A* dla układanki "15"

Nel Skowronek

1 Heurystyki

1.1 Manhattan Distance + Linear Conflict

Z drobną modyfikacją: pogorszyłam Linear Conflict, żeby heurystyka była monotoniczna. Zamiast zliczać wszystkie konflikty występujące w danym rzędzie / danej kolumnie, zliczam tylko te, które się "widzą" (*break* po znalezieniu konfliktu).

Przykład działania

Dla rzędu pierwszego od góry takiego stanu:

4	3	8	1
13	0	7	2
11	6	9	10
12	14	15	5

Klasyczny linear conflict: +**6** ((4, 3), (4, 1), (3, 1))
Mój linear conflict: +**4** ((4, 3), (3, 1))

Monotoniczność - dowód

Największym problemem klasycznego Linear Conflict są sytuacje, w których mamy element po srodku wiersza (odpowiednio: kolumny) konfliktujący z przynajmniej dwoma elementami po obydwu jego stronach. - W powyższym przykładzie jest to **3**, która konfliktuje z **4** po lewej i **1** po prawej. - Powoduje to, że przy przesunięciu tego elementu poza wiersz (zamiana **3** z **0**) "znikają" nam aż dwa konflikty, każdy liczony po +**2**. Przy jednoczesnym wzroście

Manhattan Distance o **+1**, w sumie wartość oceny heurystycznej zmalała nam o **-3** względem poprzedniego stanu.

Aby to naprawić (pozwalać na spadek maksymalnie **-1**), korzystam z tego, że konflikty są przechodnie. T.j. jeśli w naszym przykładzie **3** konfliktuje z **4** po lewej i **1** po prawej - to **4** konfliktuje z **1**.

Manhattan Distance jest heurystyką monotoniczną, sytuacja w której element przed przesunięciem ma tylko jeden konflikt też jest trywialna, a więc jedynym momentem gdzie potencjalnie możemy mieć większy spadek oceny heurystycznej jest taki, jak ten opisany powyżej: element jest po środku dwóch konfliktów.

W takiej sytuacji, w momencie kiedy przesuwamy środkowy element poza rząd tracimy 2 konflikty (ponieważ zliczamy tylko "widzące się"), ale jednocześnie z przechodniości wynika, że zyskujemy 1 nowy konflikt. Ponieważ środkowy element został odsunięty od swojego rzędu docelowego, Manhattan Distance wzrośnie o **+1**. W sumie dając nam maksymalny spadek oceny heurystycznej równy **-1** względem poprzedniego stanu.

A więc mój zmodyfikowany Linear Coflict jest monotoniczny.

1.2 Walking Distance

Znana, monotoniczna heurystyka, tym razem bez żadnych modyfikacji. Zlicza liczbę kroków, potrzebną do przesunięcia stanu, w którym wszystkie elementy są w odpowiadającym im docelowych rzędach / kolumnach, do dowolnego innego stanu, nierozróżniając poszczególnych elementów, tylko skupiając się na ich rzędach docelowych.

W celu wyliczenia Walking Distance, zrobiłam precomputing polegający na:

1. Rozpoczęciu od stanu:

	A	B	C	D
A	4	0	0	0
B	0	4	0	0
C	0	0	4	0
D	0	0	0	3

2. Wyliczeniu sąsiadujących stanów
3. Dodawaniu sąsiednich stanów do kolejki przeszukiwania i zapisywaniu ich wraz z obecną odległością
4. Powtarzaniu, aż nie zostaną przeszukane wszystkie stany
5. Zapisaniu do bazy danych (pliku binarnego)

Zauważyłam, że sytuacja dla rzędów i kolumn jest symetryczna, więc precomputing zrobiłam tylko dla rzędów, a w A obliczam stan rzędów i kolumn analogicznie i wyszukuję w bazie danych Walking Distance.

2 Generowanie Permutacji

Do generowania permutacji w C++ używam standardowej funkcji **std::shuffle**. Według dokumentacji generuje permutacje z rozkładu jednostajnego.

3 Działanie

Dla maksymalizacji efektywności stany planszy przechowuję w liczbach 64 bitowych, a większość operacji używanych m.in. przy wyznaczaniu sąsiadów to operacje bitowe (głównie bitshifty, koniunkcje i alternatywy). Surowe stany opakowuję w obiekty **Node** trzymające informacje o własnych wartościach g(), h() i f(), oraz wskaźnik (32 bity) na Node, z którego "przyszedł", co potem bardzo ułatwia śledzenie ścieżki. Node'y do odwiedzenia przetrzymuję w kolejce priorytetowej (insert / pop = O(log n), ocenia priorytet po wartościach f()), a odwiedzone w hash-mapie (insert / find = O(1)).

4 Wykresy

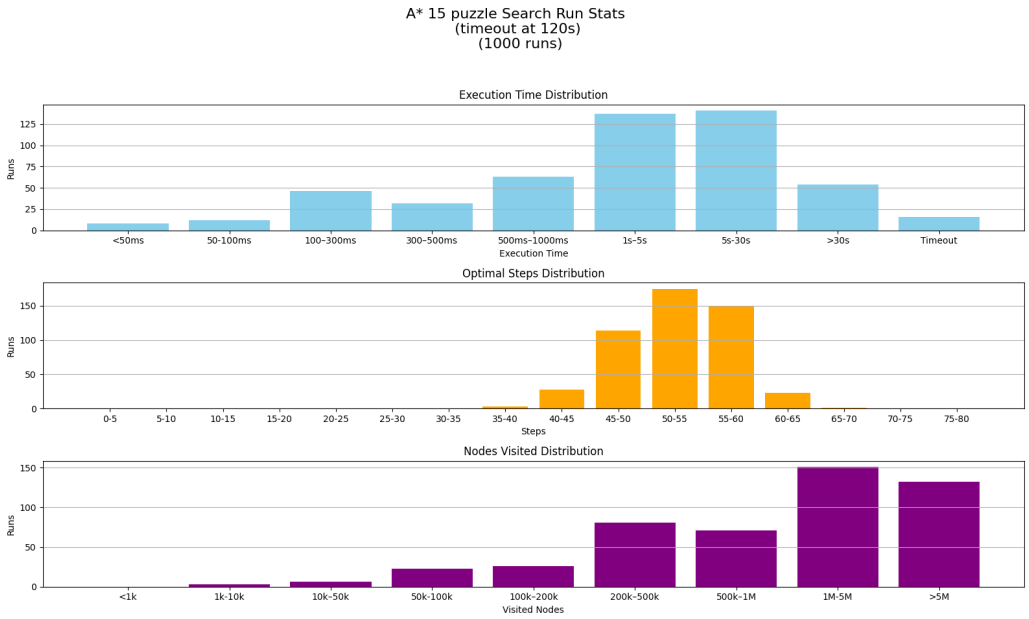

Figure 1: Statystyki dla Manhattan Distance + Linear Conflict

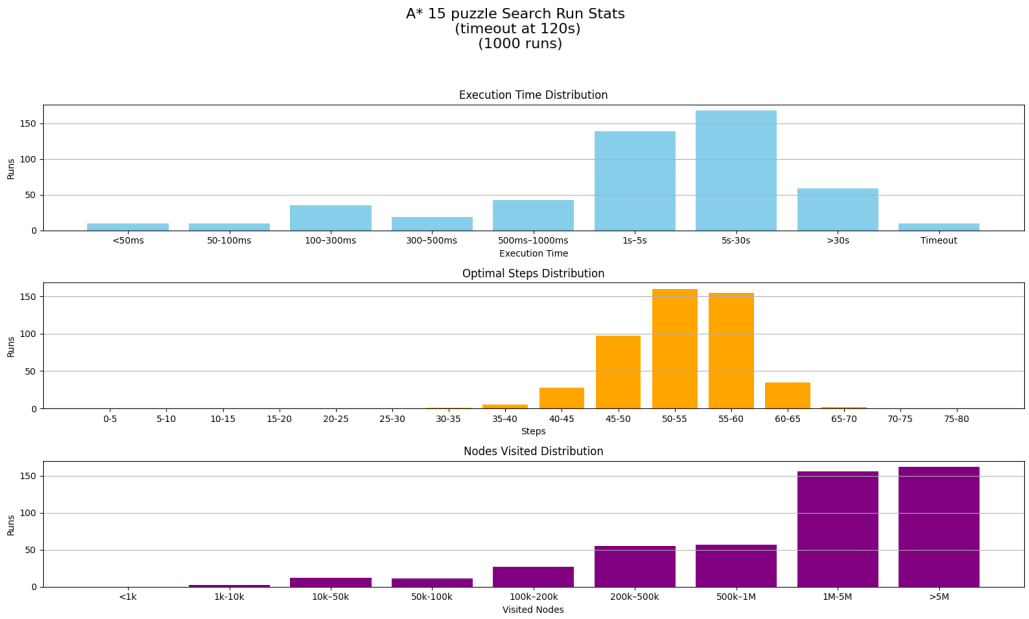

Figure 2: Statystyki dla Walking Distance

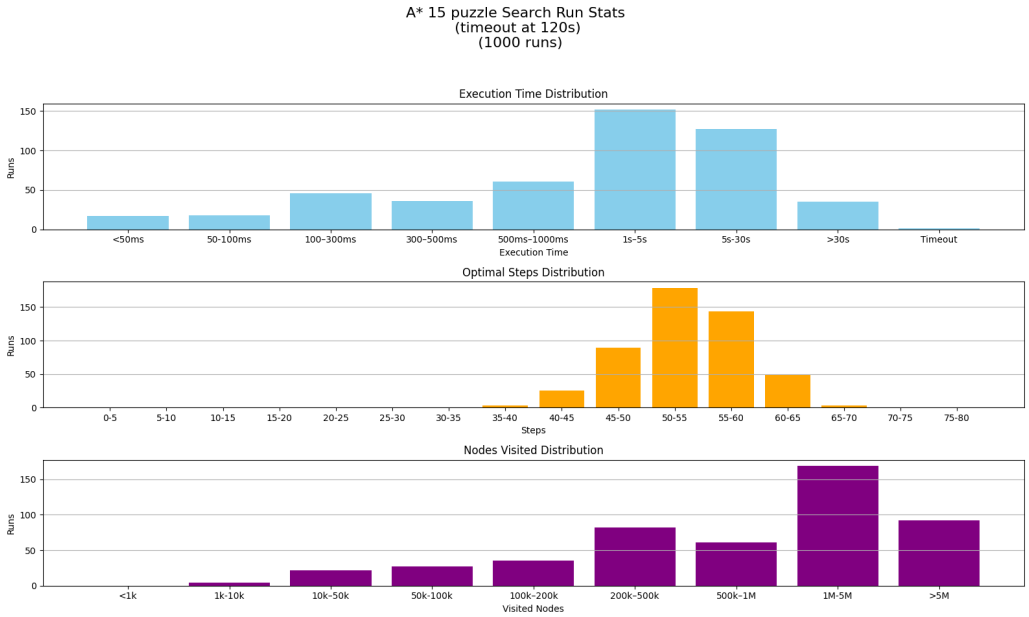

Figure 3: Statystyki dla maksimum z obu heurystyk (najlepsza)