Journal of Korea Game Society 2019 Aug; 19(4): 97-108
http://dx.doi.org/10.7583/JKGS.2019.19.4.97

게임 프로그래밍

명령어간 거리를 최대화하는 음성 게임 명령어의 선택 방법

김상철
한국외국어대학교 컴퓨터및전자시스템 공학부
kimsa@hufs.ac.kr

A Method for Selecting Voice Game Commands to Maximize the Command Distance

Sangchul Kim
Div. of Computer and Electronic Sys. Engineering, Hankuk Univ. of Foreign Studies

요 약

최근 입력 방식의 다양성이나 편리성 때문에 음성 게임 명령어에 대한 관심이 증가하고 있다. 음성 명령어의 인식률은 인식 엔진의 성능뿐만이 아니라, 명령어간의 거리에도 영향을 받는다. 명령어간 거리란 명령어 발음간의 음성적 차이를 말하는데, 이 거리가 클수록 인식률이 높아진다. 본 논문에서 우리는 명령별 명령어 후보들이 주어졌을 때 명령어간의 평균 거리를 최대화하는 명령어 조합을 선택하는 문제를 IP(Integer Programming)으로 모델링한다. 또한 명령어 선택 문제의 해를 구하는 SA(Simulated Annealing) 기반의 방법을 제안한다. 우리의 방법을 명령어 수, 허용되는 명령어 길이 등의 다양한 조건에 하에서 실험한 결과를 토대로 특징을 분석한다.

ABSTRACT

Recently interests in voice game commands have been increasing due to the diversity and convenience of the input method, but also by the distance between commands. The command distance is the phonetic difference between command utterances, and as such distance increases, the recognition rate improves. In this paper, we propose an IP(Integer Programming) modeling of the problem which is to select a combination of commands from given candidate commands for maximizing the average distance. We also propose a SA(Simulated Annealing)-based algorithm for solving the problem. We analyze the characteristics of our method using experiments under various conditions such as the number of commands, allowable command length, and so on.

Keywords: Game(게임), Command(명령어), Phonetic Distance(음성적 거리), Simulated Annealing(시뮬레이티드 어닐링)

Received: Jun. 05. 2019 Revised: Jul. 01. 2019
Accepted: Jul. 23. 2019
Corresponding Author: Sangchul Kim(Hankuk Univ. of Foreign Studies)
E-mail: kimsa@hufs.ac.kr
ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

일반적으로 게임 명령어는 키보드, 마우스 또는 터치스크린 입력을 통해 입력된다. 최근 VR/AR 기반의 게임 또는 체험형 게임에서는 전용 컨트롤러, 모션, 음성 등과 같이 다양한 방법으로 명령어를 입력한다. 전통적 방법인 키보드나 마우스와 달리 이들 입력 방법은 사용자들에게 게임 플레이의 흥미와 몰입도를 높이는 효과가 있다.

어떤 게임 명령을 음성으로 나타내고자 할 때는, 그 명령의 의도(command intent)를 반영하는 여러 표현들이 가능할 것이다. 예를 들면, “캐릭터를 공중으로 올리는 점프” 명령은 “뛰어”, “점프”, “하늘로” 등과 같이 다양한 음성 명령어가 가능할 것이다. 이런 경우 일반적으로 게임 설계자는 각 명령에 대해 여러 명령어 후보들 중에 적절한 것을 선택할 것이다. 본 논문에서는 ‘명령’은 ‘명령 의도’로, ‘명령어’는 어떤 명령을 표현하는 특정 어휘나 구문의 의미로 사용한다.

음성 명령어 선택에서 고려해야 할 요인들 중 하나가 인식률이다. 인식률은 주어진 음성 입력에 대해 해당 명령어를 얼마나 정확하게 인식하는 지를 나타내는 지표이다. 인식률은 우선적으로 인식 엔진의 성능에 좌우되지만, 명령어들 간의 거리, 즉 음성적 차이 (phonetic distance)에도 영향을 받는다[1]. 예를 들면 ‘’위로‘과 ‘뒤로’에 비해, ‘위로’와 ‘후진’ 간의 발음 차이가 크다. 따라서 서로 간에 큰 거리를 갖는 음성 명령어들을 선택하면, 거리가 작은 명령어 선택에 비해 인식률이 높아진다.

명령어 선택 시에는 명령어들 간의 거리 외에도 여러 요인들을 고려해야 한다. 이들 요인들 중 하나가 명령어의 길이이다. 어떤 명령을 표현하는데, 짧은 명령어는 발음 시간 측면에서는 유리하지만, 너무 짧은 명령어의 경우에는 문구 자체로 의미 파악이 잘 안되어 기억에 어려움이 있을 것이다.

명령어들의 공통 사용에 대한 요구가 있을 수 있다. 예를 들면, ‘앞으로 움직여’라는 명령의 후보 명령어들로 ‘앞으로’와 ‘전진’이 있고, ‘뒤로 움직여’라는 명령의 후보 명령어들로 ‘뒤로’와 ‘후진’이 있다. 만약 ‘앞으로’ 가 선택되면 ‘후진’이 아닌 ‘뒤로’를 선택하는 것이 표현의 일관성을 위해 무난할 것이다. 또한 게임이 여러 레벨로 진행될 때, 같은 명령 의도에 대해 레벨별로 다른 명령어를 사용할지 또는 같은 명령어를 사용할 지에 대한 선택도 있을 것이다.

우리는 각 명령에 대해 여러 명령어 후보들이 존재하고 또한 음성 명령어간의 거리가 주어져 있을 때, 앞에서 언급한 요인들을 고려하면서 명령어들 간의 거리를 최대화하도록 각 명령별 명령어를 선택하는 문제를 ‘명령어 선택 문제’라고 정의한다. 명령의 수가 수십 개가 넘고 명령별 명령어의 수가 다수인 경우, 명령어 선택을 수작업으로 수행하는 데에는 한계가 있다.

본 논문에서는 음성 명령어 인식률을 높이기 위해 명령어 선택 문제를 IP(Integer Programming)로 모델링하고, 그 해를 구하는 SA(Simulated Annealing)[2] 기반의 알고리즘을 제안한다. SA는 IP와 같이 폴리노미얼 타임(polynomial time)의 해가 존재하지 않는 최적화 문제(optimization problem)를 해결하는데 사용하는 대표적인 기법이다.

HMM[Hidden Markov Model]은 음성 인식에 오랫동안 사용되어 온 대표적인 방법이다[3]. HMM 기반의 음성 명령어 인식의 경우, 먼저 인식 대상이 되는 명령어별로 HMM 모듈을 훈련시킨다. 인식 과정에서는, 입력 명령어가 주어지면 가장 높은 출력 확률을 생성하는 HMM 모듈을 찾는다. 본 논문에서는 두 음성 명령어간의 거리를 해당 HMM 모듈들이 생성하는 출력 확률들 간의 차이로 정의한다.

음성 인식에 대한 연구는 국내외적으로 활발히 진행되어 왔지만, 음성으로 표현된 게임 명령어에 대한 연구는 초보 상태이다. 우리의 조사에 의하면 음성 인식률 향상을 위한 음성 명령어 선택에 관한 기존 연구는 거의 발표되지 않았다.

2. 관련 연구

음성 인식은 소프트웨어 및 장치의 사용자 인터페이스, 화자 인식, 자동 통역 등의 다양한 영역에서 적용되고 있다. 앞에서도 언급했듯이 HMM은 가장 보편적으로 사용되는 음성 인식 기술이다. 최근에는 ANN(Artificial Neural Network) 기반의 딥 러닝(Deep Learning) 모델이 음성 인식에 적용되어 좋은 음성 인식률을 보였다[4]. HMM과는 달리 인공 신경망 모델에서는 입력 신호의 음향적 모델 및 언어적 모델을 따로 다룰 필요는 없지만, 상대적으로 모델을 훈련하는데 많은 데이터와 시간이 소요되는 어려움이 보고된다[5]. 본 논문에서는 음성 인식을 위해 HMM 기반의 모델을 사용하는 것으로 가정한다. 우리가 조사한 바, ANN기반의 음성 인식에서 음성 명령어들 간의 거리에 대한 연구는 발표된 적이 없다.

지금까지 음성을 게임에 활용하는 게임이 다수 개발되었다[6,7,8,9,10,11]. 이들은 전략 게임[6] 또는 시뮬레이션 게임 [7]으로서 음성 인식을 이용해 명령어 어휘를 입력하고 있다. 그 외 게임들에서는 명령어를 음성으로 발음하는 대신에 음성의 사운드 특징을 이용하고 있다. [8]에서는 사용자가 부르는 노래가 악보상의 피치와 일치하는 지를 측정하고, [9, 10]에서는 단순 외침의 세기 또는 피치를 측정해 게임 유닛의 움직임을 제어한다. [11]에서는 음성 피치, 억양 및 제스처를 종합적으로 사용해 테트리스(Tetris) 게임에서의 유닛 위치 및 방향을 조작한다. [12]에서는 비언어적(non speech) 음성 명령어가 언어적(speech) 음성에 비해 현저히 빠른 명령어 입력이 가능하다는 연구 결과를 발표했다.

음성 게임 명령어의 인식률에 관련된 연구도 발표되고 있다. [13]에서는 명령어의 전체가 아닌 앞부분(prefix)를 입력으로 사용해, 명령어 인식률의 큰 저하 없이 명령어 발음과 인식까지의 처리 시간을 단축한 연구를 발표했다. 게임 명령어를 어떤 어조로 발음하는 지도 인식률에 큰 영향을 미칠 수 있을 것이다. [14]에서는 외치는 식의 발음하는 명령어가 평범하게 말하는 경우에 비해 인식률이 크게 낮아짐을 보였다. 또 다른 방향의 연구는 언어 모델의 선택에 대한 것으로서, [15]에서는 폴란드어 음성 명령어를 HMM 모델로 처리할 때, 3-그램으로 언어 모델을 사용했을 때에 가장 높은 명령어 인식률을 보임을 발표했다.

사람 발언(utterance)들 사이의 음성적 거리의 정의와 관련된 몇 가지 방법이 있다. 하나는 레벤쉬타인 거리(Levenshtein distance)로서[16], 발음의 문자열(spell)들 간의 편집 거리를 측정하는 것이다. 편집 거리란 한 문자열을 다른 문자열로 바꾸고자 할 때 필요한 문자 수정, 삭제 및 삽입 회수이다. 이 방법은 발음을 구성하는 음소간의 유사성을 고려하지 않아 음성적 차이를 나타내는데 한계가 있다. 또 다른 방법은 앞 방법을 개선한 것으로 두 음소간의 유사도를 가중치로 해서 편집 거리를 계산하는 것이다[17]. 또 다른 방법은 확률 모델들 간의 KL 차이(Kullback-Leibler divergence)[18]을 이용하는 것이다. HMM 모델도 확률 모델이므로, 두 HMM 모듈들이 음성 인식 과정에서 생성하는 출력 확률간의 차이는 해당 HMM 모듈들의 훈련에 사용한 발언들의 음성적 차이로 볼 수 있다.

3. 문제의 정의

[Fig. 1]은 명령어간 평균 음성적 거리를 최대화하는 명령어 선택 문제를 IP 문제로 정의한 것이다. 여기서 N은 명령의 개수, M은 각 명령별로 고려중인 후보 명령어들의 개수를 나타낸다. i번째 명령은 c_i로 나타내고, i번째 명령의 j번째 명령어는 $c_{i,j}$로 나타낸다. 집합 PS의 각 원소인 PS_p는 같이 사용되어야 할 명령어들의 인덱스 집합을 나타내는데, 명령어 $c_{i,j}$의 인덱스는 <i, j>로 표현한다. PS_p에 포함된 한 명령어가 선택된다면, 다른 명령어들도 함께 선택되어야 하는 조건을 명시한

다. GS의 각 원소인 GS_g는 명령들의 인덱스 집합으로서, 이들은 선택된 명령어 조합에 대한 명령어들 간의 평균 거리를 계산을 할 때 사용된다.

$DIFF[c_{i,j},c_{v,w}]$는 두 명령어 $c_{i,j}$와 $c_{v,w}$간의 거리로서, 다음과 같이 정의된다:

$$DIFF[c_{i,j},c_{v,w}]= \sum_{x \in U_{i,j} \cup U v,w} |\log(OP_{i,j}(x)) - \log(OP_{v,w}(x)) \ | \ / \ | \ U_{i,j} \cup U_{v,w}|$$

위의 정의는 KL 차이[18]을 HMM 모델에 적용한 것이다. $U_{i,j}$는 $c_{i,j}$를, $U_{v,w}$는 $c_{v,w}$를 발음한 데이터들의 모임이다. 발음 데이터 x와 명령어 $c_{i,j}$에 대해, $OP_{i,j}(x)$는 $c_{i,j}$ 인식을 위해 훈련된 HMM 모델에 x를 적용해서 구한 출력 확률을 나타낸다. 명령어별로 다수의 발음 데이터를 사용해 평균을 이용하는 것이 그 정확도를 높일 것이다, $c_{i,j}$와 $c_{v,w}$의 거리를 구하는데 그 외 명령어들의 발음 데이터는 사용하지 않는다.

Given CS = $\{c_{1,1}, c_{1,2}, ..., c_{1,M}, ..., c_{N,1}, c_{N,2}, ..., c_{N,M}\}$, $DIFF[c_{i,j}, c_{v,w}]$'s for $1 \le i,v \le N$ and for $1 \le j,w \le M$, $PS=\{PS_p \mid 1 \le p \le P\}$, and $GS = \{GS_g \mid 1 \le g \le G\}$, L_{Max}, L_{Min}

Find $S = <y_{1,1}, y_{1,2}, ..., y_{1,M}, ..., y_{N,1}, ..., y_{N,M}>$ that maximizes ACD such that

$$ACD = \sum_{g=1}^{G} \sum_{i,v \in G_g}^{N} \sum_{j=1}^{M} \sum_{w=1}^{M} DIFF[c_{i,j}, c_{v,w}] * y_{i,j} * y_{v,w}) / (\sum_{g=1}^{G} \sum_{i,v \in G_g}^{N} \sum_{j=1}^{M} \sum_{w=1}^{M} y_{i,j} * y_{v,w})$$

$$(C1) \sum_{j=1}^{M} y_{i,j} = 1 \text{ for } 1 \le i \le N$$

$$(C2)\ y_{i,j} = 0 \text{ or } 1 \text{ for } 1 \le i \le N,\ 1 \le j \le M$$

$$(C3) \sum_{< i,j > \in PS_p} y_{i,j} = |PS_p| \text{ for } 1 \le p \le P$$

$$(C4)\ L_{Min} \le \sum_{j=1}^{M} y_{i,j} * LEN(c_{i,j}) \le L_{Max} \text{ for } 1 \le i \le N$$

[Fig. 1] Problem Definition

[Fig. 1]에서처럼, 명령어 선택 문제는 선택된 명령어들 간의 평균 거리인 ACD를 최대화하는 $y_{i,j}$들의 값을 정하는 것이다. 명령 c_i에 대해 j번째 명령어가 선택되었다면 1, 아니면 0의 값을 갖게 된다. ACD는 같은 명령 집합에 속한 명령어 쌍들의 거리 합을 구하고, 그 개수로 나누어 명령어의 평균 거리를 구한다. 게임이 여러 레벨 (또는 서브 게임)으로 나누어져 있는 경우, 명령어 인식은 레벨별로 사용되는 명령어들을 대상으로 진행될 것이다. 따라서 명령어 선택에 대한 명령어간의 평균 거리를 계산할 때는 같은 레벨에서 사용하는 명령들 사이의 차이만을 고려하는 것이 필요하다.

조건 *C1*과 *C2*는 각 명령 c_i에 대해, 한 개의 명령어만 선택되어야 하는 조건을 나타낸다. 조건 *C3*은 각 명령어 집합 PS_p에 대해, 이 집합에 속한 한 명령어라도 선택되면, 그 집합의 모두가 같이 선택되어야 하는 조건을 나타낸다.

조건 *C4*는 게임 설계자가 선호하는 명령어의 길이에 대한 것으로서, 선택된 명령어의 길이가 최소 L_{Min}에서 최대 L_{Max}까지라는 제약을 명시한다. $LEN(\cdot)$는 명령어의 길이를 나타낸다.

4. 명령어간 평균 거리를 최대화 하는 SA 기반의 명령어 선택 방법

명령어 선택을 수행하는 우리의 SA 기반 프로

시저는 [Fig. 2]에 기술되어 있다. [Fig. 1]의 IP 모델링에서 언급한 $y_{i,j}$ 변수 대신에, 코드 작성의 편의를 위해 변수 x_i를 사용한다. x_i는 명령 c_i에 몇 번째 명령어가 선택되었는지를 표현한다. 예를 들면, x_i = 2이면, $y_{i,2}$ = 1이고 그 외 $y_{i,j}$들은 0을 의미한다. 이 프로시저에서 구하는 해(solution)는 x_1, x_2, ..., x_N로 구성된 리스트가 된다.

프로시저 *SelectCmdSetBySA*는 전형적인 SA 기반 알고리즘의 흐름을 따른다. 여기서 *Tinit*은 초기 온도, △는 쿨링 계수로서 실험적으로 정하게 된다. *Fitness*(·)는 주어진 해에 대해 명령어간의 평균 거리를 구하는 함수이다. 본 프로시저의 기본 동작 방식은 다음과 같다. *GetInitSolution* 프로시저를 통해 초기 해를 구한 후, *반복적으로* *GetNextSolution* 프로시저를 통해 다음 해를 구해 현재 해를 점진적으로 개선해 나간다. *GetInitSolution* 프로시저는 소위 brute-force 방식으로 동작하는데, 각 x_i에 가능한 값을 차례로 대입해 가면서, 모든 조건을 만족하는 첫 번째 해를 구한다.

Input: *DIFF*[$c_{i,j}$, $c_{v,w}$]'s for $1 \le i, v \le N$ and $1 \le j, w \le M$, *PS*, *GS*, L_{Max}, L_{Min}
Output: $<x_1, x_2, ..., x_N>$

Procedure *SelectCmdSetBySA*
Step 1)
Get an initial solution s_0 by calling procedure *GetInitSolution*(*DIFF*, *PS*, *GS*, L_{Max}, L_{Min});
$s = s_0$;
$s^* = s_0$;
T = *Tinit*;

Step 2)
for *loop* = 1 **to** $LOOP_{max}$ **do** {

Step 2-1)
Get the next solution s_N by calling *GetNextSolution*(*s*, *DIFF*, *PS*, *GS*, L_{Max}, L_{Min});

Step 2-2)
if(*Fitness*(s_N) > *Fitness*(s)) {
s = s_N;
if(*Fitness*(s) > *Fitness*(s^*))
$s^* = s_N$;
}
else if(*exp*((*Fitness*(s) − *Fitness*(s_N))/T) > *random*(0, 1) {
$s = s_N$;
}
Step 2-3) $T = \triangle * T$;
}

Step 3) Output s^*;

[Fig. 2] Procedure SelectCmdSetBySA

Input: the current solution *s*, *DIFF*, *PS*, *GS*, L_{Max}, L_{Min}
Output: a new solution

Procedure *GetNextSolution*
Step 1)
Let s be $<x_1, x_2, ..., x_N>$;
$s' = s$; Let s' be $<x_1', x_2', ..., x_N'>$;
Set L to an empty list;

Step 2)
Step 2-1)
Randomly choose some index i such that $1 \le i \le N$;

Add i to L;
if(*PS* contains some element, say PS_p, that contains entry $<i, x_i>$) {
for each entry, say $<v, w>$, **in** PS_p **do**
Append v to L;
}
Step 2-2)
for o = 0 **to** *sizeof*(L) − 1
Set TRY_o to an empty list;

```
Step 3)
  o = 0;
  while(o < sizeof(L)) {

  Step 3-1)
     Check whether there is an index, say
     j, such that 1 ≤ j ≤ M, x_{L[o]} != j,
     and c_{L[o],j} ∉ TRY_o;

  Step 3-2)
     if(there is no such index) {
        o--;
        if(o < 0) break;
        Set TRY_o to an empty list;
        continue;
     }
     else
      Randomly select one among those
      indices if there are more than one;

  Step 3-3)
     sSAV = s';          //save s'
     TRYSAV = TRY_o; //save TRY_o

  Step 3-4)
     Let j be the index selected in Step 3-1;
     x'_{L[o]} = j;
     Append c_{L[o],j} to TRY_o;

  Step 3-5)
     if(PS contains some set e that
     contains entry <L[o], j>)
        for each entry, say <v, w>, in e do
           x'_v = w;

   Step 3-6)
      Check whether s' meets constraints
      C3 and C4;
      if(true)
        o++;
      else {
        s' = sSAV;          //restore s'
       TRY_o = TRYSAV; //restore TRY_o
      }
   }
```

```
Step 4) Output the next solution s';
```

[Fig. 3] Procedure GetNextSolution

현재 해인 s로부터 다음 해인 s'를 구하는 *GetNextSolution* 프로시저는 [Fig. 3]에 기술되어 있다. 리스트 변수 L은 새로운 선택이 필요한 명령들의 리스트를 저장한다. $L[o]$는 L의 o번째 명령의 인덱스를 나타낸다. Step 2에서는 임의의 명령을 선택해 그 인덱스를 리스트 L에 등록한다. 만약 그 명령의 명령어가 PS_p 집합에 존재하면, 해당 명령어와 같이 사용해야하는 명령어의 명령도 L에 추가한다.

Step 3에서는 L의 모든 명령들에 대해 명령어 선택이 끝날 때까지, Step 3-1에서 Step 3-6을 반복 수행한다. 그 반복 동안에, TRY_o변수는 L 리스트의 o번째 명령에 대해 그때까지 고려해 보았던 명령어들을 저장한다. Step 3-1에서는, $L[o]$에 지정된 명령에 대해 아직까지 고려하지 않은 명령어들이 존재하는 지를 검사한다. 만약 그런 명령어가 없다면 그전 명령에 대한 선택을 취소하고 새로운 명령어 선택을 시도해 나간다. 그런 명령어들이 있다면 그중 하나는 선택하고 그 인덱스를 j라고 한다.

Step 3-3은 다음 단계를 위해 s'와 TRY_o를 별도로 보관한다. Step 3-4에서는 다음 해를 구성하는 원소인 $x'_{L[o]}$에 방금 선택한 인덱스 j를 정하고, TRY_o에 명령어 $c_{L[o],j}$를 추가한다. Step 3-6에서는, 만약 명령어 $c_{L[o],j}$가 PS에 있다면, 그 명령어와 같이 사용해야할 명령어들도 같이 사용하는 것으로 s' 리스트를 적절히 수정한다.

Step 3-6에서는 그렇게 수정된 s'가 조건 *C3*과 *C4*를 만족하는 지를 검사한다. 그 검사 결과에 따라 다음 명령에 대해 Step 3-1에서 3-6을 반복 수행해 나가거나, 또는 Step 3-3에서 보관한 값을 이용해 전 단계 Step 3-5에서 한 일을 취소시킨다.

[Fig. 4]에서 기술된 Greedy*Solution* 프로시저는 그리디(greedy) 방식으로 명령어 선택을 수행한다. 본 프로시저의 용도는 우리의 SA 기반 명령어 선택 방법의 성능을 비교해 보기 위함이다.

CIDIST(c_i)은 주어진 명령 c_i와 모든 다른 명령들 간의 평균 거리를 나타낸다. 이것은 c_i의 명령어들과 모든 다른 명령의 명령어들 간의 평균 거리로 구한다. *CDIST*(.)은 주어진 특정 명령어와 다른 명령들 간의 평균 거리를 계산하는 함수이다. 예를 들면 *CDIST*($c_{i,j}$)는 $c_{i,j}$와 모든 타 명령의 명령어들 간의 평균 거리이다.

Input: *DIFF*[$c_{i,j}$, $c_{v,w}$]'s for $1 \leq i, v \leq N$ and for $1 \leq j, w \leq M$, *PS*, and *GS*, L_{Max}, L_{Min}
Output: <x_1, x_2, ..., x_N>

Procedure Greedy*Solution*

Step 1)
Let s' be <x_1', x_2', ..., x_N'>;

Sort all command indices, say i, from 1 to N in decreasing order of *CIDIST*(c_i); Set L to the sorted list of those indices;

Step 2)
for o = 0 **to** *sizeof*(L) - 1
Set TRY_o to an empty list;

Step 3)
o = 0;
while(o < *sizeof*(L)) {

Step 3-1)
Find whether there is some index, say j, such that $1 \leq j \leq M$, and $c_{L[o],j} \notin TRY_o$;

Step 3-2)
if(there is no such index) {
o--;
Set TRY_o to an empty list;
continue;
}
else
Select one with maximum *CDIST*(.) among those indices if there are more than one;

Step 3-3)
Perform the same steps as Step 3-3 through Step 3-6 of Procedure *GetNextSolution*;

Step 4) Output solution s';

[Fig. 4] Procedure GreedySolution

5. 실 험

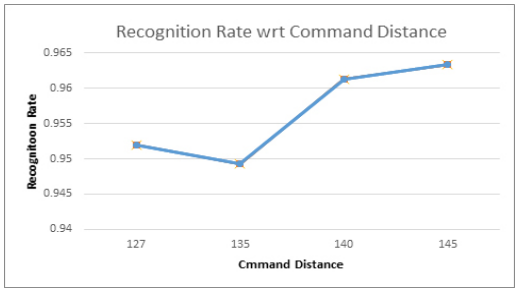
5.1 명령어간의 거리와 인식률과의 관계

앞에서 언급한 *DIFF*[·,·]를 이용해 정의한 명령어간 평균 거리인 *ACD*이 인식률에 미치는 영향을 알아보는 실험을 먼저 실시했다. 이를 위해 간단한 액션 게임에서 사용될 명령 12개와 각 명령별 2개씩의 명령어를 정했다. 이들 명령어에는 '뒤로'와 '후진' 같은 캐릭터 움직임이나 '뛰어차' 또는 '높이차' 와 같은 공격에 관한 것이 포함된다. 각 명령어별로 20개씩의 발음 데이터를 수집했다. 이들 데이터의 반은 HMM 모델의 훈련에, 다른 반은 인식에 사용했다. 이들 훈련과 인식 절차는 화자 종속적인 방식으로 수행했다.

명령어 선택의 결과로서 4개 명령어 조합을 임의로 선택했는데, 이들의 *ACD*는 각각 127, 135, 140, 145 정도이다. 각 명령어 선택에 대해 게임 명령어 인식률을 측정했다. 인식률 차이를 보다 분명히 파악하기 위해, 준비된 음성 데이터 원본뿐만이 아니라 잡음(noise)가 섞인 음성 데이터를 사용했다. 게임 플레이 환경이 게임 효과음 또는 생활 잡음이 있는 경우가 많고 잡음이 음성 인식률에 영향을 미치기 때문에[19], 이러한 실제 상황을 고려했다. 잡음 모델로는 생활 잡음 모델로 많이 사용하는 가우시안 백색 노이즈를 채택했다. 노이즈

의 세기는 실험적으로 정했는데, 명령어 발음 데이터의 5 ~ 10% 정도로 선택했다.

[Fig. 5]는 측정한 인식률 결과를 보여준다. 그림에서 보다시피, 명령어 선택의 명령어간 평균 거리가 커질수록 게임 명령어의 인식률이 향상되는 경향을 보였다.

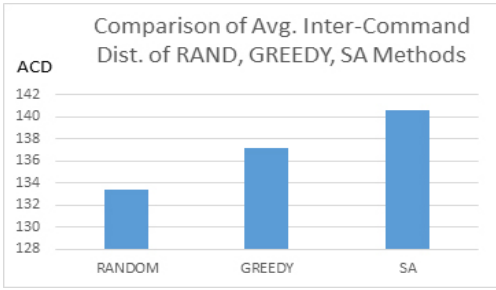


[Fig. 5] The result of testing for the dependency of recognition rate on distances between voice commands

5.2 제안된 명령어 선택 방법의 성능 및 특징

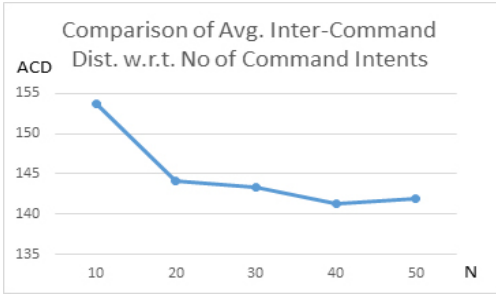
우리는 제안된 명령어 선택 방법의 성능과 특징을 분석하고자 다양한 조건하에서 여러 실험을 수행했다. 각 실험마다, 실험 조건으로 명령 의도의 수인 N과 명령 의도별 명령어 개수인 M이 주어지면, 임의의 두 명령어 간의 거리는 자동으로 부여했다. 두 명령어 사이의 거리 분포는 5.1에서 기술한 실험의 결과를 기반으로 했다. 두 명령어 사이의 거리는 최소 30 ~ 최대 210 사이에 있고, 대략 120 ~ 150이 40%, 그 외 구간이 60%로 분포된다. 앞선 실험에서처럼, 각 명령어의 길이는 2에서 8사이의 임의 값으로 정했다. 그 이유는 대부분의 실제 게임 명령어의 음절 개수가 2개 또는 3개이고, 이것을 음소 단위로 보면 대략 2개에서 8개이기 때문이다. 명령어 길이의 분포는 4가 20%, 5가 30%, 6이 20%, 나머지가 20%를 차지한다.

먼저 우리의 명령어 선택 방법과 일반적인 방법을 비교했고, [Fig. 6]은 그 결과를 보여준다. 그림에서 SA는 *SelectCmdSetBySA* 프로시저, *RAND*는 무작위 선택, *GREEDY*는 [Fig. 4]의 프로시저를 나타낸다. 본 실험은 N=50, M=4, 명령 그룹의 수인 G=1, *C3*나 *C4* 타입 조건은 특별히 없는 상황에서 수행되었다. 그림에서처럼, 우리의 방법은 *RAND*와 *GREEDY*에 비해 *ACD*가 3 ~ 6% 정도 증가한 명령어 조합을 구했다. N, M, G를 다르게 설정해도 비슷한 결과를 얻을 수 있었다.



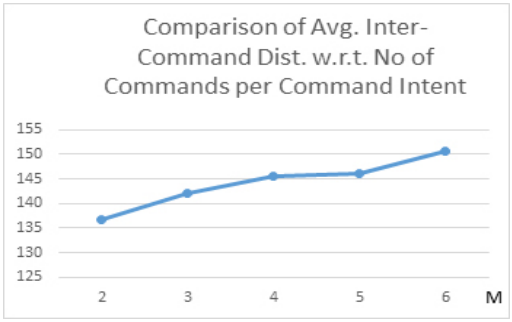
[Fig 6]. Comparison of Avg. Inter-Command Distances of RAND, GREEDY and SA

다음 실험에서는 명령의 개수의 차이가 명령어 선택의 결과에 주는 영향을 알아보았다. M=4, G=1로 설정하고, *C3*나 *C4* 타입의 제약 조건은 없었다. [Fig. 7]에서 나타나듯이, N이 20보다 크면, 명령어 선택의 결과가 N에 크게 영향을 받지 않는다. N=10이 다른 값보다 더 큰 명령어간 평균 거리를 보이는 것은, 탐색 공간이 상대적으로 좁아서 SA 알고리즘의 특성상 최적 해에 가까운 해를 구할 기회가 높기 때문으로 판단된다.



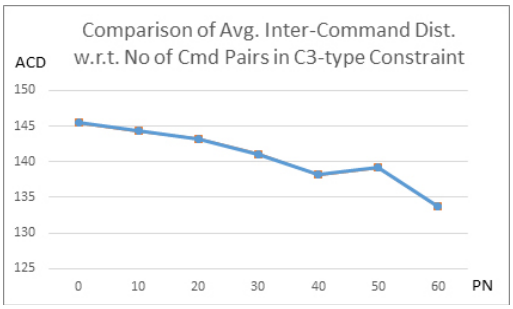
[Fig 7]. Comparison of Avg Inter-Command Distances w.r.t. No of Command Intents

[Fig. 8]은 명령별 명령어 수의 차이가 명령어 선택의 결과에 미치는 결과를 보여준다. N=30, G=1, $C3$과 $C4$ 타입 제약은 없는 조건에서 실험했다. M이 증가함에 따라 ACD가 증가하는 경향을 보였다. 이것은 명령별 선택의 폭이 넓어져서, 명령어간 거리 면에서 보다 나은 명령어 조합을 선택할 가능성이 높아지기 때문으로 판단된다.



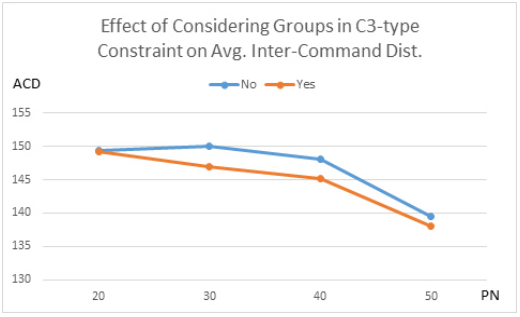
[Fig 8]. Comparison of Avg. Inter-Command Dist. w.r.t. No of Commands Per Command Intent

다음으로 $C3$ 타입 제약의 정도에 따른 명령어 선택의 결과의 차이를 비교했는데, N=30, M=4, G=1, $C4$ 타입 제한은 없는 것으로 설정했다. [Fig. 9]는 그 결과를 보여주는데, PN은 $C3$ 타입 제약에서 같이 사용해야할 명령어 쌍의 개수를 나타낸다. 그림에서 나타나듯이, PN이 증가하면 ACD가 줄어드는 경향을 보인다. $C3$ 타입 제약이 강화되면 [Fig. 2]의 프로시저에서 해를 개선하는데 명령어 선택 폭을 줄임을 알 수 있다.



[Fig. 9]. Comparison of Avg. Inter-Command Distances w.r.t. No of Command Pairs in C3-type Constraint

또한 우리는 $C3$ 타입 제약을 정하는 데, 명령어 그룹의 경계 여부가 결과에 미치는 영향을 분석했다. 명령어 그룹의 경계 여부란, 같은 명령어 그룹에 속한 명령어들에 대해서만 $C3$ 타입 제약이 정해지는 지 여부를 말한다. 실험 결과, [Fig. 10]에서처럼 명령어 그룹의 경계가 있는 경우가 없는 경우에 비해 ACD가 상대적으로 적었다. 이 실험은 N=30, M=4, G=3 조건하에서 진행되었다. ACD의 정의에 따라 평균 거리의 계산에는 같은 그룹에 속한 명령어들 간의 거리만 고려한다. $C3$ 타입 제약이 이런 명령어들에만 명시되면, 그렇지 않은 경우에 비해 해를 찾는 탐색 과정에서 선택할 수 있는 명령어 조합의 수가 더 줄어들기 때문으로 판단된다.

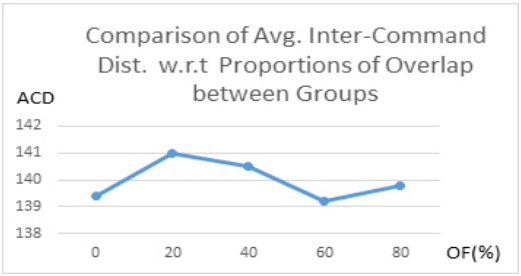


[Fig. 10]. Effect of Considering Groups in C3-type Constraints on Avg. Inter-Command Dist.

게임에서 한 명령어가 여러 레벨에 걸쳐 사용될 수도 있을 것이다. 이번에는 명령어 그룹들 간에 공통으로 사용하는 명령어가 존재하는 경우에 대해 실험해 보았다. 그 실험의 결과는 [Fig. 11]에 나타난다. 여기서 OF(Overlap Factor)는 명령어 그룹들 간의 전체 명령어 수 대비 공통되는 명령어들의 비율을 나타낸다. 본 실험은 N=30, M=4, G=2(그룹의 크기는 같음), $C3$와 $C4$ 타입 제약은 없는 조건하에서 실시했다. 그림에서처럼, OL의 차이는 선택된 명령어들 간의 평균 거리에 의미 있는 영향을 미치지 못함을 알 수 있다.

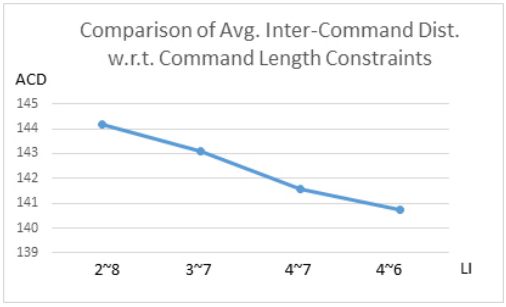
또 다른 우리의 실험에 의하면, 총 명령어 수를 유지한 채 명령어 그룹의 수의 변화를 주어도, ACD

가 감소하거나 또는 증가하는 식의 일정한 경향을 보이지는 않았다.



[Fig. 11] Comparison of Inter-Command Dist. w.r.t. Proportions of Overlap between Command Groups

마지막으로 우리는 명령어 길이의 제약의 강도에 따른 명령어 선택의 차이를 비교했고, 그 실험은 N=30, M=4, G=1, $C3$ 타입 제약은 없는 조건에서 수행했다. [Fig. 12]에서 LI는 허용되는 명령어 길이의 범위를 L_{Min} ~ L_{Max} 형태로 나타내고 있다. 그림에서 보다시피, 허용되는 명령어 길이의 제한 폭이 좁아질수록, ACD가 줄어드는 명령어 선택을 얻는 것을 알 수 있다.



[Fig. 12] Comparison of Inter-Command Dist. w.r.t. Command Length Constraints

6. 결 론

최근에 체감형 게임이나 가상현실 게임에서는 입력의 편리함이나 몰입감의 향상을 위해 키보드나 마우스 입력 대신에 음성 명령어에 대한 관심이 높다. 음성 명령어의 인식률은 음성 인식 엔진의 성능뿐만이 아니라 명령어들 간의 음성적 거리에 영향을 받는다. 특히 게임 플레이는 많은 경우 생활 잡음이나 자체 음향이 있는 환경에서 진행되기에, 높은 음성 명령어의 인식률이 필요할 것이다.

본 논문에서 우리는 명령별로 명령어 후보들이 주어졌을 때 명령어간의 평균 음성 거리를 최대화하는 명령어 조합을 선택하는 문제를 IP로 모델링하고, 그 해를 구하는 SA 기반의 알고리즘을 제안했다. 우리는 음성 인식을 HMM 기반의 모델로 하는 것을 가정했다. 최근에 연구되는 ANN기반의 모델에서도 음성 명령어들 간의 발음 거리를 적절히 정의한다면, 제안된 방법은 ANN 기반의 음식 인식을 위한 명령어 선택에도 적용될 수 있을 것이다. 우리의 방법은 특히 적당한 수준으로 주변 잡음이 있거나 발음이 불명확한 경우에 음성 명령어의 인식률 향상에 기여할 것이다.

우리가 조사한 바, 음성 명령어들을 선택하는 문제를 모델링하고, 인식률 향상을 위해 최대 명령어 간 거리를 가진 명령어 조합을 찾는 방법에 대한 기존 연구는 없었다.

실험 결과, 본 논문에서 제안한 방법이 일반적인 방법보다 나은 결과를 보여 주었다. 또는 우리의 방법이 다양한 제약 조건에 따라 선택된 명령어 조합의 평균 거리가 달라지는 특징을 보였다. 해당 제약 조건들에는 명령별 명령어 후보들의 개수, 같이 사용해야 할 명령어들의 개수, 허용되는 명령어 길이 등이 포함된다.

ACKNOWLEDGEMENT

This work was supported by Hankuk University of Foreign Studies Research Fund of 2018.

REFERENCES

[1] M. Mohri, "Edit-Distance of Weighted Automata", Int'l Conf. on Implementation and Application of Automata, 2002, pp.1-23.

[2] "Simulated Annealing", https:// en.wikipedia.org/wiki/Simulated_annealing, accessed July 10, 2019.

[3] L.R. Rabiner, "A tutorial on Hidden Markov Models and selected applications in speech recognition," Proceedings of the IEEE, 77, No.2, 1989, pp.257-286.

[4] "Speech Recognition", http://en.wikipedia.org/wiki/Speech_recognition, accessed July 10, 2019.

[5] T. Schatz1, N. H. Feldman, "Neural network vs. HMM speech recognition systems as models of human cross-linguistic phonetic perception", Proceedings of the Conference on Cognitive Computational Neuroscience, 2018. pp.1-4.

[6] "Tom Clancy's Endwar", http://endwargame.us.ubi.com/, accessed July 10, 2019.

[7] "Tom Clancy's H.A.W.X", http://www.hawxgame.com/, accessed July 10, 2019.

[8] "Karaoke Revolution", https://en.wikipedia.org/wiki/Karaoke_Revolution, accessed July 10, 2019.

[9] "Shout n Dodge", https://www.playitontheweb.com/games/Shout-n-Dodge-game.htm,accessed July 10, 2019.

[10] "Racing Pitch", http://jet.ro/games/racing-pitch/, accessed July 10, 2019.

[11] Sporka, A.J., Kurniawan, S.H., Mahmud, M., Slavík, P., "Non-speech input and speech recognition for real-time control of computer games", Proc. 8th International ACM SIGACCESS Conference on Computers and Accessibility, 2006, pp. 213 - 220.

[12] S. Harada, et. al., "Voice games: investigation into the use of non-speech voice input for making computer games more accessible", Proc. of 13th IFIP TC 13 Int'l Conf. on Human-computer interaction, 2011.

[13] D. Park, S. Kim, "A HMM-based Method of Reducing the Time for Processing Sound Commands in Computer Games", Journal of Computer Game Society, 16(2), 2016, pp.119-128

[14] H. Nanjo el al., "A fundamental study of novel speech interface for computer games," IEEE 13lh Int'l Symp. on Consumer Electronics, 2009, pp. 558-560.

[15] A. Janicki, D. Wawer, "Automatic Speech Recognition for Polish in a Computer Game Interface", Proc, Federated Conference on Computer Science and Information Systems, 2011, pp.711-716.

[16] H. Hyyro, "A bit-vector algorithm for computing Levenshtein and Damerau edit distances", Nordic Journal of Computing, 10(1), 2003, pp.29-39.

[17] A. Cutler, A. Weber, R. Smits, and N. Cooper, "Patterns of english phoneme confusions by native and non-native listeners," J. Acoust. Soc. Am., vol. 116, pp. 3668 - 3678, 2004.

[18] J. Goldberger, H. Aronowitz, "A distance measure between GMMs based on the unscented transform and its application to speaker recognition," in Proc. of INTERSPEECH, 2005, pp. 1985 - 1988.

[19] S.D. Peters, et. al., "On the limits of speech recognition in noise", Proc. of ICASSP, 1999, pp. 365-368.

김 상 철 (Kim, Sangchul)

약 력 : 1994 미시간주립대학교 컴퓨터공학 박사
1983-1994 전자통신연구원 선임연구원
1994-현재 한국외국어대학교
컴퓨터및전자시스템공학부 교수

관심분야 : 게임프로그래밍, 멀티미디어시스템