한국시뮬레이션학회 '99춘계학술대회 논문집 1999.4.17 동의대학교

실시간 시뮬레이션을 위한 최적 검색 기법 연구

A Study on Real-time Searching Algorithms

윤 석 준, 강 현 주

Sugjoon Yoon, Hyounjoo Kang

세종대학교 항공우주공학과, 응용통계학과

서울특별시 광진구 군자동

(Email: sjyoon@sejong.ac.kr, Fax: 02-3408-3333, Tel: 02-3408-3283)

Abstract

항공기, 자동차, 전차 등 차량의 시스템 또는 운동의 시뮬레이션에서는 다양한 종류의 불연속적인 파라미터 값들이 데이터 테이블의 형태로 주어지게 되는데, 본 연구에서의 관심은 적분 스텝의 크기가 고정되는 일반적인 실시간 시뮬레이션의 제약 하에서 기존의 검색 기법들을 비교하고, 최적의 검색 기법을 개발하는 것이다. 본 연구에서는 전통적으로 수치해석, 시뮬레이션, 데이터베이스 등 다양한 학문 및 응용기술 분야에서 개발된 데이터 검색 기법들을 조사하여 비교하였다. 또한, 다양한 크기와 형태의 데이터 테이블들을 사용하여 수치비교시험을 수행하였는데, 그 경향은 이론에 근거한 예상과 대체로 일치하였다. 한편, 검색하고자 하는 파라미터 값이 임의의 dynamics를 갖고 변한다면, 이러한 정보를 이용하여 주어진 데이터 테이블 내의 검색 영역을 축소 시켜 검색속도를 향상시킬 수가 있다. 다양한 수치시험에서 이분 검색법(bisection method)은 축소된 테이블의 크기에만 영향을 받지만 보간 검색법(interpolation method)과 그 변형 기법들은 검색 대상 테이블의 축소로 데이터의 형태가 직선형이 되는 효과를 얻기 때문에 검색속도를 단축시키는데 매우 탁월한 효과를 나타내었다. 결론적으로 dynamic-window개념을 도입한 보간 검색법과 그 변형들이 이론적으로도 실험적으로도 최적의 검색 속도를 보장함이 입증되었다.

1. 서 론

항공기, 자동차, 전차 등 차량의 시스템 또는 운동의 시뮬레이션에서는 다양한 종류의 불연속적인 파라미터 값들이 데이터 테이블의 형태로 주어지게 되는데, 주로 미분방정식으로 정의되는 수학적 모델들을 시뮬레이션하기 위하여 매 적분 스텝마다 데이터 테이블을 이용하여 필요한 파라미터들의 함수 값들을 산출하기 위한 소위 함수발생(function generation) 작업들이 수행된다. 이 함수발생[1]은 검색과 보간 과정으로 분리될 수 있는데, 본 연구에서의 관심은 적분 스텝의 크기가 고정되는 일반적인 실시간 시뮬레이션의 제약 하에서 기존의 검색 기법들을 비교하고 최적의 검색 기법을 개발하는 것이다. 본 연구에서는 전통적으로 수치해석, 시뮬레이션, 데이터베이스 등 다양한 학문 및 응용기술 분야에서 개발된 데이터 검색 기법들을 조사하여 비교하였으며, 그 기준은 검색 횟수의 기대치 (expected value)와 최대값(worst case)이다. 이 두 가지 중에서도 하드웨어로의 입출력이 동반되는 동적 시스템의 실시간 시뮬레이션에서는 고정 적분 프레임을 주로 사용하기 때문에 최대값이 주 관심의 대상이 된다.

실시간 시뮬레이션에서 사용되는 대부분의 데이터들은 불연속적이면서 등 간격이 아니다. 또한 검색하게 되는 파라미터 값들이 데이터 테이블 내에 바로 존재하지 않는다. 따라서, 불연속적으로 저장된 키(key) 값들이 아니라 파라미터 값을 포함하는 단위 구간에 대한 검색을 수행해야 한다. 실시간 시뮬레이션에서는 시뮬레이션 시작 이전에 off-line으로 데이터의 정렬이 가능하기 때문에 정렬된 데이터를 대상으로 하는 검색 방법들만이 본 비교연구의 대상이 된다. 데이터 베이스 분야에서는 hashing function[2]을 이용한 검색 방법이나, block search[3], Fibonacci search[3]등의 매우 효과적인 검색 방법들이 데이터 테이블의 형태에 따라 채택되고 있다. 하지만, 이들은 검색속도가 구간을 대상으로 검색하기에는 적합하지 않은 기법들이다. 이런 이유들로 이분 검색(bisection search)[4], 보간 검색(interpolation search)[5], fast search[6] 등의 방법들이 실시간 시뮬레이션 특히 HILS(Hardware-in-the-Loop Simulation) 분야에서 사용되어왔다. 데이터를 검색하는 방법은 근본적으로 수치해석 분야에서 대수방정식의 근을 찾는 원리와 같다. 수치해석 분야에서 사용되는 기존의 근을 구하는 방식들로는 modified regular falsi method[7], secant method[7], Newton' s method[7] 등이 있다. 이들 중 secant method와 Newton' s method는 검색 시작 시의 초기값에 따라 검색에 실패할 가능성이 존재하므로 본 비교 연구에서는 modified regular falsi method만을 대상으로 하였다. 즉, 이분 검색, 보간 검색과 그 변형인 fast search, modified regular falsi method만이 본 연구의 검토대상이 된다.

m개의 선형 데이터에 대하여 검색 횟수의 최대값이 $O(log2m)$으로 나타나는 이분 검색법은 데이터 테이블의 구간을 줄이는 방식이기 때문에 데이터의 형태에 영향을 받지 않는다. 반면에 최대값이 $O(m)$인 보간 검

색법과 그 변형들은 직선형 데이터에 대한 검색속도가 매우 빠르지만 변화가 많은 데이터에서는 검색속도가 격감한다. 다양한 크기와 형태의 데이터 테이블들을 사용하여 수치비교시험을 수행하였는데, 그 경향은 이론에 근거한 예상과 대체로 일치하였다. 한편, 검색하고자 하는 파라미터 값이 임의의 원칙 즉, dynamics를 갖고 변한다면, 이러한 정보를 이용하여 주어진 데이터 테이블 내의 검색 영역을 축소 시켜 검색속도를 향상시킬 수가 있다. 다양한 수치시험을 통하여 이분 검색법은 축소된 테이블의 크기에만 영향을 받지만 보간 검색법들은 검색 대상 테이블의 축소로 데이터의 형태가 직선형이 되는 효과를 얻기 때문에 검색속도를 단축시키는데 매우 탁월한 효과를 나타내었다. 결론적으로 dynamic-window개념[8]을 도입한 보간 검색법과 그 변형들은 이론적으로도 실험적으로도 최적의 검색 속도를 보장한다.

2. 검색 알고리즘

실시간 시뮬레이션에서 사용되는 대부분의 데이터들은 불연속적이면서 등 간격이 아니다. 또한 검색하게 되는 파라미터 값들이 데이터 테이블 내에 바로 존재하지 않는다. 따라서 키 값이 아니라 단위 구간에 대한 검색을 수행해야 한다. 실시간 시뮬레이션에서는 시뮬레이션 시작 이전에 off-line으로 데이터의 정렬이 가능하기 때문에 정렬된 데이터를 대상으로 하는 검색 방법들만이 본 비교연구의 대상이 된다.

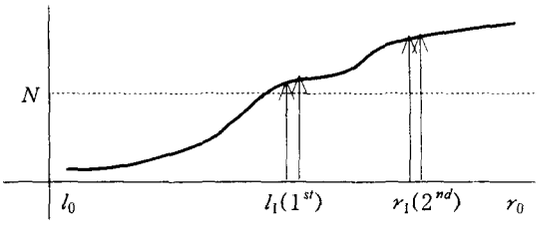
본 절에서는 n개의 data를 가진 1-dimension의 데이터 테이블을 대상으로, 기존의 검색 알고리즘들인 이분 검색법, 보간 검색법, fast search 등과 수치해석에서 대수방정식의 근을 구하는 방식의 변형 기법인 modified regular falsi 기법을 소개한다.

Bisection method(이분 검색법)

이분 검색은 원하는 파라미터 값을 포함하는 단위 구간을 찾을 때까지 데이터 테이블의 크기를 되풀이하여 이등분하는 방식이다. 다음은 n개의 데이터에 대한 알고리즘으로 i번째 데이터를 $D(i)$로 표시하며 구하는 파라미터 값을 N으로 표시한다:

Algorithm

$l=1,\ r=n$
For $j=0,1,2,\cdots$ *until satisfied, do*:
Set $m=(l+r)/2$
If $D(m+1) < N$, *set* $l=m+1$
Else if $D(m) > N$, *set* $r=m$
Otherwise 'N' exist in the inteval $[l, r]$



<그림 1> Bisection Search

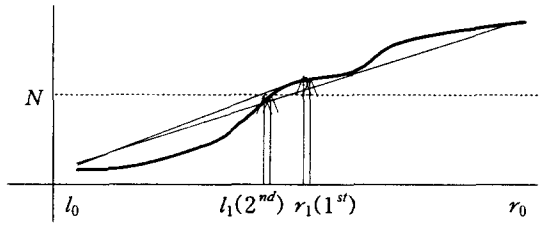
Interpolation method (보간 검색법)

보간 검색은 대수방정식의 근을 구하는 수치해석 기법인 Regular falsi의 변형으로 주어진 데이터 테이블 내의 파라미터 값들이 선형적으로 분포됨을 가정하여 양 단을 잇는 직선상에서 파라미터 값을 포함하는 단위 구간을 검색하는 방법이다. 데이터 테이블의 형태에 따라 검색속도 차이가 심하게 나타나

며, 최악의 경우는 순차 검색을 하게 된다.

Algorithm

$l=1,\ r=n$
For $j=0,1,2,\cdots$ *until satisfied, do*:
Set $m=\frac{(N-D(L))*(r-l)}{(D(r)-D(l))}+1$
If $D(m+1)<N$, *set* $l=m+1$
Else if $D(m)>N$, *set* $r=m$
Otherwise 'N *exist in the inteval* $[l, r]$



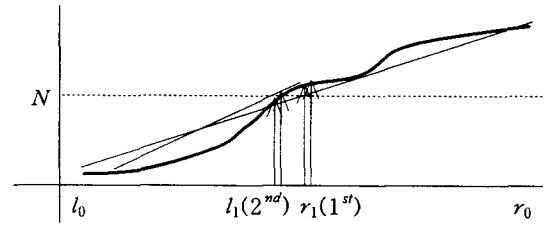
<그림 2> 보간 검색

Fast Search

Fast Search는 보간 검색법의 변형으로, 구간의 크기를 정하는 임의의 수를 이용하여 각 loop에서의 구간을 줄여 키를 포함하는 단위 구간에 대한 검색속도를 증가시키는 방법이다. 최악의 경우 보간 검색법은 순차 검색을 하기도 하는데 Fast Search는 이를 회피할 수 있다. 따라서 최악의 경우에도 검색 속도가 보간 검색 법보다 빠르다.

Algorithm

$l=1,\ r=n$
For $j=0,1,2,\cdots$ *until satisfied, do*:
Set $m=\frac{(N-D(L))*(r-l)}{(D(r)-D(l))}+1$
If $D(m+1)<N$, *set* $l=m+1$
  If also $|m+1, r-1|<|m, l+1|$,
    set $r=r+2$
  Else set $r=r+1$
Else if $D(m)>N$, *set* $r=m$
  If also $|m+1, r-1|<|m, l+1|$,
    set $l=l+2$
  Else set $l=l+1$
Otherwise 'N *exist in the inteval* $[l, r]$



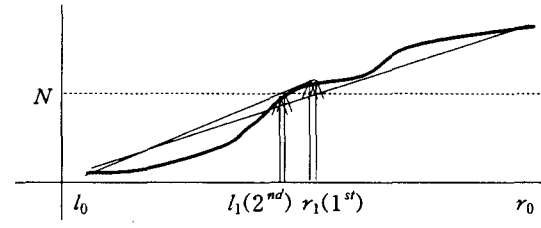
<그림 3> Fast Search

Modified regular falsi

Modified regular falsi 방식은 수치해석에서 대수방정식의 근을 찾는 방법으로 이용되어 왔다. 그 원리는 보간 검색법의 변형으로 구간의 양 끝 점 중 하나의 크기를 반으로 줄여서 키를 포함하는 구간의 크기를 좁혀 검색 속도를 증가 시키는 방법이다.

Algorithm

$l=1,\ r=n,\ L=D(1),\ R=D(n)$
For $j=0,1,2,\cdots$ *until satisfied, do*:
Set $m=\frac{(N-D(L))*(r-l)}{(D(r)-D(l))}+1$
If $D(m+1)<N$, *set* $l=m+1$ and
  $L=D(m+1),\ R=(D(r)+D(r-1))/2$
Else if $D(m)>N$, *set* $r=m$ and
  $R=D(m),\ L=(D(l)+D(l+1))/2$
Otherwise 'N *exist in the inteval* $[l, r]$



<그림 4> Modified regular falsi

3. DYNAMIC-WINDOW SEARCH

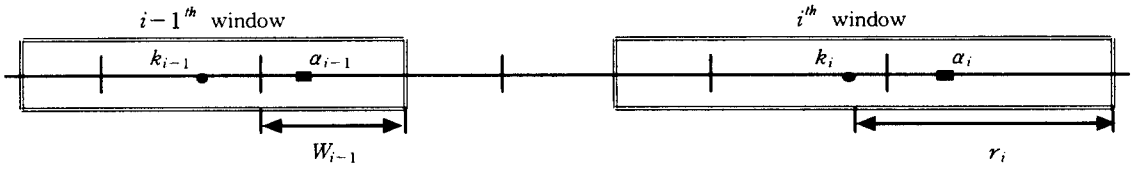
검색 기법에 대한 연구로는 실시간 시뮬레이션의 제약을 기준으로 한 기존 검색 알고리즘들의 비교 이외에도 원래의 주어진 데이터 테이블에서 검색 대상이 되는 영역을

축소함으로써 검색 속도를 향상시키기 위한 dynamic-window search의 개념을 시도하였다. 검색하는 데이터가 임의의 dynamics를 갖고 매 적분 프레임마다 변한다면, 이 정보를 활용하여 주어진 데이터 테이블 중 일부의 영역으로 검색 대상을 축소할 수 있을 것이다. 즉, 데이터가 이전 프레임에서의 위치 주변에서 이동하기 때문에 데이터의 위치를 찾기 위하여 데이터베이스 전체를 검색할 필요가 없다. 데이터의 초기 위치는 실시간 시뮬레이션 시작 이전에 추출이 가능하며, 이후의 데이터 위치는 시스템의 dynamics와 시뮬레이션에서 사용되는 적분 프레임 크기에 근거하여 이동한다.

예를 들어, $C_L(\alpha)$값이 테이블로 주어지며 α가 매 적분 스텝마다 적분 되는 상태변수라 가정하자. 그림 5에서와 같이 α_i를 포함하리라 예상되는 테이블의 부분집합 즉 window의 중심을 k_i라 하자. 즉, $C_L(\alpha_i(t))$를 구하기 위하여 테이블 전체를 검색하는 것이 아니라 k_i를 중심으로 일정한 반경 r_i을 갖는 window만을 검색의 대상으로 하고자 한다. 여기서 k_i는 키의 *index*를 지칭하며, r_i는 키들의 *index*차를 정의하기 위한 정수 값이다. i번째 적분스텝에서 자리할 k_i의 테이블 내 위치는 다음과 같은 방식으로 예상할 수 있을 것이다.

$$k_i = index(\alpha_{i-1}) + int\left(\frac{\alpha_i - \alpha_{i-1}}{w_{i-1}}\right)$$

여기서 *index*는 독립변수 α를 포함하는 2개의 키로 형성된 영역에서 α 진행방향 상위 키의 *index*를 산출하는 함수로 정의 되며, *int*는 독립변수 값과 가장 가까운 정수값을 산출하는 함수이다. 또한, w_{i-1}은 α_{i-1}을 포함하는 구간의 크기이다. 검색 대상 window의 크기는 r_i로 결정되며 다음과 같이 정의된다. 여기서 δ는 상수로 off-line으로 시뮬레이션하여 테이블에 따라 적정 값을 산출하거나, 0으로 정의되어 r의 크기를 고정할 수도 있다. 따라서 α_i의 함수 값을 추출하기 위하여 k_i를 중심으로 하는 window만을 검색 대상으로 하며, window의 하위 *index*는 $k_i - r_i$로, 상위 *index*는 $k_i + r_i$로 정의된다. k_i와 r_i를 연산하기 위한 $i-1$ 스텝의 정보와 α_i는 현 스텝에서 테이블 검색 이전에 주어진 값들이다. 결국, 검색 시간은 테이블 전체 크기가 m에서 dynamic window의 $2r_i$로 축소된 만큼 빨라지게 될 것이다. dynamic window 검색은 다 차원 테이블로 확대되어 적용될 수 있으며, 테이블과 시뮬레이션의 특성에 따라 변형이 가능하다.



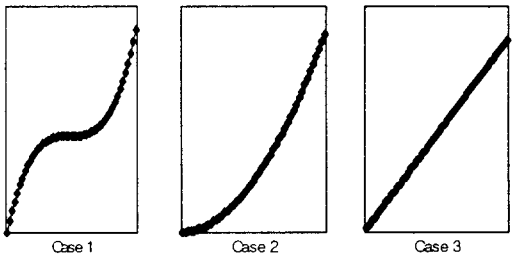
<그림 5> Dynamic-window Search

4. 검색 속도

일반적으로 동적 시스템의 시뮬레이션의 경우 데이터 테이블 내에 주어진 키가 아닌 파라미터 값을 포함하는 단위 구간을 검색하게 된다. 따라서 보통의 검색법을 사용하면 2개의 키를 찾아야 하는데, 이러한 알고리즘을 이용하면 검색 속도가 매우 느려진다. 다음은 이러한 조건에 맞추어 단위 구간, 즉 2개의 키를 하나의 단위로 생각하여 검색법의 속도를 산출한 것이다.

보편적인 이분 검색법을 이용하는 경우에는 어떤 구간을 찾든지 최저(worst case)의 속도인 $O(\log_2 n)$을 갖게 된다. 앞서 언급하였듯이 구간을 하나의 단위로 사용하면 비교횟수는 늘어나지만 $n-1$개의 데이터 테이블에 대한 검색과 같아진다. 또한 이분검색법의 평균속도는 최저 속도와 거의 대동소이하다. 보간 검색법을 이용하는 경우 최악의 경우에는 $O(n)$의 속도를 갖는다. 그러나 데이터의 구간에 따른 변화가 급격한 경우에는 평균속도보다 빠르게 찾고 직선형의 경우에는 단 한번에 찾기 때문에 dynamic-window search를 도입할 경우 이상적이다. 이 검색의 변형인 fast search의 경우 평균 속도는 $O(\log_2 \log_2 n)$이며, 최악의 경우 $O((\log_2 m)^2)$의 속도를 갖는다. 한편, 보간 검색, fast search, modified regular falsi 등의 검색법들은 데이터 테이블의 형태에 따라 검색속도의 기복이 심하므로 이론적인 평균 검색속도가 시험치와 일치하지 않는다.

5. 수치 시험



<그림 6> 수치시험 데이터 테이블의 유형

그림 6과 같이 각각 1000개의 키가 등 간격으로 배치된 3종류의 데이터 테이블에 대해 이분 검색법, 보간 검색법, fast search, modified regular-falsi 등의 검색법들을 시험하였다. 표1에 표시된 수치들은 단위 구간을 찾는데 소요되는 총 탐색(probe) 횟수를 나타내며, 작을수록 검색속도가 빠름을 의미한다. 표1의 결과와 같이 이분 검색법의 검색 속도는 데이터의 형태에 상관없이 데이터 키의 개수에만 좌우된다. 그러나 보간 검색 방법들은 데이터의 형태에 따라 많

<표1> 기존 검색 기법들의 수치시험 비교

	Bisection		Interpolation		Fast		Modified	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Case 1	9.0	10	16.8	65	16.3	64	13.4	61
Case 2	9.0	10	8.8	25	8.7	25	7.4	17
Case 3	9.0	10	1	1	1	1	1	1
Total	9.0	10	8.9	65	8.7	64	7.2	61

은 영향을 받게 된다. 직선형의 경우 데이터의 개수에 관계없이 빠른 속도로 검색을 하나, 굴곡이 심한 데이터의 경우 데이터의 개수에 관계없이 검색 속도가 급격히 감소한다.

다음으로 dynamic-window개념을 도입한 이분 검색법과 보간 검색법을 비교하였다. sine 함수 형태로 분포된 1000×1000개의 등 간격 키 값를 포함하는 2-D 데이터 테이블에서 200개의 파라미터 값을 검색하는데 소요되는 탐색 횟수를 비교하였다. 표2의 결과와 같이 dynamic-window개념을 도입함으로써 이분 검색법이나 보간 검색법 모두에서 데이터 테이블의 검색 대상 영역을 축소하는 효과로 검색 속도가 월등하게 향상됨이 시험적으로도 입증되었다. 특히, 보간 검색법의 경우 검색 영역을 축소하는 효과 뿐 아니라 데이터 분포의 형태를 직선형으로 변환시킴으로써 검색속도를 향상시키는데 탁월한 효과를 나타내었다.

<표2> Dynamic-window search 기법을 도입한 검색법들의 수치시험 비교

	Bisection		Interpolation	
	Mean	Max	Mean	Max
Simple	8.95	10	6.95	13
Dynamic-window	3.16	6	1.21	2

6. 결 론

기존 검색 알고리즘들의 이론적인 검색속도 비교연구와 수치시험결과에서 나타났듯이 최저(worst case) 검색 속도는 이분 검색법이 가장 빠르고, 평균속도는 보간 검색법의 변형들인 fast search나 modified regular falsi가 상대적으로 우수하다. 특히, 직선형의 데이터 테이블에 있어서 보간 검색법들은 매우 빠른 속도를 기대할 수 있다. 또한, 기존의 검색 기법들에 Dynamic-window search를 도입시키면 이분 검색법은 축소된 테이블의 크기에만 영향을 받지만 보간 검색법들은 검색 대상 테이블의 축소로 데이터의 형태가 직선형이 되는 효과도 얻기 때문에 검색속도가 월등하게 향상된다. 결론적으로 dynamic-window개념을 도입한 보간 검색법과 그 변형들은 이론적으로도 실험적으로도 최적의 검색 속도를 보장한다.

후 기

본 연구는 과학기술부의 선도기술개발사업인 감성공학기술개발사업의 위탁과제로 수행되었으며, 지원에 감사 드립니다.

참고 문헌

[1] Kuo-Chi Lin, "The Use of Function Generation in The Real Time Simulation of Stiff Systems, The University of Michigan, 1990.

[2] Sedgewick, Robert, Algorithm in C, Addison-Wesley Pub.Co., 1990.

[3] 박영배, 자료구조, 정보문화사, 1998.

[4] 박정호, (컴퓨터)알고리즘, 상조사,1996.

[5] F. Warren Burton and Gilbert N. Lewis, A Robust Variation of Interpolation

Search, Information Processing Letter, 10(4,5), pp.198-201, 5 July 1980.

[6] Gilbert N. Lewis, Nancy J. Boynton and F. Warren Burton, Expected Complexity of Fast Search with Uniformly Distributed Data, Information Processing Letter, 10(1), pp.4-7, 27 October 1981.

[7] Samuel D. Conte and Carl de Boor, Elementary Numerical Analysis an Algorithmic Approach (3th), McGraw-Hill, 1965.

[8] 윤석준, "A Study of Searching Algorithms for Real-time Simulation of a Dynamic System", Proc. of AIAA Flight Simulation Conference, pp. 269-274, August 1996.