Journal of the Korean Institute of Industrial Engineers
Vol. 26, No. 4, pp. 315-324, December 2000.

효율적인 동적계획법을 이용한 최적 교통 신호제어

박윤선 · 김창욱

명지대학교 산업시스템공학부

Optimal Traffic Signal Control Using an Efficient Dynamic Programming

Yunsun Park · Chang-Ouk Kim

This paper presents an efficient dynamic programming(DP) method, so called EDPAS (Efficient Dynamic Programming Algorithm for Signal), for optimally controlling traffic signal in real-time mode at a single intersection. The objective of EDPAS is to minimize total vehicle delay. It applies reaching method to solve forward DP functional equation, which does not need any priori knowledge on the states of DP network. Two acceleration techniques within reaching method are the main feature of EDPAS. They are devised to eliminate inferior DP states by comparing between states and maintaining incumbent value, resulting in a great amount of computational efficiency. An example is shown to verify the advantage of EDPAS.

1. 서 론

ITS(Intelligent Transportation System)는 혼잡, 비효율성, 위험 등의 오늘날의 교통문제를 해결하기 위해 발전되었다. 성공적인 ITS 수행을 위한 가장 중요한 문제 중 하나는 교차로에서의 효율적인 신호제어를 위한 실시간, 적응 신호계획을 고안하는 것이다. 본 논문에서는 단일 교차로에 대해 계산상의 가속 스킴과 함께 효율적인 최적 신호정책 알고리듬에 대해 다룰 것이다. 이 알고리듬의 목표는 인접한 교차로에서 본 교차로로 향하는 차량들의 데이터를 실시간으로 전송 받아 본 교차로를 통과하는 차량의 총 지연시간을 최소화하는 것이다. 이러한 수행 척도는 비용함수로 계산된다.

교통제어 정책은 크게 두 가지의 범주로 나눌 수 있다. 하나는 고정주기제어 정책이고, 다른 하나는 적응제어 정책이다. 고정주기제어 정책에 기반을 둔 대표적인 기법은 Webster 방법(Webster, 1958)이 있으며, 이 기법은 현재까지 널리 이용되고 있다. 이러한 기법은 고정주기의 교차로에서의 녹색신호 점등시간의 최적 분할에 대한 공식을 제공하며, 효율적인 주기 시간(Cycle Time)을 찾아낸다. 그러나 고정주기 신호정책은 일반적으로 과거의 기록이나 관찰된 양에 의해 계산되기 때문에 실시간으로 변하는 복잡한 교통 상황을 수용하고 제어하기 위해서는 좀더 정교한 형태의 신호제어 방법이 필요하게 되었다.

적응 신호제어 정책이 이러한 현재의 교통상황을 해결하기 위해 실제 교통상황의 영향을 받은 신호정책이다. SCOOT (Hunt *et al.*, 1981)와 SCATS(Sims, 1979)는 현재 이용되는 주목할 만한 적응 신호제어 정책이다. 그러나 그들이 채택하는 신호계획은 중앙 집중 방식이다. 즉, 그 방법들은 미리 결정된 신호계획에 기초를 두고 Off-Line의 신호 계획을 선택하고 있다. OPAC(Gartner, 1983), PRODYN(Henry *et al.*, 1983), SPORT(Yager and Han, 1994), UTOPIA(Mauro and DiTaranto, 1990), 그리고 ALLONS-D(Porche and Lafortune, 1997)는 근사적 적응 신호제어 방법들이지만 집중 방식이 아닌 분산 방식이다. 이러한 분산 방식의 방법들은 실시간 데이터를 측정하고, 최적의 신호정책을 찾고, 또한 On-Line으로 신호정책을 찾는다. 한편 이 방법들은 시간 주기를 더 짧은 단계로 나누고 오직 각 단계의 첫번째 결정만 수행하는 Rolling Horizon 개념을 사용한다. 최근에 실시간 적응 교통제어에 대한 COP(Controlled Optimization of Phases) 이라는 알고리듬이 Sen and Head (1997)에 의해 개발되었다. COP은 동적계획법 접근을 사용한다. 그러나 COP은 계산상의 편리를 위해 상태(State)들을 근사적으로 군집화(Aggregation)하기 때문에 교차로에서의 총 지연을 최소화하기 위한 최적의 신호정책을 만들지는 못한다. COP에서의 상태는 시간의 단일 변수로 구성된다. 총 지연을 최소화하기 위한 최적의 신호정책을 얻기 위해 고려하는 교차로에서의 차량의 수는 계산상의 편의를 위해 사용된다. 앞서 언급한 다른 스킴들도 최적화 성질을 띠지 않는다. OPAC와 PRODYN도 동적계획 문제화를 언급하지만 알고리듬을 위해서 동적계획의 최적 성질을 사용하

지는 않는다. ALLONS-D도 최적 스킴으로 사용될 수 있지만, ALLONS-D는 최적해를 위해 거의 철저하게 모든 상태를 고려하기 때문에 계산상의 효율성이 떨어진다.

본 연구에서는 교차로 내에서 차량의 총 지연 시간을 최소화하기 위한 적응성 높고, 최적 성질을 갖고 계산의 효율성이 높은 신호정책 알고리듬을 소개한다. 본 알고리듬은 최적값의 손실 없이 불필요한 상태를 제거하는 두 가지의 가속기법을 이용한 동적계획법에 기초를 두며, 이를 EDPAS(Efficient Dynamic Programming Algorithm for Signaling)라고 부른다. 다른 스킴들과 비교하여 EDPAS는 최적 신호정책을 생성시킨다. 본 알고리듬에서의 의사결정은 각 의사결정 시점에서 미리 정해진 양의 시간을 할당할 단계(Phase)를 찾음으로써 이루어진다. 그러므로 이러한 방법은 고정된 주기로 제한되지도 않고, 어떠한 단계의 순서(Sequence)도 생성할 수 있다. 여기서 단계란 충돌 없이 녹색신호를 받을 수 있는 방향을 일컫는다. 예를 들어, 단일 교차로에서 직진과 좌회전이 허용되면, 총 단계는 양쪽 방향의 직진(첫번째와 두 번째 단계)과 양쪽 방향에서의 좌회전(세 번째와 네 번째 단계)이 있다.

일반적으로 동적계획법 문제는 후진전개(Backward)과 전진전개(Forward)의 두 가지 접근 방법으로 풀 수 있다. 주로 순환고정(Recursive Fixing) 방법을 사용하는 후진전개 방법은 계산을 위해 DP 네트워크의 완전한 사전 정보를 필요로 한다. 그러나 전진전개 방법과 리칭(Reaching)기법을 적용하면 DP 네트워크에서 필요한 상태만을 생성해 가면서 문제를 풀기 때문에 계산상의 부담을 줄일 수 있다. 그러므로 본 논문에서는 교통신호 계획을 위한 비 제한적이고 최적인 연구를 수행하기 위한 전진전개 동적계획 알고리듬을 개발한다. 구체적으로 본 알고리듬은 이미 현재 해(Incumbent) 값을 초과한 비용(지연)을 갖는 상태들을 제거하기 위해서 현재 해를 알고리듬 내에 유지함으로써 계산상의 효율을 가속시킬 것이다. 또 다른 가속기법은 같은 시간, 같은 단계의 두 상태를 비교하는 것이다. 만일 하나의 상태가 다른 상태보다 더 많은 차량이 대기시킨다면 최적값의 손실 없이 그 상태를 제거할 수 있으므로, 네트워크의 상태 수를 줄일 수 있다. 본 논문에서는 이 두 가지 가속기법을 리칭기법 내에 적용함으로써 효율적인 전진전개 순환 알고리듬을 개발한다.

2절에서는 교차로에서의 신호제어 문제에 대한 동적계획 문제 공식화에 대해 설명할 것이다. 2절에 기초하여, 3절에서는 가속 스킴과 그에 따른 효율적인 동적계획 알고리듬에 대해 설명할 것이다. 4절에서는 3절에서 설명한 알고리듬의 효율성을 예제를 통해 설명할 것이다. 5절에서는 몇 가지 주목할 점과 향후 연구과제를 제시하고 결론을 맺을 것이다.

2. 최적 신호제어 계획을 위한 동적 계획법 문제

앞 절에서 언급한 바와 같이 대부분의 신호제어 방법들은 근사적 접근방법으로 개발되어왔다. 이 절에서는 신호제어 문제를 최적으로 풀기 위해 동적 계획법에 적합한 공식을 소개하며, 이는 3절에서 제시될 가속화 기법의 기초가 된다.

2.1 문제 정의

동적계획 문제 공식에 대한 일반적인 절차에 대해서는 Denardo (1982)에 잘 나타나있다. 시간 0부터 H까지(Finite Horizon 문제) 교차로에서의 차량 총 지연을 최소화하기 위한 동적계획 문제를 공식화하기 위해서는 그 문제에 대한 상태를 정의해야한다. 상태에 대한 정의를 기초로 하여 순환함수식을 적절히 표현하는 것은 문제 해결의 기본이 된다. 같은 문제라 할지라도 많은 다른 상태 정의들이 있을 수 있고, 그 결과로 다른 동적계획 공식화가 있을 수 있다. 또한 다른 상태 정의를 내림으로써 각 의사결정 시점에서의 다른 의사결정 방법들이 나온다.

본 논문에서 각 의사결정 시점에서의 의사결정은 다음 의사결정 시점까지 녹색점등을 받아야 하는 단계를 결정하는 것이다. 단계의 집합은 교차로를 통과하는 차량들의 서로 다른 방향을 뜻한다. 또 다른 가능한 의사결정은 COP에서처럼 현재의 단계를 얼마나 오래 유지할 것인가 이다. 그러므로 만약 다음 의사결정 시점까지 현재의 녹색점등 단계를 유지하기로 결정하면 우리는 미리 정해놓은 녹색점등 시간(Δ)동안 현재의 녹색점등 시간을 유지한다. 이에 반하여 만약 다른 녹색점등 단계를 만들기로 결정하면 미리 정해놓은 황색, 적색점등(t_{YR})을 이용하여 교차로를 비우고 새로운 녹색 단계동안 미리 정해진 최소 녹색점등 시간(t_{MG})을 적용한다.

각각의 의사결정들은 새로운 상태를 이끌어낸다. 동적계획 문제에서 상태는 의사결정으로부터 다음 단계로 넘어가기 위한 최소한의, 그리고 완벽한 정보를 가지고 있어야 한다. 그렇기 때문에 각 의사결정 시점에서 다음의 단계가 녹색점등이 되게 하는 선택의 결정 방법에 의해서 수반되는 상태는 현재 시간, 현재의 녹색점등 단계, 그리고 각 단계에서 대기중인 차량의 수에 대한 정보를 가지고 있어야 한다.

여기서, 두 개의 단계로 구성된 교차로를 가정해 보면, 시간 t에서 상태 $S(t)$는 다음과 같다.

$$S(t) = (t,\ p(t),\ m(t),\ n(t))$$
$$= (\text{현재시간, 현재의 녹색점등 단계,}$$
$$\text{단계 1에서의 현재 대기차량의 수,}$$
$$\text{단계 2에서의 현재 대기차량의 수})$$

시간 t에서 현재의 상태 $S(t)$로부터 의사결정 $x(t)$를 내리면, 총 차량 지연 $\sum_{phase,\ l}\sum_{vehicles,\ i} D_{i,\ l,\ t}(x(t))$가 초래되고, 그 의사결정은 시간 t_{step} 후에 새로운 상태를 발생시킨다.

$$x(t)=\begin{cases}0 & \text{현재의 단계에 녹색 신호를 유지할 경우}\\ 1 & \text{다른 단계에 녹색 신호를 줄 경우}\end{cases}$$

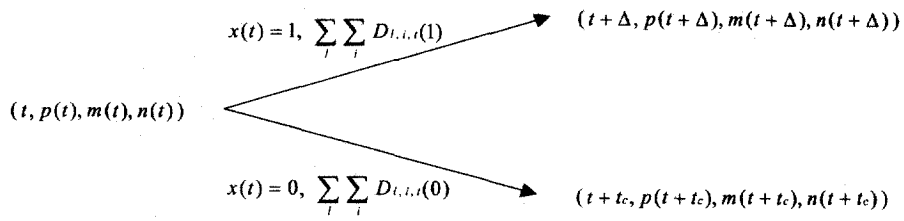


그림 1. 시간 t에서 의사결정 $x(t)$에 의해 변화하는 상태 전이.

$$t_{step}=\begin{cases}\Delta & \text{if } x(t)=0\\ t_c & \text{if } x(t)=1\end{cases} \quad where \quad t_c=t_{YR}+t_{MG}$$

$D_{i,l,t}(x(t))$ = 시간 t에서 의사결정 $x(t)$에 의해 단계 l에서 차량 i에 의해 초래되는 지연

$\sum_{phase,\,l}\sum_{vehicles,\,i} D_{i,l,t}(x(t))$ = 시간 t에서 의사결정 $x(t)$에 의해 서 교차로에서 발생하는 총 지연

<그림 1>은 시간 t에서 의사결정의 역학적인 모습을 보여준다. 기본적으로 차량의 지연은 차량의 도착시간과 출발시간에 따라 다르고, 또한 각 단계의 포화통행률 뿐만 아니라 교차로에 진입해 있는 차량의 단계에 의해서도 영향을 받는다. $D_{i,l,t}(x(t))$의 계산법은 Porch and Lafortune(1997)에 자세히 나타나 있다.

상태와 의사결정 및 그에 따라 수반되는 상태 전이를 결정함으로 인해 동적계획법의 순환함수식을 결정할 수 있다. 마지막으로 순환함수식을 완성하기 위해서는 적당한 경계조건(Boundary Condition)을 결정해야 하며, 이는 전진전개 또는 후진전개 방법을 적용하느냐에 따라서 달라진다.

앞에서 언급한 바와 동적계획법의 순환함수식을 푸는 방법에는 전진전개 방식과 후진전개 방식이 있으며, 일반적으로 계산의 복잡성에 있어서는 서로 대등하다. 그러나 특별한 경우에 있어서는 둘 중 한가지 방법이 다른 방법보다 계산의 효율성 면에서 더 좋을 수 있다. 그러므로 본 논문에서는 두 가지 방법 모두를 설명하고 최적 신호제어와 같은 특별한 문제에 있어서 기대되는 두 기법의 계산상의 효율에 대해 설명할 것이다.

2.2 후진전개 순환 방법

후진전개 순환은 DP 네트워크의 끝 부분부터 시작점까지 역방향으로 문제를 해결해 나간다. 따라서 상태 $(t, p(t), m(t), n(t))$에서의 최적 함수값은 시간 t부터 H까지의 총 지연을 최소화하는 값으로 정의된다. 그러면 이 최적 함수값에 기초한 후진전개 순환식은 다음과 같이 정의된다.

$$f(t,\ p(t),\ m(t),\ n(t)) = \min_{x(t)=0,1}\{\sum_l\sum_i D_{i,l,t}(x(t)) + f(t+t_{step}, p(t+t_{step}), m(t+t_{step}), n(t+t_{step}))\}$$

for $t=\Delta, 2\Delta, \ldots, tc+\Delta, tc+2\Delta, \ldots, 2tc+\Delta, 2tc+2\Delta, \ldots$ and appropriate $p(t), m(t), n(t)$.

계획구간(Planning Horizon)의 H점부터 순환을 시작하기 위해서는 적당한 경계 조건을 정의해 주어야 한다. 의사결정에 의해서 초래될 상태가 계획구간 H 안에 위치할 수도 있고 계획구간 H 밖에 위치할 수도 있기 때문에 경계 조건 식은 다음과 같다(<그림 2(a)> 참조).

$$f(t,\ p(t),\ m(t), n(t)) = \min\{\sum_l\sum_i D_{i,l,t}(x(t)/H)$$

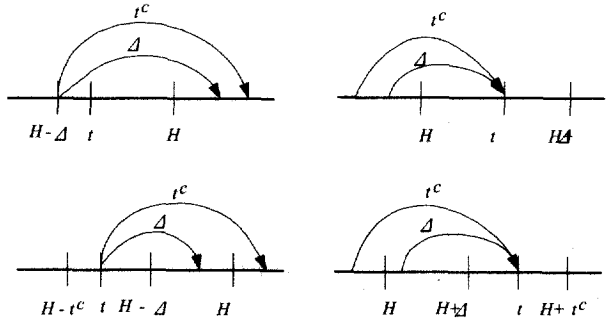
for $H-\Delta \le t \le H$ and the corresponding $p(t), m(t), n(t)$

$$= \min\{\sum_l\sum_i D_{i,l,t}(x(t)=0) f(t+\Delta, p(t+\Delta), m(t+\Delta), n(t+\Delta)), \sum_l\sum_i D_{i,l,t}(x(t)=1/H)\}$$

for $H-tc \le t \le H-\Delta$ and the corresponding $p(t), m(t), n(t)$.

여기서, $\sum_l\sum_i D_{l,i,t}(x(t)/H)$는 시간 H까지 $\sum_l\sum_i D_{l,i,t}(x(t))$의 잘려진 지연이다.

그러므로 경계조건으로부터 출발을 하고, 후진전개 순환을 수행해 나감으로써 시간 0부터 H까지의 최소비용값 $f(0, p(0), m(0), n(0))$를 구할 수 있다. 그러나 이러한 후진전개 순환을 이용하기 위해서는 순환을 시작하기 전에 DP 네트워크의 모든 가능한 상태들에 대한 정보를 알고 있어야 하므로 계산상의 비효율성을 초래한다. 다음에 소개할 전진전개 순환은



(a) 후진전개 방식 (b) 전진전개 방식

그림 2. 경계조건.

모든 상태에 대한 사전지식이 없어도 무관하고, 특정한 가속기법을 이용하여 계산상의 효율을 높일 수 있기 때문에 EDPAS는 전진전개 순환에 바탕을 둔 알고리듬을 사용한다.

2.3 전진전개 순환 방법

전진전개 순환은 DP 네트워크의 시작점부터 계산을 해 나가고, 계획구간 끝점인 H에 이를 때까지 계산을 해나간다. 전진전개 순환에서 현재의 상태 $(t, p(t), m(t), n(t))$에 대한 시간 0부터 t까지 발생하는 총 최소지연은 최적 순환함수값 $f(t, p(t), m(t), n(t))$로 정의한다. 여기서 최소비용을 얻기 위해 선택하는 의사결정은 현재의 상태 $(t, p(t), m(t), n(t))$ "까지"를 의미하지 후진전개 순환에서처럼 현재 상태 "부터"를 의미하는 것이 아니다. $Y(t)$를 현재의 상태 $(t, p(t), m(t), n(t))$로 이끄는 의사결정 집합이라고 하자. 시간 t 바로 전 시점에서 다른 상태들에 같은 의사결정을 내려도 현재의 동일한 상태를 두 개 이상 생성할 수 있다(뒤의 예제를 참조). 따라서, $Y(t)$는 현재 상태를 발생시키는 의사결정이 두 개 이상이 될 수 있음을 주목할 필요가 있다(<그림 3> 참조). 또한 상태 군집화(State Aggregation)를 통해서 중복된 현재 상태의 개수를 줄일 수 있다. 이러한 점을 고려하여 전진전개 순환함수식을 표현하면 다음과 같다.

$$f(t,\ p(t),\ m(t),\ n(t)) =$$
$$\min_{y(t)\in Y(t)}\{f(t-t_y, p(t-t_y), m(t-t_y), n(t-t_y) + \sum_l\sum_i D_{i,l,t-t_c}(y(t))\}$$

for $t = \Delta, 2\Delta, \ldots, t_c+\Delta, t_c+2\Delta, \ldots, 2t_c+\Delta, 2t_c+2\Delta, \ldots$

and appropriate $p(t), m(t), n(t)$,

$$where \quad t_y = \begin{cases} \Delta & \text{if} \quad y(t)=0 \\ t_c & \text{if} \quad y(t)=1 \end{cases}$$

전진전개 순환식은 DP 네트워크의 시점부터 계산을 시작하기 때문에 경계조건은 $f(0, p(0), m(0), n(0))=0$으로 단순하다. 그러나 계획구간을 고려하기 때문에 순환의 마지막에 얻어야하는 최소 총 지연은 다음과 같이 다소 복잡하다(<그림 2-b> 참조).

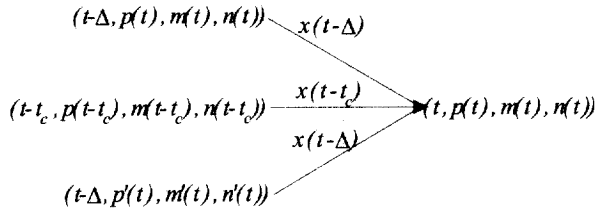


그림 3. 현재 상태로의 전이 및 의사결정.

$$\min[\min_{H\le t\le H+\Delta}\{\min_{y(t)=0,1}\{f(t-t_y, p(t-t_y), m(t-t_y), n(t-t_y)) + \sum_l\sum_i D_{l,i,t-t_y}(y(t)/H)\}\},$$
$$\min_{H+\Delta\le t\le H+t_c}\{\min_{y(t)\neq 0}\{f(t-t_y, p(t-t_y), m(t-t_y), n(t-t_y)) + \sum_l\sum_i D_{l,i,t-t_y}(y(t)/H)\}\}]$$

일반적으로 전진전개 순환식을 해결하는 방법에는 순환고정기법과 리칭기법이 있다(Denardo, 1982). 후진전개 방식의 경우처럼 순환고정기법은 미리 가능한 상태들에 대한 정보를 가지고 있어야 한다. 그러나 리칭기법을 적용하면 DP 네트워크에서의 모든 가능한 상태들에 대한 사전 정보 없이 순환 과정을 수행할 수 있고, 이를 통해 효율적인 알고리듬을 얻을 수 있다. 다음절에서는 불필요한 상태를 제거하며 순환식 계산을 가속화하는 리칭기법을 설명한다.

3. 동적계획 알고리듬

리칭기법은 Denardo(1982)에 자세히 소개되어 있다. 이 기법은 고려중인 현재 상태마다 최적값을 구하기 위해서 바로 전 시점의 모든 상태들의 최적값을 고려하는 방식(순환고정기법)이 아니라, 바로 전 상태마다 생성될 수 있는 현재 상태를 고려하는 방식이다. 따라서 현재 상태들을 새로 만들어 나가면서 순환함수식을 반복적으로 수행한다. 일반적으로 리칭기법의 계산양은 순환고정기법과 비교하여 같으나, 만일 효율적으로 불필요한 현재 상태를 발견하는 가속화 기법을 함께 사용하면 매우 효율적인 알고리듬을 개발할 수 있다. 이 장에서는 두 가지 가속화 기법을 소개한다.

첫번째 가속화 기법은 알고리듬을 통해 현재까지의 가장 적절한 해를 유지함으로써 현재 해보다 값이 크거나 같은 값을 가진 상태들을 제거한다.

또 다른 가속기법은 같은 시간, 같은 단계의 두 상태를 관찰함으로써 가능하다. 예를 들어, 각각 f와 f'의 최적 함수값을 가진 두 상태 $(t, p(t), m(t), n(t))$와 $(t, p(t), m'(t), n'(t))$를 보자. 만일 $m(t)\le m'(t)$, $n(t)\le n'(t)$이고 $f\le f'$이면, 같은 환경 하에서 단계에 더 많은 차량이 존재한다는 것은 미래에 더 많은 지연을 초래한다는 것을 의미하므로 최적값에 영향을 주지 않고 상태 $(t, p(t), m'(t), n'(t))$를 제거할 수 있다.

이러한 상태 제거 방법은 각 상태를 하나 하나 조사하기 전에 열등한 상태들을 제거함으로써 알고리듬을 가속화할 것이다. 다음의 보조 정리와 정리는 앞의 설명을 정형화한 것이다.

보조 정리 1. 두 상태 $(t, p(t), m(t), n(t))$, $(t, p(t), m'(t), n'(t))$를 생각해 보자. 만일 $(m(t), n(t))\le(m'(t), n'(t))$이면, 상태 $(t, p(t), m(t), n(t))$의 차량의 수는 상태 $(t, p(t), m'(t), n'(t))$의 차량의 수의 부분집합이다.

(증명) 각 상태를 형성되는 시간 t에서의 도착 패턴이 같고,

두 상태가 같은 시간이고, 같은 단계이므로, $(m(t), n(t)) \leq (m'(t), n'(t))$는 시간 t에 교차로에 도착하는 차량의 수를 의미하고, 의사결정의 결과가 상태 $(t, p(t), m(t), n(t))$라는 것을 의미하고, 교차로 내의 차량이 더 적은 것을 의미한다. 그러므로 상태 $(t, p(t), m'(t), n'(t))$의 차량의 수는 상태 $(t, p(t), m(t), n(t))$를 포함한다.

보조정리 1.로부터 최적값에 영향을 주지 않고 열등한 상태를 제거할 수 있다는 것을 다음의 정리로부터 입증할 수 있다.

정리 1. 최적 함수값 f와 f'을 가진 두 상태 $(t, p(t), m(t), n(t))$와 $(t, p(t), m'(t), n'(t))$를 고려해보자. 만일 $(m(t), n(t)) \leq (m'(t), n'(t))$이고 $f \leq f'$이면, 최적값(최소값)에 영향을 주지 않고 상태 $(t, p(t), m'(t), n'(t))$를 제거할 수 있다.

(증명) 두 상태가 같은 시간에 같은 단계에 있으므로 미래의 프로세스는 오직 각 단계의 차량의 수에만 영향을 받는다. 보조정리 1.로부터 $(m(t), n(t)) \leq (m'(t), n'(t))$는 상태 $(t, p(t), m'(t), n'(t))$가 각각의 단계에 더 많은 차량이 있다는 것을 의미한다는 것을 안다. 더욱이 $f \leq f'$이기 때문에, 상태 $(t, p(t), m'(t), n'(t))$로부터 파생되는 미래의 프로세스는 어떤 의사결정의 결과에 대해서도 상태 $(t, p(t), m(t), n(t))$로부터 파생되는 프로세스보다 더 많은 지연을 초래한다.

다음은 상태 비교와 현재 해를 통한 두 가지 가속기법을 리칭기법과 함께 적용한 최적 신호정책에 대한 알고리듬을 보일 것이다. 알고리듬 전반에 걸쳐서 다음을 유지할 것이다.

Incumb : 현재 가장 유력한 해(Incumbent)
P : 영구 상태(Permanent State)의 집합
T : 임시 상태(Temporary State)의 집합

P는 현재까지의 최적해가 찾아진 state들의 집합이고, 반면 T는 아직까지 최적해가 찾아지지 않은 상태들의 집합이다. 그러나 임시 상태들도 함수값들을 가지고 있다. 또한, 본 고에서는 편의를 위해 함수값을 상태에 포함시킬 것이다. 따라서, 수정된 상태는 $(t, p(t), m(t), n(t))$로 구성되어 있다.

EDPAS ALGORITHM

Step 0 : Initialization
- $t = 0$, $Incumb = \infty$, $P = \emptyset$, $T = \{(0, p(0), m(0), n(0), 0)\}$

Step 1 : Stopping Criterion
- T로부터 $f \geq Incumb$인 state 제거
- 만일 $T = \emptyset$이면 종료

Step 2 : State Selection
- T로부터 가장 작은 시간 t를 갖는 상태 선택(만일 같으면, 가장 작은 최적값; 만일 또 같으면, 단계에 대기중인 차량의 수가 가장 적은 것을 선택)
- 이것을 현재의 상태 $(t, p(t), m(t), n(t))$ 라고 명명

Step 3 : Pruning by State Comparison (Acceleration 1)
- 만일 현재 상태의 $(m(t), n(t), f)$가 같은 시간, 같은 상태에 있는 P 안의 어떤 상태보다 크거나 같으면 현재의 상태는 제거되고 Step 1로 되돌아간다.

Step 4 : Reaching
- 현재 상태의 각 의사결정으로부터 새로운 상태 $(t', p(t'), m(t'), n(t'), f')$ 생성

$$t' = \begin{cases} \min\{H, t+\Delta\} & \text{if } x(t)=0 \\ \min\{H, t+t_c\} & \text{if } x(t)=1 \end{cases}$$

$$p(t') = \begin{cases} p(t) & \text{if } x(t)=0 \\ p^c(t) & \text{if } x(t)=1 \end{cases}$$

where $p^c(t) \equiv$ *complement of* $p(t)$

$$m(t') = m(t) + \#Arrv1(t, t') - \#Dept1(t, t')$$

$$n(t') = n(t) + \#Arrv2(t, t') - \#Dept2(t, t')$$

$$f = f + \begin{cases} \sum_l \sum_i D_{l,i,t}(x(t)) & \text{if } t' < H \\ \sum_l \sum_i D_{l,i,t}(x(t)/H) & \text{if } t' = H \end{cases}$$

$Arrvi(t, t')$는 단계 i에서 시간 (t, t')동안 그 단계에 도착한 차량의 수를 의미하며, $Depti(t, t')$는 단계 i에서 시간 (t, t')동안 그 단계에서 떠난 차량의 수를 뜻한다.
- 현재의 상태를 P에 포함시킨다.

Step 5 : Pruning by Incumbent (Acceleration 2)
(step 4에서 계산된 각각의 상태에 대해서)
- 만일 $t' = H$ 이고 $f' < Incumb$ 이면 $Incumb \leftarrow f'$
- 만일 $f' \geq Incumb$이면 새로이 생성된 상태를 제거한다.

Step 6 : State Aggregation
(step 4에서 계산된 각각의 상태에 대해)
- 만일 T에 상태 $(t', p(t'), m(t'), n(t'), f'')$이 존재하고 같은 시간, 같은 단계에 같은 수의 차량의 수가 대기중인 상태 $(t', p(t'), m(t''), n(t''), f')$가 step 4에 의해 발생하면 두 상태를 하나의 상태 $(t', p(t'), m(t'), n(t'), \min(f', f''))$ $n(t'), \min(f', f''))$로 군집시킨다.

- 만일 $t' < H$이면 군집된 상태를 T에 포함시킨다.
- 만일 $t' = H$이면 군집된 상태를 P에 포함시킨다.
- Step 1로 간다.

이 알고리듬은 임시 상태 집합 T가 ∅이 되면 종료하는데 그것은 DP 네트워크의 모든 상태들에 최적 함수값이 할당되었거나 제거되었음을 의미한다. 그러면 최적해는 계획구간에서 최소 함수값을 가진 집합 P로부터 상태들을 고름으로써 얻을 수 있다. Step 3과 Step 5는 상태들을 비교하고, 현재 해를 이용하여 함수값들을 비교함으로써 불필요한 상태들을 제거하는 가속절차이다. Step 6은 Step 4의 리칭 절차에 의해 생성된 트리로부터 얻어진 상태에 대한 군집 절차이다.

4. 예 제

본 절에서는 앞 절에서 소개한 리칭기법을 이용하여 예제 문제를 푼다. 본 예제는 Sen and Head(1997)에서 발췌하였다. 향후 10단위의 시간동안의 도착패턴은 <그림 4>에 나타나 있다.

교차로는 A, B, C 3개의 단계로 구성되어있다. $\Delta = 2$, $t_{MG} = 2$, $t_{YG} = 1$이고 본 예제에서 포화통행률은 ∞로 가정하고, 의사결정변수 $x(t)$는 A, B, C로 한다. 녹색점등이 되는 마지막 단계는 최소 점등시간을 필요로 하지 않는다. 여기서 우리는 10단위의 시간동안 도착하는 차량들에 대한 총 지연의 최소값을 구하려 한다.

본 예제에서는 차량들의 지연은 속도 제한을 무시하고 단순히 교차로의 진입시간과 교차로를 떠나는 시간의 차이로 계산한다. 그것은 차량들 앞의 차량의 유무에 관계없이 해당 단계에 녹색신호가 점등되면 교차로를 떠난다는 것을 의미한다.

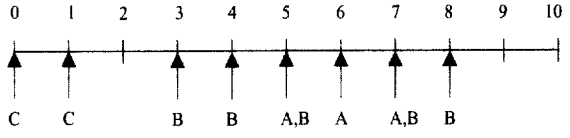


그림 4. 차량 도착시간 패턴.

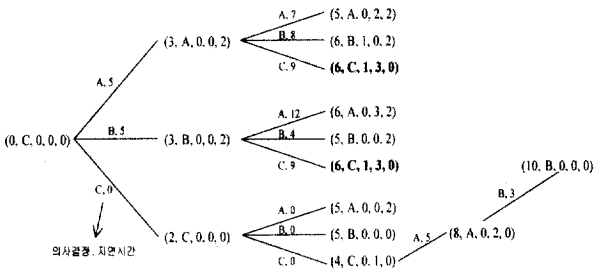


그림 5. 의사 결정 트리 일부분.

그림 6. 최적 신호제어 계획.

<그림 5>는 본 예제에 대한 전체 의사결정 트리에서 두 상태 $(6, C, 1, 0, 3)$을 군집함으로써 이 트리를 DP 네트워크로 변형시킬 수 있다.

<표 1~3>은 본 예제에 적용된 알고리듬의 전체 절차를 보여준다. 그러므로 <표 1~3>으로부터 최적해는 <그림 6>에 묘사된 바와 같이 2단위의 녹색점등 시간을 단계 C에 먼저 할당하고, 그 다음에 2단위의 녹색점등 시간을 단계 B에 할당하고, 그 다음에 2단위의 녹색점등 시간을 단계 A에 할당하고 1단위의 녹색점등 시간을 단계 B에 할당한다(단계가 바뀔 때 황색-적색 시간을 포함시켜서 할당한다). 녹색점등의 조합으로 인한 총 지연은 8이다. 앞서 가정한 바와 같이 마지막 녹색점등 단계는 최소 녹색점등 시간을 필요로 하지 않는 다는 것에 유의해야 한다.

5. 결 론

본 논문에서는 차량들이 교차로를 통과할 때 발생하는 총 지연 시간을 최소화하기 위한 단일 교차로에서의 신호정책에 대한 효율적인 동적계획 알고리듬을 개발했으며, 이를 EDPAS라 부른다. COP을 포함해서 동적계획 접근법을 이용하는 다른 알고리듬들과 비교해 볼 때 EDPAS는 어떤 단계 순서를 가진 교차로에서도 총 지연을 최소화하기 위한 정확한 최적 신호정책을 제공하는 최초의 알고리듬이다. COP는 각 단계를 통해 "고정된 순서의 각 단계에 얼마나 많은 녹색점등 시간을 할당할 것인가"를 결정하는 반면 EDPAS는 미리 정한 양의 시간에 대하여 "어떤 단계에 녹색점등을 할당할 것인가"를 결정한다.

EDPAS는 적당한 녹색점등 시간을 선택함으로써 COP의 의사결정 하에서 어떤 신호정책을 주기 위해 쉽게 변경할 수 있다. EDPAS는 교차로에서 계획구간의 끝에 도착했을 때 차량이 교차로에 남아있는 결과를 무시함으로써 신호정책에 대한 고정된 계획구간을 가정한다. 계획구간 동안 교차로에 차량이 남게 하는 의사결정의 결과는 그 불완전성으로 인해 완벽하지 못하므로 향후에 연구되어야 할 분야이다.

EDPAS의 계산상의 가속성은 좋은 현재 해에 의해 증가되므로 알맞게 좋은 시작 해를 찾는 것이 향후 연구과제가 될 것이다. 또 다른 중요한 연구과제는 각각의 교차로에서의 적응 신호정책으로 교통 네트워크에서 조합된 신호정책을 뽑아내는 것이다. 각 교차로에서의 신호정책의 조화가 없이는 독립적인 교차로의 신호정책이 아무리 좋다고 하여도 전체 네트워크에서는 큰 의미를 발휘할 수 없다.

표 1. 반복 절차에 대한 예

Step	Permanent Nodes	Temporary Nodes	Current Node	Reaching	Acceleration	Incumb
0	ϕ	(0,C,0,0,0,0)	(0,C,0,0,0,0)	(3,A,0,0,2,5) (3,B,0,0,2,5) (2,C,0,0,0,0)		∞
1	(0,C,0,0,0,0)	(2,C,0,0,0,0) (3,A,0,0,0,0) (3,B,0,0,0,0)	(2,C,0,0,0,0)	(5,A,0,2,0,3) (5,B,0,0,0,0) (4,C,0,1,0,1)		∞
2	(0,C,0,0,0,0) (2,C,0,0,0,0)	(3,A,0,0,2,5) (3,B,0,0,2,5) (4,C,0,1,0,1) (5,B,0,0,0,0) (5,A,0,2,0,3)	(3,A,0,0,2,5)	(5,A,0,2,2,12) (6,B,1,0,2,13) (6,C,1,3,0,14)		∞
3	(0,C,0,0,0,0) (2,C,0,0,0,0) (3,A,0,0,2,5)	(3,B,0,0,2,5) (4,C,0,1,0,1) (5,B,0,0,0,0) (5,A,0,2,2,12) (6,B,1,0,2,13) (6,C,1,3,0,14)	(3,B,0,0,2,5)	(6,A,0,3,2,17) (5,B,0,0,2,9) (6,C,1,3,0,14)	Aggregation	∞
4	(0,C,0,0,0,0) (2,C,0,0,0,0) (3,A,0,0,2,5) (3,B,0,0,2,5)	(4,C,0,1,0,1) (5,B,0,0,0,0) (5,A,0,2,0,3) (5,B,0,0,2,9) (5,A,0,2,2,12) (6,B,1,0,2,13) (6,C,1,3,0,14) (6,A,0,3,2,17)	(4,C,0,1,0,1)	(7,A,0,3,0,9) (7,B,2,0,0,6) (6,C,1,3,0,7)	Aggregation	∞
5	(0,C,0,0,0,0) (2,C,0,0,0,0) (3,A,0,0,2,5) (3,B,0,0,2,5) (4,C,0,1,0,1)	(5,B,0,0,0,0) (5,A,0,2,0,3) (5,B,0,0,2,9) (5,A,0,2,2,12) (6,C,1,3,0,7) (6,B,1,0,2,13) (6,A,0,3,2,17) (7,B,2,0,0,6) (7,A,0,3,0,9)	(5,B,0,0,0,0)	(8,A,0,2,0,5) (7,B,2,0,0,3) (8,C,3,2,0,10)	Aggregation	∞
6	(0,C,0,0,0,0) (2,C,0,0,0,0) (3,A,0,0,2,5) (3,B,0,0,2,5) (4,C,0,1,0,1) (5,B,0,0,0,0)	(5,A,0,2,0,3) (5,B,0,0,2,9) (5,A,0,2,2,12) (6,C,1,3,0,7) (6,B,1,0,2,13) (6,A,0,3,2,17) (7,B,2,0,0,3) (7,A,0,3,0,9) (8,A,0,2,0,5) (8,C,3,2,0,10)	(5,A,0,2,0,3)	(7,A,0,3,0,9) (8,B,3,0,0,12) (8,C,3,4,0,19)	Aggregation Aggregation	∞
7	(0,C,0,0,0,0) (2,C,0,0,0,0) (3,A,0,0,2,5) (3,B,0,0,2,5) (4,C,0,1,0,1) (5,B,0,0,0,0) (5,A,0,2,0,3)	(5,B,0,0,2,9) (5,A,0,2,2,12) (6,C,1,3,0,7) (6,B,1,0,2,13) (6,A,0,3,2,17) (7,B,2,0,0,3) (7,A,0,3,0,9) (8,A,0,2,0,5) (8,C,3,2,0,10) (8,B,3,0,0,12) (8,C,3,4,0,19)	(5,B,0,0,2,9) (5,A,0,2,2,12) (6,C,1,3,0,7)	(9,A,0,5,0,21) (9,B,3,0,0,18) (8,C,3,4,0,19)	Pruned Pruned Aggregation	∞

표 1. 반복 절차에 대한 예(계속)

Step	Permanent Nodes	Temporary Nodes	Current Node	Reaching	Acceleration	Incumb
8	(0,C,0,0,0,0) (2,C,0,0,0,0) (3,A,0,0,2,5) (3,B,0,0,2,5) (4,C,0,1,0,1) (5,B,0,0,0,0) (5,A,0,2,0,3) (6,C,1,3,0,7)	(6,B,1,0,2,13) (6,A,0,3,2,17) (7,B,2,0,0,3) (7,A,0,3,0,9) (8,A,0,2,0,5) (8,C,3,2,0,10) (8,B,3,0,0,12) (8,C,3,4,0,19) (9,B,3,0,0,18) (9,A,0,5,0,21)	(6,B,1,0,2,13)	(9,A,0,2,2,24) (8,B,3,0,2,22) (9,C,3,2,0,26)		∞
9	(0,C,0,0,0,0) (2,C,0,0,0,0) (3,A,0,0,2,5) (3,B,0,0,2,5) (4,C,0,1,0,1) (5,B,0,0,0,0) (5,A,0,2,0,3) (6,C,1,3,0,7) (6,B,1,0,2,13)	(6,A,0,3,2,17) (7,B,2,0,0,3) (7,A,0,3,0,9) (8,A,0,2,0,5) (8,C,3,2,0,10) (8,B,3,0,0,12) (8,C,3,4,0,19) (8,B,3,0,2,22) (9,B,3,0,0,18) (9,A,0,5,0,21) (9,A,2,2,0,24) (9,C,2,5,0,31)	(6,A,0,3,2,17)	(8,A,0,4,0,28) (9,B,2,0,2,24) (9,C,2,5,0,31)		∞
10	(0,C,0,0,0,0) (2,C,0,0,0,0) (3,A,0,0,2,5) (3,B,0,0,2,5) (4,C,0,1,0,1) (5,B,0,0,0,0) (5,A,0,2,0,3) (6,C,1,3,0,7) (6,B,1,0,2,13) (6,A,0,3,2,17)	(7,B,2,0,0,3) (7,A,0,3,0,9) (8,A,0,2,0,5) (8,C,3,2,0,10) (8,B,3,0,0,12) (8,C,3,4,0,19) (8,B,3,0,2,22) (8,A,0,4,2,28) (9,B,3,0,0,18) (9,A,0,5,0,21) (9,A,2,2,0,24) (9,B,2,0,2,24) (9,C,3,2,0,26) (9,C,2,5,0,31)	(7,B,2,0,0,3)	(10,A,0,2,0,11) (9,B,3,0,0,10) (10,C,3,2,0,17)	Aggregation ≥ Incumb ≥ Incumb ≥ Incumb ≥ Incumb ≥ Incumb ≥ Incumb ≥ Incumb ≥ Incumb ≥ Incumb ≥ Incumb ≥ Incumb ≥ Incumb	11
11	(0,C,0,0,0,0) (2,C,0,0,0,0) (3,A,0,0,2,5) (3,B,0,0,2,5) (4,C,0,1,0,1) (5,B,0,0,0,0) (5,A,0,2,0,3) (6,C,1,3,0,7) (6,B,1,0,2,13) (6,A,0,3,2,17) (7,B,2,0,0,3) (10,A,0,2,0,11)	(7,A,0,3,0,9) (8,A,0,2,0,5) (8,C,3,2,0,10) (9,B,3,0,0,10)	(7,A,0,3,0,9)	(9,A,0,5,0,19) (10,B,1,0,0,15) (10,C,1,5,0,20)	≥ Incumb ≥ Incumb ≥ Incumb	11

표 1. 반복 절차에 대한 예(계속)

Step	Permanent Nodes	Temporary Nodes	Current Node	Reaching	Acceleration	Incumb
12	(0,C,0,0,0,0) (2,C,0,0,0,0) (3,A,0,0,2,5) (3,B,0,0,2,5) (4,C,0,1,0,1) (5,B,0,0,0,0) (5,A,0,2,0,3) (6,C,1,3,0,7) (6,B,1,0,2,13) (6,A,0,3,2,17) (7,B,2,0,0,3) (10,A,0,2,0,11)	(8,A,0,2,0,5) (8,C,3,2,0,10) (9,B,3,0,0,10)	(8,A,0,2,0,5)	(10,A,0,2,0,11) (10,B,0,0,0,8) (10,C,0,2,0,11)	≥ Incumb ≥ Incumb	8
13	(0,C,0,0,0,0) (2,C,0,0,0,0) (3,A,0,0,2,5) (3,B,0,0,2,5) (4,C,0,1,0,1) (5,B,0,0,0,0) (5,A,0,2,0,3) (6,C,1,3,0,7) (6,B,1,0,2,13) (6,A,0,3,2,17) (7,B,2,0,0,3) (10,A,0,2,0,11) (8,A,0,2,0,5) (10,B,0,0,0,8)	(8,C,3,2,0,10) (9,B,3,0,0,10)			≥ Incumb ≥ Incumb	8
9	(0,C,0,0,0,0) (2,C,0,0,0,0) (3,A,0,0,2,5) (3,B,0,0,2,5) (4,C,0,1,0,1) (5,B,0,0,0,0) (5,A,0,2,0,3) (6,C,1,3,0,7) (6,B,1,0,2,13) (6,A,0,3,2,17) (7,B,2,0,0,3) (10,A,0,2,0,11) (8,A,0,2,0,5) (10,B,0,0,0,8)	ϕ				8

참고문헌

Denardo, E. V. (1982), *Dynamic Programming: Models and Applications*, Prentice Hall.

Gartner, N. H. (1983), OPAC: a demand-responsive strategy for traffic signal control, *Transportation Research Record*, **906**, 75-81.

Henry, J. J., Farges, J. L. and Tuffal, J. (1983), The PRODYN real time traffic algorithm, 4th IFAC-IFIP-IFORS Conference on Control in Transportation System, Baden Baden, Germany.

Hunt, P. B., Robertson, D. I., Bretherton, R. D. and Winton, R. I. (1981), SCOOT - a traffic responsive method for coordinating signals, *Laboratory Report No. LR 1014*, Transportation and Road Research, Crowthorne, Berkshire, England.

Mauro, V. and DiTronto, D. (1990), UTOPIA, *Proceedings of the 6th IFAC/IFIP/IFORS Symposium on Control Computers and Communication in Transportation*, 12, 245-252.

Porche, I. and Laforune, S. (1997), Adaptive look-ahead optimization of traffic signals, *Technical Report No. CGR 97-11*, Department of Electric Engineering and Computer Science, The University of Michigan.

Sen, S. and Head, K. L. (1997), Controlled optimization at an intersection, *Transportation Science*, 31(1), 5-17.

Sims, A. G. (1979), The sydney coordinated adaptive traffic system, *Urban Transport Division of ASCE Proceedings, Engineering Foundation Conference on Research Directions in Computer Control of Urban Traffic Systems*, 12-27. New York.

Webster, F. V. (1958), Traffic signal settings, *Road Research Technical Paper*, 39, HMSO, London.

Yargar, S. and Han, B. (1994), A procedure for real-time signal control that considers transit interference and priority, *Transportation Research B*, **28**, 315-331.