유전 알고리즘을 이용한 정렬 알고리즘의 최악의 인스턴스 탐색

전소영○, 김용혁
광운대학교 컴퓨터소프트웨어학과
jesoyo6316@hanmail.net, yhdfly@kw.ac.kr

Finding the Worst-case Instances of Some Sorting Algorithms Using Genetic Algorithms

So-Yeong Jeon○ and Yong-Hyuk Kim
Department of Computer Science and Engineering, Kwangwoon University

요 약

정렬 알고리즘에서 사용한 원소 간 비교횟수를 기준으로, 비교횟수가 많게 되는 순열을 최악의 인스턴스(worst-case instance)라 명명하고 이를 찾기 위해 유전 알고리즘(genetic algorithm)을 사용하였다. 잘 알려진 퀵 정렬(quick sort), 머지 정렬(merge sort), 힙 정렬(heap sort), 삽입 정렬(insertion sort), 쉘 정렬(shell sort), 개선된 퀵 정렬(advanced quick sort)에 대해서 실험하였다. 머지 정렬과 삽입 정렬에 대해 탐색한 인스턴스는 최악의 인스턴스에 거의 근접하였다. 퀵 정렬은 크기가 증가함에 따라 최악의 인스턴스 탐색이 어려웠다. 나머지 정렬에 대해서 찾은 인스턴스는 최악의 인스턴스인지 이론적으로 보장할 수 없지만, 임의의 1,000개 순열을 정렬해서 얻은 비교횟수들의 평균치보다는 훨씬 높았다. 본 논문의 최악의 인스턴스를 탐색하는 시도는 알고리즘의 성능 검증을 위한 테스트 데이터를 생성한다는 점에서 의미가 크다.

키워드 : 최악의 경우(worst-case), 인스턴스(instance), 정렬 알고리즘(sorting algorithm), 테스트 데이터 생성(test data generation), 유전 알고리즘(genetic algorithm)

1. 서 론

테스트 데이터를 만드는 것은 중요한 일이지만 수동으로 만드는 것은 많은 시간과 비용이 소모된다. 테스트 데이터를 자동으로 생성하는 문제는 일반적으로 결정 불능 문제(undecidable problem)이기 때문에 다루기 쉽지 않다. 이를 위해 메타 휴리스틱 탐색 기법(metaheuristic search technique)을 사용할 수 있는데, McMinn[1]은 이러한 기법을 중심으로 테스트 데이터 생성의 연구들을 정리했다. 알고리즘의 테스트 데이터는 입력 인스턴스가 되는데, 알고리즘의 경우는 성능을 분석하는 일이 특히 중요하므로, 그것을 위해 나름의 기준을 세워서 최악의 인스턴스를 찾는 시도가 의미있다고 본다. 그러나 저자가 아는 한 자동으로 최악의 인스턴스를 찾는 연구는 현재까지 거의 없는 것으로 보인다.

이 연구에서는 정렬 알고리즘을 대상으로, 정렬을 할 때 일어나는 원소 간의 비교횟수를 기준으로 최악의 인스턴스를 찾고자한다. 이를 위해, 메타휴리스틱 탐색 기법의 하나인 유전 알고리즘을 사용하고 그 결과를 논의할 것이다.

다음 2절에서 유전 알고리즘 및 실험 대상의 정렬 알고리즘들을 소개하고, 3절에서는 사용한 유전 알고리즘의 구현을 소개하며, 4절에서는 실험 계획 및 결과, 5절에서는 본 연구에 대한 논의와 향후 연구 방향을 제시한다.

2. 유전 알고리즘과 정렬 알고리즘 소개

2.1. 유전 알고리즘 소개

유전 알고리즘은 비결정론적 알고리즘(nondeterministic algorithm)이다. 비결정론적 알고리즘의 구성은 문제의 해를 '추측'하는 단계와 그 해가 맞는지 '검증'하는 단계로 되어있다. 여기서 추측한다는 것은, 임의로 해를 만들 수도 있고, 정해진 규칙을 가지고 해를 만드는 것일 수도 있다. 유전 알고리즘이 해를 추측하는 과정은 생물 진화의 원리를 이용한다. 유전 알고리즘은 복수의 해들을 집단으로 운영하면서 그것의 검증 단계를 거친다(품질 및 적합도 평가). 그 결과와 몇 가지 기준을 토대로, 몇 개의 해를 선발하여(선택) 미세한 변화를 가하거나(변이) 그들 중 두 개를 서로 조립해서(교차) 새로운 해를 만든다. 새로운 해들은 어떤 기준에 따라서, 기존의 해집단의 일부 혹은 거의 전체와 교체된다(대치).

또한, 유전 알고리즘에는 대개 최적화 과정도 포함된다. 어떤 해에 대한 지역 최적화란, 탐색을 수행할 때 주변의 해들 중에서 가장 품질이 높은 해로 이동시키는 것을 말한다. 지역 최적화 과정을 포함한 유전 알고리즘을 미미틱 유전 알고리즘[2](memetic genetic algorithm)이라고 한다.

유전 알고리즘에서 품질 및 적합도 평가, 선택, 변이, 교차, 대치를 위한 알고리즘과 지역 최적화 알고리즘은 해를 프로그램 상에서 인코딩하는 방법과 문제의 특성

에 따라서 다양하다.

2.2. 정렬 알고리즘 소개

다음은 실험 대상의 정렬 알고리즘들이다.

- 퀵 정렬 (quick sort)
- 머지 정렬 (merge sort)
- 힙 정렬 (heap sort)
- 삽입 정렬 (insertion sort)
- 쉘 정렬 (shell sort)
- 개선된 퀵 정렬 (advanced quick sort)

퀵 정렬[3]은 분할-정복(divide-and-conquer)기법을 사용한 정렬 알고리즘이다. 정렬할 배열을 두 개의 분할(partition)로 나누고 각 분할에 대해서 다시 두 개의 분할로 나누는 작업을 반복한다. 본 논문의 구현에서는 피봇(pivot)을 나누게 될 배열의 첫 번째 원소로 삼는다. 나누어진 한쪽 분할의 원소들은 피봇보다 크고 반대쪽 분할의 원소들은 피봇보다 작다. 원소들 간의 비교는 두 개의 분할로 나눌 때 하게 된다.

머지 정렬[3]도 분할-정복의 기법을 사용한다. 머지정렬에서는 정렬할 배열을 분할 간의 대소관계없이 나누지만, 분할들을 역으로 합쳐갈 때는 합쳐진 분할이 정렬되도록 한다. 원소들 간의 비교는 분할이 합쳐질 때 하게 된다.

힙 정렬[3]은 원소를 힙(heap)이라는 자료구조에 넣는다. 본 논문의 구현에서는 힙을 배열로 구현하였다. 정렬할 원소들의 배열은 힙을 나타내는 영역과 정렬을 유지하고 있는 영역으로 구성된다. 힙에서 부모 노드가 자식 노드보다 항상 크게 유지하며, 루트에 있는 원소를 정렬을 유지하고 있는 영역의 앞에 더하고 다시 힙의 조건에 맞게 힙을 수리(repair)하는 과정을 반복한다. 원소 간의 비교는 힙과 관련이 있다. 즉, 처음에 힙을 구성할 때와 중간에 힙을 수리할 때 원소들 간에 비교가 필요하다.

삽입 정렬[3]은 정렬할 원소들의 배열을 정렬된 영역과 그렇지 않은 영역으로 나눈다. 처음에는 배열 전체가 정렬되지 않은 부분이다. 정렬되지 않은 부분에서 원소를 빼서 정렬된 부분의 적절한 위치에 끼워 넣는다. 적절한 위치를 찾기 위해서 정렬된 영역의 원소들과 비교를 하게 된다.

쉘 정렬[4]은 삽입 정렬을 응용한 것이다. 쉘 정렬은 처음부터 모든 원소를 대상으로 삽입 정렬을 하지 않고, 일정한 간격만큼 떨어진 원소들끼리만 서로 비교해서 삽입정렬을 수행한다. 이것은 배열을 몇 개 영역으로 나누어서 각각 정렬을 하는 셈이 된다. 각 영역에 대해 정렬이 끝나게 되면, 간격을 좀더 줄여서 다시 수행하게 된다. 매 단계의 간격 크기를 나열한 것을 간격 수열(gap sequence)이라고 부르며 이 간격 수열은 다양하게 택할 수 있다[5]. 간격 수열의 마지막 값은 항상 1이 되어야 하고, 이 때는 단지 보통의 삽입 정렬을 수행하는 것과 같다. 각각의 삽입 정렬에서 원소 간의 비교가 필요하다.

개선된 퀵 정렬[3]은 피봇을 택하는 과정을 개선한 것이다. 본 논문의 구현에서는 나누어야 할 배열(혹은 전 단계의 분할)에서 임의로 3개의 원소를 골라 대소를 비교한 결과, 그 중 중간값을 피봇으로 삼았다. 개선된 퀵정렬에서는 임의로 원소를 고르는 과정때문에 비교횟수가 일정하게 나오지 않으므로, 본 논문에서는 50번 정렬했을 때 얻은 비교횟수의 평균값을 기준으로 최악의 인스턴스를 찾았다. 원소들 간의 비교는 퀵 정렬의 경우와 같다.

3. 사용한 유전 알고리즘 및 파라미터

3.1. 유전 알고리즘의 구조

```
//P: 해집단
//R: 해집단에 들어갈 새로운 해의 집합
P 생성;
repeat{
    P의 각 해에 대해 적합도 계산;
    repeat{
        P에서 p1 선택;
        P에서 p2 선택;
        c1 ← p1과 p2의 교차;
        c2 ← p1의 변이와 지역최적화;
        c3 ← p2의 변이와 지역최적화;
        R에 c1, c2, c3 추가;
    }until(대치 해야할 개수 > R의 크기)
    P의 일부를 R에 있는 해들로 대치;
    R을 비움;
}until(종료 조건이 만족)
```

그림 1. 사용한 유전 알고리즘의 구조

사용한 유전 알고리즘의 구조는 그림 1과 같다. 처음 해집단 P 를 생성하는 법은 3.2절에서 설명한다. 그 다음은 종료조건을 만족할 때까지 새로운 해들을 생성해서 대치하는 일을 반복하는 데, P 에서 두 개의 해 p_1, p_2를 선택해서 p_1과 p_2의 교차, p_1의 변이 그리고 p_2의 변이를 통해 3개의 새로운 해를 얻는다. 새로이 얻은 해들의 개수가 지정한 대치 수보다 작은 경우 위의 과정을 반복한다.

3.2. 초기 해집단 생성

해의 범위는 1부터 n까지의 자연수가 한 번씩 나오는 순열(permutation)로 제한하며, 처음 해집단의 해들은 무작위 생성된다. 예를 들어, 크기 10인 순열 하나를 무작위 생성한다는 것은 1부터 10까지의 숫자를 한 번씩만 써서 임의로 나열하는 것이다.

3.3. 품질 및 적합도 평가 그리고 선택

자연수의 대소를 비교연산자로 사용하여 오름차순으로 정렬하려고 할 때 사용한 비교횟수가 해의 품질이 된다. 다만, 개선된 퀵 정렬의 경우에는 정렬을 독립적으로 몇 번 반복 후 얻은 비교횟수들의 평균값을 품질로 사용하였다.

어떤 해의 적합도는 해집단으로부터 그 해를 선택할 때 참고하는 값이며, 이 값에는 그 해의 품질도 반영된다. 여기서는 공유 기반 적합도 계산[2]을 이용하였다. 그림 2는 적합도 계산식을 보여준다. 선택은 품질비례 룰렛휠 선택방법[2]를 사용하였다.

$$F_i = \frac{f_i}{\sum_{j=1}^{n} s(d_{i,j})}$$

F_i : 공유를 고려한 해집단의 i번째 해의 적합도
n : 해집단의 크기

$$s(d_{i,j}) = 1 - \frac{d_{i,j}}{\sigma}$$

$d_{i,j}$: i번째 해와 j번째 해의 거리
여기서는 두 수열의 같은 인덱스에서,
원소값의 차의 절대값을 전부 더한 값으로 삼았다.
σ : 해집단에서 가장 차이가 많이 나는 두 해의 차이

$$f_i = (C_i - C_w) + (C_b - C_w)/(k-1),\ k > 1$$

C_i : i번째 해의 품질
C_w : 해집단에서 가장 나쁜 해의 품질(가장 낮은 값)
C_b : 해집단에서 가장 좋은 해의 품질(가장 높은 값)
k : 선택압

그림 2. 적합도 계산 식

3.4. 교차, 변이, 지역최적화, 대치 그리고 종료조건

교차는 PMX[5]를 사용하였다. 한편, 변이는 해가 순열인 점을 이용하여, 순열의 각 인덱스를 차례대로 보고 정해진 변이 확률로 해당 인덱스의 원소를 다른 임의의 인덱스의 원소와 자리 바꾸는 방법을 쓴다.

지역최적화는 해가 나타내는 순열에서 한 쌍의 원소를 골라서 자리를 바꾸고, 품질의 개선이 있으면 이제 그 순열에서 다시 한 쌍의 원소를 택해 자리를 바꾸는 작업을 시도하는 것을 계속한다. 더이상 품질의 개선이 없으면 지역최적화를 끝낸다.

한편, 교차와 변이로 만들어진 새로운 해들은 기존 해집단에서 품질이 가장 나쁜 것들과 교체된다. 새로운 해집단은 바로 전 해집단을 n 세대라고 부를 때, $n+1$ 세대라고 한다. 초기 해집단은 0 혹은 1 세대로 볼 수 있지만, 여기서는 0 세대로 간주하였다. 종료조건은 해집단이 지정한 최대세대수에 도달하는 것이며, 도중에 해집단의 모든 해가 같아지게 되어도 종료조건이 성립된다.

4. 실험 계획 및 결과

우선 실험에 사용했던 컴퓨터의 CPU 사양과 사용한 파라미터 값들을 정리한다(표 2).

표 2. CPU 사양 및 파라미터

CPU 사양	Intel Core2 Duo T8100 2.10Ghz
최대 세대 수	1,000
해집단의 크기 (해의 개수)	100
선택압	3
변이 확률	0.15
대치 수	30
유전 알고리즘 반복 횟수	50

실험은 순열의 크기 n과, 해의 품질을 평가하는데 쓰는 정렬 알고리즘에 따라 분류된다. 크기는 10부터 40까지 10씩 증가한 값들이다. 단, 개선된 퀵 정렬은 20까지만 실험한다. 유전 알고리즘은 분류된 각 항목에 대해 1,000 세대까지 수행하며, 초기 해집단 생성이나 변이에 임의성이 있으므로 독립적으로 50번 반복된다. 이때 각 1,000 세대째의 해집단으로부터 가장 품질이 우수한 해(elite)를 추출하여 얻은 50개의 품질값(비교횟수)을 가지고 최악(비교횟수가 제일 많은 경우), 평균, 표준편차, 오름차순과 내림차순으로 나열된 순열의 정렬 시 비교횟수, 그리고 이론적인 최악의 비교횟수를 구해 정리하였다(표 3, 5, 7, 9). 또한 각 크기와 정렬 알고리즘에 따른 최악의 순열도 구해보았다(표 4, 6, 8, 10).

표 3. 크기 10일 때 비교횟수

정렬 방식	최악	평균	표준편차	오름차순	내림차순	이론적 최악
퀵	63	63.00	0.00	63	63	63
머지	25	25.00	0.00	15	19	25
힙	44	44.00	0.00	41	35	N/A
삽입	45	45.00	0.00	9	45	45
쉘	37	37.00	0.00	15	21	N/A
개선된 퀵	59.84	59.51	0.10	44.29	52.32	N/A

표 4. 크기 10일 때 최악의 순열

정렬 방식	최악의 순열
퀵	[10 2 6 5 4 3 1 7 8 9]
머지	[1 9 3 7 2 5 8 6 10 4]
힙	[7 8 3 5 4 2 1 9 10 6]
삽입	[10 9 8 7 5 6 4 3 1 2]
쉘	[6 2 10 7 4 3 9 8 5 1]
개선된 퀵	[7 5 6 8 10 9 4 1 2 3] 1,000회 평균 비교횟수: 58.442

표 5. 크기 20일 때 비교횟수

정렬 방식	최악	평균	표준 편차	오름-차순	내림-차순	이론적 최악
퀵	228	228.00	0.00	228	228	228
머지	69	69.00	0.00	40	48	69
힙	129	129.00	0.00	121	105	N/A
삽입	190	190.00	0.00	19	190	190
쉘	127	127.00	0.00	42	60	N/A
개선된 퀵	149.16	148.10	0.32	110.45	127.84	N/A

표 6. 크기 20일 때 최악의 순열

정렬 방식	최악의 순열
퀵	[1 20 19 4 16 15 14 13 11 10 9 8 12 7 6 5 3 17 18 2]
머지	[16 13 12 17 10 4 20 15 5 1 14 8 9 7 18 3 11 19 2 6]
힙	[5 16 13 18 10 8 15 19 2 6 11 3 1 7 4 14 9 20 17 12]
삽입	[20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1]
쉘	[19 11 4 6 14 9 3 7 18 15 8 5 17 20 12 1 16 13 10 2]
개선된 퀵	[12 13 14 10 11 15 16 19 20 17 18 8 6 3 2 1 9 7 5 4] 1,000회 평균 비교횟수: 147.283

표 7. 크기 30일 때 비교횟수

정렬 방식	최악	평균	표준 편차	오름-차순	내림-차순	이론적 최악
퀵	493	493.00	0.00	493	493	493
머지	119	119.00	0.00	71	77	119
힙	225	225.00	0.00	211	188	N/A
삽입	435	435.00	0.00	29	435	435
쉘	251	251.00	0.00	72	115	N/A

퀵 정렬의 비교횟수는 2절에서 제시한 구현을 기준으로 할 때, 정렬된 상태 혹은 역으로 정렬된 상태에서 가장 많다. 삽입정렬의 경우 역으로 정렬된 상태에서 비교횟수가 가장 많다. 2.2절에서 머지정렬은 분할을 합칠 때 비교가 일어난다고 하였다. 분할을 합칠 때, 가장 많은 비교횟수는 '합쳐진 분할의 크기값-1'이다. 예를 들어 합쳐진 분할이 1,2,...,10이라고 하자. 그러면 합쳐야 할 분할 중 하나가 1,3,5,7,9이고 나머지 하나가 2,4,6,8,10이라고 한다면 비교를 가장 많이 하게 된다(다른 유형도 가능하다). 합치는 과정에서 '합쳐진 분할의 크기값-1'을 전부 다 더하면 그것이 머지 정렬의 최악의 비교횟수가 된다. 다만, 같은 머지정렬이라도 사용한

표 8. 크기 30일 때 최악의 순열

정렬 방식	최악의 순열
퀵	[30 2 28 25 23 6 22 18 15 14 11 12 10 13 9 8 16 17 7 19 20 21 5 4 24 3 26 27 1 29]
머지	[28 5 2 17 27 18 12 21 15 24 1 6 29 4 8 20 19 11 3 9 26 7 22 13 30 10 23 16 25 14]
힙	[10 24 7 19 25 27 6 21 18 20 2 15 28 5 3 12 9 4 17 22 30 26 23 11 13 8 16 14 29 1]
삽입	[30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 1 2]
쉘	[22 25 3 1 13 21 30 5 17 23 29 2 11 19 28 6 15 14 24 27 7 12 20 26 8 16 10 18 4 9]

표 9. 크기 40일 때 비교횟수

정렬 방식	최악	평균	표준 편차	오름-차순	내림-차순	이론적 최악
퀵	858	838.92	23.05	858	858	858
머지	177	177.00	0.00	100	116	177
힙	342	342.00	0.00	321	278	N/A
삽입	780	780.00	0.00	39	780	780
쉘	399	399.00	0.00	102	201	N/A

표 10. 크기 40일 때 최악의 순열

정렬 방식	최악의 순열
퀵	[40 2 3 29 22 21 20 19 16 15 10 11 12 13 14 9 8 17 18 7 6 5 4 23 24 25 26 27 28 1 30 31 32 33 34 35 36 37 38 39]
머지	[18 8 9 25 7 37 14 19 15 4 3 27 35 2 1 6 40 34 26 24 12 33 30 36 5 11 39 32 31 29 38 13 23 16 17 28 20 22 10 21]
힙	[18 23 27 13 20 5 3 34 38 17 11 31 8 30 7 32 35 40 37 22 12 21 24 4 28 16 29 15 1 14 6 36 26 39 33 19 9 10 2 25]
삽입	[40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 1 2]
쉘	[31 18 40 3 8 33 39 5 22 19 38 12 21 10 34 30 14 24 27 29 13 9 32 35 2 7 25 23 37 6 17 20 36 1 16 26 28 4 11 15]

구현에서 홀수 크기의 순열을 분할할 때 어떻게 하는지에 따라서 달라질 수 있음에 주의하자. 본 논문의 머지 정렬 구현에서는 홀수 크기의 분할은 왼쪽(낮은 인덱스)에, 짝수 크기의 분할은 오른쪽(높은 인덱스)에 두었다. 앞에서 언급한 정렬들의 경우 실험에서 구한 최악의 비교횟수와 이론적 최악의 비교횟수가 같음을 표 3, 5, 7, 9에서 확인할 수 있다.

나머지 정렬에 대해서는 실험에서 찾은 인스턴스가 최악임을 이론적으로 보장하지 않지만, 일반적인 순열보다 비교횟수가 많은 것을 알 수 있다(표 11). 표 12은 실험한 각 항목에 대해 유전 알고리즘을 1번 수행했을 때 걸린 시간을 나타낸다.

표 11. 각 크기별 임의의 1,000개 순열의 평균 비교횟수

크기	퀵	머지	힙	삽입	쉘	개선된 퀵
10	45.72	22.74	38.65	29.81	25.40	53.95
20	122.49	63.52	114.54	111.71	76.41	137.43
30	212.61	111.66	203.52	242.64	139.64	N/A
40	310.77	165.18	306.84	423.35	209.96	

표 12. 유전 알고리즘의 1회 수행 시 걸린 시간(초)

크기	퀵	머지	힙	삽입	쉘	개선된 퀵
10	16	20	19	19	17	1352
20	75	122	123	134	85	12,502
30	214	415	414	496	282	N/A
40	456	1,059	1,093	1,359	729	

5. 결 론

본 연구는 알고리즘의 최악의 인스턴스를 찾는 시도 중의 하나로서, 잘 알려진 몇 가지 정렬알고리즘에 대해 각각, 원소 간 비교횟수가 제일 많은 순열을 찾아보려고 했다. 찾은 순열의 비교횟수는 임의의 1,000개 순열의 평균 비교횟수보다 많았다. 몇몇 정렬 알고리즘에 대한 결과가 실제로 최악의 순열인지 검증되지 않은 점에서 한계가 있었지만, 실험 결과를 볼 때, 이러한 시도가 가능성 있음을 보여준다.

다른 메타휴리스틱 기법 혹은 보다 개선된 유전 알고리즘으로 최악의 순열을 탐색하는 능력을 개선하는 것을 앞으로의 연구 주제로 남긴다. 물론 정렬 알고리즘을 제외한 다른 알고리즘을 대상으로 연구하는 것도 흥미 있을 것이다.

참 고 문 헌

[1] P. McMinn, "Search-based software test data generation: a survey", *Software Testing Verification And Reliablity*, Vol. 14, No. 2, pp. 105-156, 2004.
[2] 문병로, *쉽게 배우는 유전 알고리즘-진화적 접근법*, 한빛미디어, 2008.
[3] M. Main and W. Savitch, *Data Structures and Other Objects Using C++*, 3rd Edition, Addison Wesley, 2004.
[4] http://www.cs.utexas.edu/users/lavender/courses/ee360c/lectures/lecture-21.pdf
[5] http://en.wikipedia.org/wiki/Shell_sort
[6] http://www.rubicite.com/Genetic/tutorial/crossover5.php