	7,7844
test_futo_8_2	116,0373	148,2639	4830,8584	154,3986
test_futo_9_0	n/a	2,1556	10,8396	0,3307
test_futo_9_1	n/a	n/a	2657,8022	63,5044
test_futo_9_2	n/a	n/a	2677,7288	64,6978

W tabeli 2 przedstawione zostały czasy przetwarzania dla różnych algorytmów na różnych instancjach problemu Futoshiki. Można tam zauważyć m.in. to, że gdy po zastosowaniu heurystyki MRV, udało się rozwiązać duże instancje problemów, które nie były możliwe do rozwiązania bez tego. Heurystyka ta bardzo dobrze komponuje się w działanie algorytmu FC. Dzięki temu narzut obliczeniowy nie jest tam aż tak duży i dla większych instancji problemów pozwala to znacznie przyspieszyć ich rozwiązywanie metodą sprawdzania w przód. Jeżeli chodzi o algorytm BT, to widać, że o ile zastosowanie w nim tej heurystyki czasami daje sensowne rezultaty, o tyle narzut obliczeniowy jest ogromny. Co za tym idzie, użycie jej na dużych instancjach problemów, potrafi znacząco spowolnić jego działanie.

2.1.2 Skyscrapper

Tabela 3: Liczba węzłów drzewa rozwiązań przeglądanych przez algorytmy

	BT	**FC**	**BT + MRV**	**FC + MRV**
test_sky_4_0	215	115	110	81
test_sky_4_1	161	77	74	58
test_sky_4_2	304	139	193	164
test_sky_4_3	1270	614	393	311
test_sky_4_4	137	76	82	68
test_sky_5_0	93272	28099	9476	6227
test_sky_5_1	58713	14751	**1816**	**1168**
test_sky_5_2	15713	6139	9867	7053
test_sky_5_3	48598	17191	6393	4455
test_sky_5_4	7193	2538	1069	705
test_sky_6_0	126773	55630	199669	145408
test_sky_6_1	375146	139577	133312	98099
test_sky_6_2	14703689	3389857	**538231**	**402902**
test_sky_6_3	64137671	26275623	6758373	4955527
test_sky_6_4	336873	159064	68576	49537

Tabela 4: Czasy przetwarzania poszczególnych algorytmów dla różnych instancji problemu Skyscrapper

	BT (s)	**FC (s)**	**BT + MRV (s)**	**FC + MRV (s)**
test_sky_4_0	0,0002	0,0005	0,0026	0,0024
test_sky_4_1	0,0001	0,0004	0,0020	0,0009
test_sky_4_2	0,0002	0,0006	0,0042	0,0015
test_sky_4_3	0,0009	0,0024	0,0084	0,0026
test_sky_4_4	0,0001	0,0004	0,0025	0,0006
test_sky_5_0	0,0845	0,1205	0,4130	0,0492
test_sky_5_1	0,0533	0,0569	0,0791	0,0087
test_sky_5_2	0,0142	0,0288	0,4208	0,0523
test_sky_5_3	0,0464	0,0749	0,2787	0,0338
test_sky_5_4	0,0070	0,0113	0,0513	0,0058
test_sky_6_0	0,1405	0,7224	13,7718	0,9643
test_sky_6_1	0,3634	1,0723	8,7668	0,6316
test_sky_6_2	14,4541	16,2330	34,8568	2,5252
test_sky_6_3	66,6152	136,4960	436,4031	32,3194
test_sky_6_4	0,3229	0,8969	5,5211	0,3557

W tabelach 3 i 4 przedstawione zostały wyniki badań szybkości działania algorytmów dla instancji problemów Skyscrapper. Wnioski, które można z nich wyciągnąć w zasadzie nie różnią się od tych dla problemu Futoshiki. Algorytm FC przegląda zdecydowanie mniej węzłów drzewa rozwiązań. Zastosowanie heurystyki MRV, dla obu algorytmów daje znaczne obniżenie liczby odwiedzanych węzłów. Zastosowanie jej razem z algorytmem FC daje bardzo dobre zyski czasowe, ale już przy algorytmie BT, przez dodatkowe obliczenia, nie jest dobrym rozwiązaniem.

2.2 Wpływ heurystyki wyboru wartości

W tej sekcji przedstawione zostały wyniki badań wpływu zastosowania heurystyki wyboru wartości na szybkość znajdowania pierwszego rozwiązania, wyrażoną w liczbie odwiedzonych węzłów i czasie potrzebnym na jego znalezienie.
Badania zostały przeprowadzone pod takim kątem, ponieważ tak jak wspomniano już wcześniej, przy zdefiniowaniu celu problemu jako znalezienia wszystkich istniejących rozwiązań, kolejność wyboru wartości nie ma żadnego wpływu na liczbę odwiedzanych przez algorytmy węzłów. Dzieje się tak, ponieważ aby znaleźć wszystkie rozwiązania, i tak należy przejrzeć wszystkie dopuszczalne w danym węźle wartości.

Tabela 5: Liczba węzłów drzewa rozwiązań przegladanych przez algorytmy do znalezienia pierwszego rozwiązania dla problemów Futoshiki

	BTM	**BTM + VH**	**FCM**	**FCM+ VH**
test_futo_4_0	16	16	16	16
test_futo_4_1	21	**17**	19	**17**
test_futo_4_2	46	46	41	41
test_futo_5_0	59	**57**	**45**	46
test_futo_5_1	78	78	63	63
test_futo_5_2	**29**	45	**27**	42
test_futo_6_0	160	**67**	137	**61**
test_futo_6_1	138	**132**	121	**116**
test_futo_6_2	**149**	317	**124**	260
test_futo_7_0	23030	**20638**	18024	**16180**
test_futo_7_1	**21437**	30681	**15621**	21186
test_futo_7_2	32752	**30553**	21744	**21335**

Tabela 6: Czas przetwarzania do znalezienia pierwszego rozwiązania dla problemów Futoshiki

	BTM (s)	**BTM + VH (s)**	**FCM (s)**	**FCM + VH (s)**
test_futo_4_0	0,0011	0,0012	0,0101	0,0134
test_futo_4_1	0,0009	0,0009	0,0028	0,0032
test_futo_4_2	0,0017	0,0020	0,0065	0,0057
test_futo_5_0	0,0055	0,0068	0,0134	0,0085
test_futo_5_1	0,0062	0,0087	0,0045	0,0051
test_futo_5_2	0,0019	0,0043	0,0009	0,0081
test_futo_6_0	0,0218	0,0110	0,0114	0,0037
test_futo_6_1	0,0172	0,0196	0,0048	0,0064
test_futo_6_2	0,0134	0,0528	0,0255	0,0375
test_futo_7_0	4,2814	5,6408	0,3443	0,4907
test_futo_7_1	4,1773	9,4359	0,1930	0,4368
test_futo_7_2	6,8676	9,9061	0,2321	0,4498

Tabela 7: Liczba węzłów drzewa rozwiązań przegladanych przez algorytmy do znalezienia pierwszego rozwiązania dla problemów Skyscrapper

	BTM	**BTM + VH**	**FCM**	**FCM + VH**
test_sky_6_0	**126192**	131629	**90921**	95082
test_sky_6_1	58130	**50396**	43637	**37344**
test_sky_6_2	**202190**	211993	**151972**	160707
test_sky_6_3	**6279151**	6329223	4606340	**4601170**
test_sky_6_4	**52059**	54246	**37588**	38667

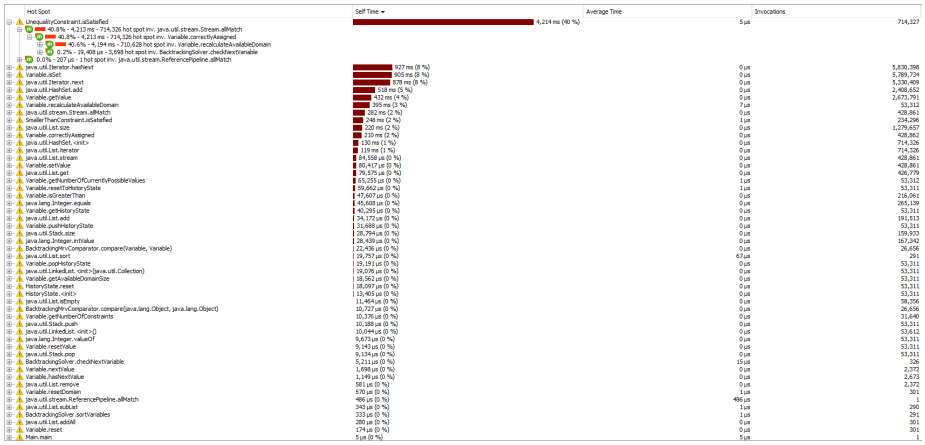
Tabela 8: Czasy przetwarzania do znalezienia pierwszego rozwiązania dla problemów Skyscrapper

	BTM (s)	**BTM + VH (s)**	**FCM (s)**	**FCM + VH (s)**
test_sky_6_0	8,7182	12,4947	0,6015	1,3103
test_sky_6_1	3,8109	4,9899	0,2768	0,4737
test_sky_6_2	12,7111	18,6572	0,9401	1,9689
test_sky_6_3	405,1843	806,1923	30,0207	59,8440
test_sky_6_4	4,1970	6,1803	0,2709	0,5504

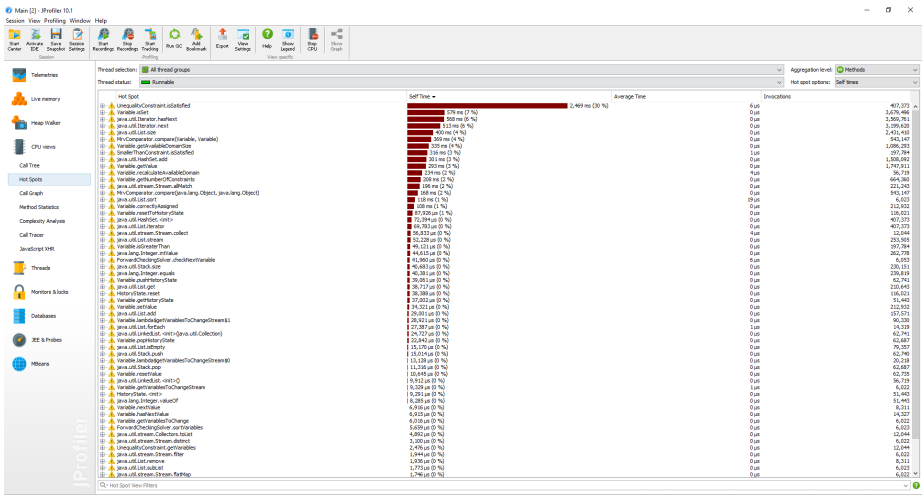
Jak widać we wszystkich tabelach przedstawionych powyżej, skutki zastosowania heurystyki wyboru wartości dla zmiennej nie są tak proste do przewidzenia jak tej odpowiedzialnej za wybór zmiennej. W wynikach widać, że zastosowanie heurystyki potrafi poprawić szybkość znajdowania pierwszego rozwiązania, jak i ją pogorszyć. Zdaje się to być cecha poszczególnych instancji problemu, bardziej niż samej heurystyki.

3 Profilowanie kodu

Aby wyszukać potencjalne problemy w implementacji i usprawnić działanie programu, przeprowadzone zostało profilowanie kodu pod kątem użycia czasu procesora przez poszczególne fragmenty programu.



Rysunek 5: Okno profilera przy uruchomionym algorytmie BT z heurystyką MRV



Rysunek 6: Okno profilera przy uruchomionym algorytmie FC

Z przeprowadzonej analizy wynikło, że najczęściej używanym fragmentem kodu, niezależnie od uruchamianego algorytmu jest metoda UnequalityConstraint.isSatisfied(), więc tam właśnie został położony nacisk na optymalizację.

4 Wnioski

Problemy spełniania ograniczeń, przez swoją bardzo szybko rosnącą przestrzeń rozwiązań stanowią spore wyzwanie obliczeniowe. Algorytmy przeszukiwania przyrostowego z powracaniem i sprawdzania w przód, oferują efektywny sposób przeszukiwania tej przestrzeni dla dowolnego problemu sformułowanego w terminach zmiennych i ograniczeń. Podsumowując, oba zbadane algorytmy wraz z przebadanymi heurystykami, stanowią bardzo dobre, ogólne narzędzia do rozwiązywania złożonych problemów spełniania ograniczeń. Końcowo jednak ze wszystkich zbadanych kombinacji, algorytm sprawdzania w przód z dodatkowym zastosowaniem heurystyki do wyboru zmiennej, wydaje się być najlepszym rozwiązaniem do dużych instancji tego typu problemów.