Teoria obliczeń i złożoności obliczeniowej

Projekt

Jakub Madej
Email: jadqba@student.uci.agh.edu.pl

1. Opis algorytmu

Działanie zastosowanego algorytmu można podzielić na trzy fazy:

1. Faza przygotowania

W fazie tej dokonywana jest analiza planszy i eliminacja tych pól, o których wiadomo, że umieszczenie na nich skrzynki uniemożliwi rozwiązanie zadania. W pierwszej kolejności jako blokujące usuwanę są pola znajdujące się w “rogach”, pod warunkiem, że pole to nie jest miejscem docelowym. Oczywiste jest, że umieszczenie skrzynki w takiej pozycji uniemożliwia jej dalszy ruch. Korzystając z tak wyznaczonych pól algorytm oznacza także pola znajdujące się przy ścianach. Należy zauważyć, że dosunięcie skrzynki do ściany, od której nie da się już jej odsunąć także uniemożliwia rozwiązanie. Wyjątkiem od tej reguły jest sytuacja, w której przy takiej ścianie znajduje się pole docelowe.

Aby w fazie rozwiązania móc oceniać jakość poszczególnych ruchów dokonywana jest dalsza analiza planszy gry. Każdej skrzynce przypisywane jest jedno pole docelowe, możliwie najbliższe jej początkowej pozycji. Dla każdej skrzynki allokowana jest tablica reprezentująca planszę i zawierająca odległośći poszczególnych pól od miejsca docelowego. Przez odległość rozumie się najmniejszą ilość kroków potrzebnych magazynierowi do przebycia tej drogi.

2. Faza rozwiązania

W tej fazie następuje właściwe rozwiązanie problermu. Podstawą działania algorytmu jest przeszukiwanie “best first search” (stosowany dalej skót BFS oznacza tan algorytm, a nie “breadth first search”). Jako ruch traktuje się przesunięcie pojedynczej skrzynki o jedną pozycję. Decyzja o tym, którą skrzynkę przesunąć jest podejmowana na podstawie ostatniego ruchu. Jeśli uprzednio ruch skrzynki doprowadził do umieszczenia jej na pozycji docelowej, brana pod uwagę jest następna skrzynka możliwa do poruszenia. Jeżeli ostatni ruch nie doprowadził skrzynki na miejsce docelowe to o ile jest to możliwe przesunięta zostaje ta sama skrzynka. Decyzję o tym, w którym kierunku przesunąć skrzynkę jest podejmowana na podstawie przygotowanych wcześniej tablic odległości. Ruch następuje w kierunku, który najbardziej przybliży skrzynkę do celu, lub najmniej ją oddali. Po podjęciu decyzji o ruchu aktualna sytuacja zostaje zapamiętana na stosie. W przypadku, gdy w danej sytuacji nie da się wykonać żadnego ruchu, poprzednia sytuacja zostaje zdjęta ze stosu i działanie algorytmu jest kontynuowane.

Aby uniknąć cykli, czyli wielokrotnego sprawdzania tych samych stanów, sprawdzone stany przechowuje się, i przy każdym ruchu sprawdza się, czy dany stan był już rozważany. Do przechowywania stanów wykorzystana jest struktura drzewa czerwono-czarnego. Wartość przechowywana w każdym węźle jest jednocześnie jego kluczem. W skład stanu wchodzą pozycje wszystkich skrzynek. Przed zapamiętaniem są one sortowane. Jako, iż pozycję stanowi para liczb, to za mniejszą pozycję przyjmuje się tą o mniejszej odciętej. W przypadku równych odciętych porównuje się rzędne. Kolejną informacją niezbędną do zapamiętania w drzewie jest pozycja magazyniera. Jako że ruchy magazyniera nie są istotne z punktu widzenia algorytmu to niezbędny do zapamiętania jest cały obszar, po którym w danym stanie może poruszać się magazynier. Jest to konieczne, ponieważ przy tych samych ułożeniach skrzyń magazynier może mieć dostęp do różnych obszarów. Jednak dla danego stanu możliwe obszary zawsze będą

rozłączne, co powoduje, że dla jednoznacznej identyfikacji obszaru wystarczy przechowywać jeden punkt. Jednoznaczność wyboru punktu jest osiągana przez pamiętanie punktu o najmniejszej odciętej i rzędnej.

Przy dokonywaniu ruchów brana jest także pod uwagę wiedza o pozycjach blokujących zdobyta podczas fazy przygotowania. Jeżeli jakiś ruch miałby doprowadzić do blokady, to nie jest on wykonywany.

Dodatkowym usprawnieniem jest wykrywanie blokad, w których udział mają nie tylko ściany, ale także inne skrzynki. Należy zauważyć, iż jeżeli jakaś skrzynka nie jest możliwa do poruszenia w pionie lub poziomie to jest ona zablokowana. Jeżeli blokada ta wynika z faktu sąsiedztwa ścian, to można to stwierdzić natychmiast. W przeciwnym wypadku można najczęściej odblokować skrzynkę. Aby to stwierdzić oznacza się badaną skrzynkę jako zablokowaną i rekurencyjnie uruchamia się procedurę sprawdzającą blokadę dla skrzynek sąsiadujących. Jeżeli procedura stwierdzi, że dana skrzynka jest zablokowana i niemożliwa do odblokowania to ruch prowadzący do takiej sytuacji nie jest brany pod uwagę.

3. Faza odtworzenia drogi

W fazie tej odtwarzane są ruchy, jakie są konieczne do wykonania w celu rozwiązania planszy. Po zakończeniu fazy rozwiązywania na stosie znajdują się kolejne ruchy skrzynek, które doprowadziły do rozwiązania. Wychodząc od początkowej pozycji magazyniera odtwarza się najkrótszą drogę, jaką musi on przebyć od swojej aktualnej pozycji do pozycji umożliwiającej przesunięcie skrzyni. Następnie pozycja jest uaktualniana dotychczasową pozycją skrzyni i procedura jest kontynuowana w pętli, do czasu opróżnienia stosu ruchów.

2. Analiza złożoności algorytmu

Załóżmy, że plansze rozwiązywane przez algorytm we właściwej fazie rozwiązania składają się z m pól, na których można umieścić skrzynkę lub magazyniera (ściany nie są wliczane) oraz że na planszy znajduje się n skrzynek. Załóżmy ponadto, że plansza jest pustym prostokątem – jest to przypadek najbardziej pesymistyczny, gdyż takie ułożenie może

doprowadzić do całkowitej bezużyteczności algorytmu wykrywającego we wstępnej fazie pozycje blokujące – wystarczy, że w każdym rogu znajdzie się jedna pozycja docelowa. W tej sytuacji każda skrzynka może zostać umieszczona na jedenej z m pozycji, co daje w konsekwencji odgórne ograniczenie m!/(n!(m-n)!) możliwych stanów. Należy jeszcze uwzględnić fakt, iż z punktu widzenia algorytmu innymi są stany dla których różne są obszary, do których ma dostęp magazynier. Przy odpowiednim ułożeniu skrzynek można utworzyć [m/2]+1 różnych, rozłącznych obszarów. A zatem pesymistyczna ilość możliwcyh do uzyskania stanów wynosi (m!/(n!(m-n)!))*([m/2]+1). Pozostaje jeszcze uwzględnić mechanizm, który na bieżąco próbuje zapobiegać powstawaniu zakleszczeń. Z analizowanego drzewa stanów usunie on te, w których przy ścianie znajdują się w bezpośrednim sąsiedztwie dwie skrzynki. Ponadto usunięte zostaną stany, w których znajdują się zgrupowania czterech skrzynek ułożonych w ten sposób, że każda sąsiaduje z dwiema i żadna nie sąsiaduje bezpośrednio ze ścianą. Możliwych pozycji przy ścianach dla par skrzynek jest dla planszy o wymiarach k na l 2(k-3)+2(l-3), stąd dla rozważanego pesymistycznego przypadku będzie to 2(m-3) pozycji. Daje to wykluczenie 2(m-3)*n*(n-1) stanów. Możliwych pozycji, na których można utworzyć “czwórki” może nie być w ogóle. Rozważania te mają oczywiście sens wyłącznie dla n>1. Sam algorytm BFS w pesymistycznym przypadku nie będzie w niczym lepszy od DFS. Widać zatem, że pomimo wprowadzenia licznych mechanizmów poprawiających wydajność algorytmu, czasowo pozostaje on nadal klasy O([m/2]m!/(n!(m-n)!)).

W celu oszacowania złożoności pamięciowej algorytmu załóżmy, iż ma on do sprawdzenia oszacowaną powyżej pesymistyczną liczbę stanów. Z każdym sprawdzanym stanem wiąże się jego zapamiętanie, aby nie był on sprawdzany ponownie. Jak zaznaczone zostało w opisie algorytmu każda instancja struktury reprezentującej sprawdzony stan przechowuje listę (tablicę) pozycji skrzynek, skutkiem czego w celu przechowania wszystkich sprawdzonych stanów potrzebna jest pamięć rzędu n*((m!/(n!(m-n)!))*([m/2]+1) – 2(m-3)*n*(n-1)). Dodatkowo dużym obciążeniem pamięciowym jest stos, na którym odkładane są niesprawdzone ruchy. W pesymistycznym przypadku może on zawierać nawet tyle stanów, ile drzewo stanów zbadanych (przypadek, w którym do rozwiązania prowadzi tylko jedna droga i jest to jedyna droga, którą z punktu widzenia algorytmu można w ogóle pójść). W praktyce ze względu na przechowywaną tablicę potrzebna jest do tego pamięć rzędu k*l*((m!/(n!(m-n)!))*([m/2]+1) – 2(m-3)*n*(n-1)), co daje złożoność pamięciową O((n+kl)[m/2]m!/(n!(m-n)!)).

Wkład obliczeniowy i pamięciowy wnosi teoretycznie także faza przygotowania i odtworzenia drogi. Faza przygotowania nie wnosi jednak praktycznego wkładu – proste algorytmy tam zastosowane działają w czasie co najwyżej wielomianowym i sprowadzają się do kilkukrotnego przeglądania tablicy reprezentującej planszę gry lub rekurencyjnego oznaczania kolejnych pól. W fazie odtworzenia drogi dokonywana jest jedna iteracja po kolejnych ruchach odłożonych na stos. Złożoność pamięciowa tej opercji jest stała. Złożoność obliczeniowa zależy od ilości ruchów na stosie. Dla każdego ruchu znajdowana jest najkrótsza droga pomiędzy dwoma współrzędnymi na planszy. Procedura ta jest uproszczoną, rekurencyjną wersją algorytmu Dijkstry i ma złożoność $O(m^2)$. W konsekwencji w pesymistycznym przypadku odtworzenie drogi będzie miało złożoność $O(m^2[m/2]m!/(n!(m-n)!))$. Jest to jednocześnie ostateczna pesymistyczna złożoność czasowa całego algorytmu.

Na rozmiar problemu składa się zatem zarówno liczba pól rozwiązywanej planszy (m), wymiary planszy (k, l) oraz liczba skrzynek (n). Oczywiście zachodzi związek: m mniejsze bądź równe (k-2)(l-2). Wymiary planszy nie miałyby znaczenia przy zastosowaniu innej, niezależnej od wymiarów metody przechowywania sprawdzanych stanów.

3. Silne i słabe strony algorytmu

Niewątpliwą zaletą algorytmu jest fakt zastosowania przeszukiwania BFS z oceną jakości ruchu w czasie stałym. Zastosowana funkcja oceny sprawdza się i dla większości plansz takie podejście pozwala poprawić wynik. Kolejną zaletą jest fakt generowania stosunkowo krótkich rozwiązań. Najlepszym kandydatem do następnego ruchu jest skrzynka poruszona w poprzednim. W połączeniu z BFS pozwala to daną skrzynkę przemieścić na miejsce docelowe o ile tylko jest to możliwe, i o ile ruch nie doprowadzi do blokady, której algorytm nie jest w stanie wykryć.

Dla niektórych plansz to co jest zaletą może okazać się jednak wadą. Problemem są plansze, w których miejsca docelowe nie są zgrupowane w jednym miejscu. W początkowej fazie pewnej skrzynce może zostać przypisane miejsce docelowe, które jest dla niej niemożliwe do osiągnięcia nie tyle z powodu rozmieszczenia innych skrzynek, ale budowy planszy (w skrajnym przypadku). W tej sytuacji algorytm będzie usilnie starał się pchać skrzynkę w złym

kierunku. Nie uniemożliwia to oczywiście rozwiązania ale może doprowadzić do jego znacznego wydłużenia. Testy pokazują, iż czas oczekiwania na rozwiązanie może wydłużyć się nawet o 4 rzędy wielkości.

4. Możliwości udoskonalenia algorytmu

Poprawienie ogólnej sprawności algorytmu jest możliwe poprzez zastosowanie lepszej, uwzględniającej aktualne pozycje skrzynek funkcji oceniającej w algorytmie BFS i w konsekwencji byćmoże zamianę tego algorytmu na A*. Nie jest to łatwe, ponieważ każda funkcja będzie poprawiać wyniki na jednych planszach, pogarszając je na innych. Ze względu na to pozostałem przy nieskomplikowanej, ale dającej się obliczyć w stałym czasie funkcji.

Kolejną rzeczą, którą należałoby uwzględnić przy dalszym rozwoju algorytmu jest zmniejszenie zużycia pamięci. W każdym nie do końca sprawdzonym stanie odkładanym na stos przechowywana jest dwuwymiarowa tablica charów reprezentująca planszę gry. Jej przechowywanie nie jest konieczne, pozwala jedynie w nieznacznym, w praktyce niezauważalnym stopniu ograniczyć złożoność czasową. Takie rozwiązanie jest w moim przypadku podyktowane nie do końca przemyślanym podejściem we wstępnej fazie projektowania programu. W obecnym stanie cały główny algorytm rozwiązujący wymagałby modyfikacji w celu uwzględnienia tej poprawki. Ponadto maksymalny czas przewidziany na rozwiązanie planszy ustalony na 120s nie powoduje wyczerpania dostępnej pamięci.

Możliwych, ale już znacznie trudniejszych do zaimplementowania potencjalnych udoskonaleń jest znacznie więcej. Należy tutaj wymienić przede wszystkim zastosowanie algorytmu IDA* zamiast BFS czy A*, wprowadzenie statycznych, wbudowanych w program tablic blokujących sytuacji, rozwinięcie algorytmu do wykrywania sytuacji blokujących (np. poprzez zapamiętywanie raz wykrytych zakleszczeń), a także uwzględnienie pewnych możliwych do wykorzystania elementów plansz, na przykład tuneli (szczególnie złośliwe dla algorytmu mogą być ślepe tunele).