JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 26 No. 12, pp. 29-43, December 2021
https://doi.org/10.9708/jksci.2021.26.12.029

An Approach of Solving the Constrained Dynamic Programming - an Application to the Long-Term Car Rental Financing Problem

Tae Joon Park*, Hak-Jin Kim*, Jinhee Kim*

*Student, School of Business, Yonsei University, Seoul, Korea
*Professor, School of Business, Yonsei University, Seoul, Korea
*Student, School of Business, Yonsei University, Seoul, Korea

[Abstract]

In this paper, a new approach to solve the constrained dynamic programming is proposed by using the constraint programming. While the conventional dynamic programming scheme has the state space augmented with states on constraints, this approach, without state augmentation, represents states of constraints as domains in a contraining programming solver. It has a hybrid computational mechanism in its computation by combining solving the Bellman equation in the dynamic programming framework and exploiting the propagation and inference methods of the constraint programming. In order to portray the differences of the two approaches, this paper solves a simple version of the long-term car rental financing problem. In the conventional scheme, data structures for state on constraints are designed, and a simple inference borrowed from the constraint programming is used to the reduction of violation of constraints because no inference risks failure of a solution. In the hybrid approach, the architecture of interface of the dynamic programming solution method and the constraint programming solution method is shown. It finally discusses the advantages of the proposed method with the conventional method.

▸ **Key words**: the dynamic programming, constraint programming,
the long-term car rental financing problem, computation theory, optimization

[요 약]

본 연구에서 제약식프로그래밍을 이용하여 제약식 있는 동적계획법 모형을 푸는 한 방법을 제시한다. 현재 제약식 있는 동적계획법을 다루는 방법은 각 단계별 제약식들의 상태를 일반적인 동적계획법의 상태공간에 추가하여 마치 제약식이 없는 동적계획법 방식을 적용하는데 반해, 이 연구에서 제시하는 방식은 제약식의 상태가 제약식프로그래밍의 변수 도메인으로 표현되고 저장된다. 계산에 있어서도 일반적인 동적계획법의 벨만방정식의 해법과 함께 제약식을 다루기 위한 제약식프로그래밍의 확산-추론 방법을 사용하는 하이브리드 방식을 따른다. 이 두가지 방식의 비교를 위해 특별히 장기 자동차 렌탈 문제를 제시하고 이 문제의 단순화된 모형을 중심으로 다른 방식으로 해결하는 과정을 보고 그 장단점을 논한다.

▸ **주제어**: 동적계획법, 제약식프로그래밍, 장기 자동차 렌탈 문제, 계산 이론, 최적화

• First Author: Tae Joon Park, Corresponding Author: Hak-Jin Kim
*Tae Joon Park (xoxoqkr@naver.com), School of Business, Yonsei University
*Hak-Jin Kim (hakjin@yonsei.ac.kr), School of Business, Yonsei University
*Jinhee Kim (masulsa96@yonsei.ac.kr), School of Business, Yonsei University
• Received: 2021. 10. 27, Revised: 2021. 12. 06, Accepted: 2021. 12. 06.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

동적계획법은 이산 시스템의 해를 구하거나, 정수 해를 만들기 위해서 사용된다. 또 동적 계획법은 문제의 상태가 시간에 따라 변하는 수리 모형으로 표현하는데 적합하다. 잘 알려진 배낭채우기 문제(Knapsack problem)와 같이, 문제의 모형 표현들이 분리될 수 있는(separable) 단순한 수리 모형은 동적 계획법으로 변환될 수 있다. 하지만 제약식의 수가 많은 복잡한 수리모형의 경우에는 동적 계획법으로의 전환이 어려운데, 이는 단순한 문제와 비교하였을 때, 상대적으로 상태 공간을 정의하는 것이 어렵기 때문이다. 즉 문제 내의 가능한 상태 수가 증가함으로써, 상태 공간이 증가하기 때문이다. 본 논문에서는 제약식프로그래밍을 사용하여 제약식이 많은 수리모형을 동적 계획법으로 전환하는 어려움을 해결할 수 있음을 장기 자동차 렌털 문제에 대한 적용을 통해 보이고자 한다.

제약식이 있는 동적계획법은 [8]에서 최적 정책의 존재성과 구조에 대해 증명되었고 이를 파레토 최적문제로 보고 그에 대한 성질들을 얘기하고 있고 [7]은 제약식과 불확실성을 지닌 최적화문제를 동적계획법과 모수계획법으로 푼다. [14]는 제약식 있는 동적계획법의 합산 구조에 대한 전제를 일반화하여 결합확률 함수를 지표 확률변수를 이용하여 다루는 방법을 재설정하는 방법을 이용한다. [12]는 수력발전의 운영전략을 위해 제약식있는 동적계획법으로 설정하고 이를 풀고 있다.

장기 자동차 렌털 비스니스에 대해 간단히 설명한다면, 먼저 렌터카 사업은 계약기간에 따라 단기 렌터카와 장기 렌터카 사업으로 구분된다. 단기 렌터카는 주로 계약기간이 2주일을 넘지 않으면서 비즈니스나 관광을 목적으로 하는 여행객들이 주로 이용하는 반면, 장기 렌터카는 계약기간이 2~3년 이상의 장기간 계약을 목표로 하는 자동차의 소유에서 오는 불편을 피하기 위한 고객들을 주 목표로한다. 그 고객들은 새 자동차를 이용할 수 있다는 점, 자동차의 소유 및 유지를 위한 여러 부가적인 비용과 노력없이 렌트비만 지불하면 렌터카 회사에서 수리 유지 등 여러 편이를 제공한다는 점, 계약 종료를 통해 자동차의 처분에 드는 수고를 덜 수 있다는 점 등의 여러 장점으로 자동차의 소유보다는 이용의 편이성이 중요한 기업 고객 등에 의해 많이 이용된다. 문헌에는 주로 단기 렌터카 사업에 대한 관련한 연구가 많이 언급되어 있고 이와 관련된 연구로는 현금관리([2,6,15,16,20])와 수익관리 ([4,5,11,13,19,21,22])와 밀접한 관련을 갖는다. 이 연구는 장기 자동차 렌터카 사업의 운영에 대한 재정 문제를 수리적인 모형으로 접근하고 있다. 특히 김현호와 김학진의 연구 [1]에서 문제에 대한 수리계획법적인 접근은 구입하는 차량의 대수를 실수로 완화(relax)하여 풀이하였지만, 이 문제에 대해 좀더 만족스러운 결과로서 이끌어내기를 원한다면 차량 대수에 대한 변수를 정수화하여 모형을 풀어야 한다. 본 연구에서는 이를 동적계획법의 접근 방식을 이용하여 정수해를 탐색하는 방식으로 문제에 접근한다. 이를 통해 이 논문에서 제시하고자 하는 전통적인 동적계획법의 틀과 제약식프로그래밍의 협력 방식을 보여주기 위해, 장기 자동차렌탈 문제의 원모형에 나타난 모든 제약식을 이용하기 보다는 이를 단순화한 모형에서 제안된 방법이 어떻게 적용되는지를 통해 제안된 방법을 설명하고 그 특성들을 살피고자 한다.

이 논문에서 제안하는 방법으로는 동적계획법의 틀에서 제약식프로그래밍의 방식을 결합하는 것이다. [17]에 의하면 제약식프로그래밍은 인공지능 분야에서 문제 해결을 위해 사용하는 개념적인 접근법이다. 제약 프로그래밍은 문제가 해결되는 주제(도메인) 중심이기 때문에, 각 주제를 중심으로 다양한 제약식프로그래밍 기술들이 연구되어 왔고, 이를 기반으로 각 주제에 맞추어 개발된 다양한 제약식프로그래밍 솔버가 있다. 본 연구는 정수해를 가지는 제약식을 다룰 수 있는 유한정의역 제약식프로래밍(finite domain constraint programming, FD) ([18])을 사용한다. 이는 우리의 장기 자동차 렌탈 재정 문제가 정수 해를 가지는 제약식들로 정의되기 때문이다. 다수의 제약식이 주어졌을 때, FD는 정수 변수에 대해 정의된 값들의 도메인을 점진적으로 줄여나가며 변수들에 대한 일관 해를 찾는 확산법(propagation)을 적용한다. 해를 찾는 과정에서 솔버가 제약식을 만족하지 못하는 일관적(consistent)이지 않은 부분해를 만나게 되면(No-good), 그 부분해의 선택을 도메인에서 제거하여 추후의 연산과정에서의 실패를 방지하게 된다. 이와 같은 계산방식은 동적계획법과 결합하여 행의 탐색 과정에 필요한 노력을 줄여줄 수 있다. [3]과 [31]은 이를 이용한 응용을 보여준다.

본 논문은 먼저 관련 연구로 시작하여 장기 자동차 렌탈 금융 문제를 설명하고 연구 모형을 살펴본다. 다음으로 제약식을 가진 동적 계획법 모형의 특징에 대해 논하고 이를 풀기위한 방법으로서 전통적 동적 계획법 틀 안에서의 접근 방법 두가지를 제시하고 동적 계획법과 제약식프로그래밍을 혼합한 방법을 설명한다. 이후 랜덤으로 생성한 인스턴스들을 이용한 계산 실험 결과와 논의, 그리고 결론을 맺는다.

II. The problem model and its simplification

1. Definition of the problem

이 논문에서는 동적계획법과 제약식프로그래밍의 혼합 방법을 자동차렌탈 문제에 적용하여 설명한다. 즉 문제의 상황에 대한 인식이 혼합 방법의 설명에 대한 이해에 요구되므로 이 장에서는 장기 자동차렌탈 문제에 대한 구체적인 정의와 모형을 논의한다.

차량 렌탈 회사가 고객과 차량 렌탈 계약을 맺으면, 장기 차량 렌탈이 시작된다. 이후 회사는 고객의 요구 조건에 맞는 차량을 구매하고, 해당 차량을 고객에게 렌탈 계약과 함께 인도한다. 일반적으로 차량 렌탈 회사는 다수의 차량을 한 번에 구매할 수 있는 자금이 없기 때문에, 이들은 은행과 자동차 판매를 촉진하기 위해 자금을 빌려주는 금융 캐피탈 등으로부터 대출을 받게 된다. 이러한 금융 기업들은 렌탈 회사가 그들과 제휴관계를 맺은 제조사의 자동차를 구입하는 경우에는 낮은 이자와 같은 유인책을 제공하게 된다. 또한 각 금융 회사는 채무 불이행의 위험을 줄이기 위해서, 개별 사업체에 대한 대출 금액 상한을 가지고 있으며, 대출 금액에 따라 이자율을 다르게 적용한다. 금융 회사의 모든 대출 상한액과 이자율을 알고 있는 상태에서, 어떤 금융 회사로부터 대출을 받아 자동차를 구입할지는 차량 렌탈 회사의 이익과 직결되는 중요한 문제이다.

고객은 장기 렌트카 회사와 계약 할 때 렌트카 모델, 차량 수, 계약 기간 및 계약 유형을 결정한다. 계약 유형은 "반환"과 "인수" 로 나뉘는데, "반환"은 차량 렌탈 기간의 종료 후 차를 다시 렌탈사에 반환하는 형태이며, "인수"는 렌트 계약 종료 후 계약에서 렌탈사가 제안한 가격으로 렌트카를 인수하는 형태이다. 계약 시 고객은 렌탈 하려는 자동차 가격의 α %로 보증금을 지불하고, 이 보증금은 자동차 제조업체에서 해당 자동차를 구매하는 데 사용된다. 만약, 이 보증금이 렌탈 회사가 등록한 캐쉬백 보상 프로그램의 신용 카드로 지불되는 경우 회사는 해당 신용 카드 회사로부터 보상을 받을 수 있다. 일반적으로 캐쉬백은 신용 카드 사용액의 b %이다. 자동차 구매 가격의 나머지 금액 (1-α) %는 금융 회사에서 대출 한도와 이자율을 고려하여 대출받는다.

렌트카 회사가 벌어들이는 수익은 1) 계약 기간 h 동안 계약 고객으로부터 매월 받는 렌탈 수수료, 2) 계약 종료 후 렌트카 재판매 가격, 3) 신용 카드 회사의 캐쉬백 보상, 4) 회사의 자산으로 등록된 차량의 감가상각에 의한 세금 감면으로 구성된다. 재판매 가격은 렌탈 계약이 "반환"인지 "인수"인지에 따라 달라진다. "반환"의 경우 중고차 시장에서의 시장 가격에 따라 재판매 가격이 결정되며, "인수"의 경우 가격은 고객과 체결한 계약 상의 금액에 따라 결정된다. 따라서 렌트카 회사가 지불하는 비용은 차량에 대한 예치금, 회사가 예치금을 제외한 금액에 대하여 금융 회사로부터 차입한 대출금과 그에 대한 이자로 구성된다.

Table 1은 문제의 모형에서 사용하고 있는 기호들의 의미들을 보여주고 있다.

Table 1. Symbols in the model

t	time (month)
τ	the lead time till the car delivery to a customer
c	a financing firm
g	the type of a rent car
T	the horizon of planning
C	the number of financing firms
G	the number of kinds of rental
h	contract period
$x_{t,g,c}$	the number of purchased cars of the kind g using the fund from c at time t (integer variable)
$R_{t,g}$	revenue from the type g at time t
$F_{t,g,c}$	the interest payment during period h
$P^{I_{t,g}}$	the purchasing price of type g at time t
$C^{R_{t,g,c}}$	the interest rate of type g for the fund c at time t
$P^{R_{t,g}}$	the resale price of type g at time t
$A_{t,g}$	the tax reduction of type g
$I_{t,g}$	the inventory cost of type g at time t
$D_{t,g}$	the demand for type g at time t
ρ	a computational factor (=max{1, t-h+1})
$C^{L_{t,c}}$	the loan limit of fund c at time t
M	the capacity of inventory for cars
R_{tax}	tax rate
α	the rate of prepayment in contracts (=0.2)
b	the rate of cash back from financing firms (=0.2)
S_g	the salvage rate for type g after the end of contract

장기 차량 렌탈 금융 문제의 수리 모형은 식 (1)-(9) 와 같으며, 이 때 사용되는 변수는 [표 1]을 따른다. 수리 모형의 목적은 T 기간 동안의 이익을 최대화하는 것이다. 하지만 목적식의 계산은 T-h 기간에 대해서만 이루어지는데, 이는 고객과 회사의 계약 기간이 상수 h 로 고려되기 때문이다. 이는 마지막 h-1 시점에 진행된 렌탈 거래에 대한 계산이 T 시간 내에서 끝나지 않기 때문에 목적식에서 완전하게 고려될 수 없음을 의미한다. 하지만 계산의 단순화를 위해, 이러한 경우는 고려하지 않도록 한다. 식 (1)은 T-h 동안 발생하는 차량 판매와 금융 활동에서 발생하는 순이익을 계산한다. $hR_{t+\tau,g}$ 는 타입 g차량을 h기간 동

안 렌탈해 주었을 때 고객이 회사에 지불하는 금액이다. 리드타임 τ는 고객과의 계약 체결시점 t 와 이후 렌탈 차량이 실제로 도착하는 시점 t+τ 까지에 걸리는 시간이다. 첫 번째 렌탈 수수료는 차량이 고객에게 지급된 τ 개월 이내에 지급되며, 총 렌탈 수수료 이익은 $R_{t+\tau,g}$ 에 계약 기간인 h 를 곱한 형태로 표현된다. 식 (2) 는 렌탈 회사가 금융 회사 c 로부터 차량 g 를 사기위해 대출한 금액에 대한 자본 비용과 이에 따른 이자 $F_{t,g,c}$ 를 정의한다. 렌탈 회사는 금융 회사로부터 차량 g 구입 가격인 $P^{I_{t,g}}$에서 고객이 지급한 보증금 α을 제외한 1-α 만큼을 대출받게 된다. 금융 회사로부터 대출받은 금액에 대한 이자 비용은 대출금에 이자율을 곱한 값으로 계산된다. 캐피탈 금융 회사가 렌탈 회사에게 지급하는 캐쉬백은 고객이 납부한 차량의 금액에 캐피탈 회사의 캐쉬백 비율을 곱한 $abP^I_{t,g}$ 로 계산된다. 식 (3)에서 정의되는 종료시점의 세금 감면은 세율에 $(1-s_g)P^I_{t,g}$ 의 감가상각 금액을 곱한 값이다. 이 때 s_g 는 차량 g 의 잔존가치를 의미한다. 실제 잔존가치는 $P^R_{t,g}$ 는 계약 종료 이후에 계산된다. 이를 통해 렌탈 회사는 이익으로 렌탈 수익, 캐쉬백, 세금감면, 구입된 차량의 잔존가치를 비용으로 차량 구입 가격, 금융 비용, 고객 보증금을 식별할 수 있다. 식 (6)은 재고 비용으로 이는 단위 개월 당 재고 비용인 $I_{t,g}$ 와 재고량인 $Y_{t,g}$ 의 곱으로 정의된다.

$$\max \quad Z=\sum_{t=1}^{T-h}\sum_{g=1}^{G}\left\{\sum_{c=1}^{C}\left[hR_{t+\tau,g}-F_{t,g,c}-(1-\alpha)P^{I_{t,g}}\right]\right\} \tag{1}$$
$$-\alpha P^I_{t,g} \;+\; \alpha bP^I_{t,g} \;+\; A_{t,g}+\; P^R_{t,g}\Big]\; x_{t,g,c}-I_{t,g}Y_{t,g}\Big\}$$

$$\text{s.t} \quad F_{t,g,c}=(1-\alpha)P^I_{t,g}C^R_{t,g,c}, \qquad \forall\; t,\; c,\; g \tag{2}$$

$$A_{t,g}=R_{tax}(1-s_g)P^I_{t,g}, \qquad \forall\; t,\; g \tag{3}$$

$$\sum_{g=1}^{G}\left[P^I_{t,g}x_{t,g,c}-\frac{1}{h}\sum_{t=\rho}^{t-1}P^I_{\bar{t},g}x_{\bar{t},g,c}\right]\leqq\;\frac{C^L_{t,c}}{1-\alpha} \qquad \forall\; t,\; c \tag{4}$$

$$\frac{C^L_{t+1,c}-C^L_{t,c}}{1-\alpha}=\sum_{g=1}^{G}\left[\frac{1}{h}\sum_{t=\rho}^{t-1}P^I_{\bar{t},g}x_{\bar{t},g,c}-P^I_{t,g}x_{t,g,c}\right] \qquad \forall\; t,\; c \tag{5}$$

$$Y_{t,g}=\sum_{c=1}^{C}\sum_{\bar{t}=1}^{t}x_{\bar{t},g,c}-\sum_{\bar{t}}^{t}D_{\bar{t},g} \qquad \forall\; t,\; g \tag{6}$$

$$\sum_{g=1}^{G}Y_{t,g}\leqq\;M \qquad \forall\; t \tag{7}$$

$$Y_{t,g}\;\geq\;0 \qquad \forall\; t,g \tag{8}$$

$$Y_{T-h,g}\;=\;0 \qquad \forall\; g \tag{9}$$

식 (4)의 좌변은 t 시점에서 차량 g 의 구입과 관련된 모든 비용의 합에서 대출금으로 나가는 비용을 뺀 값이다. 이 값은 현재 캐피탈 회사의 c 에 대한 대출 한도인 $C^{L_{t,c}}$ 보다는 작아야 한다. 또 캐피탈에서 대출 받은 금액은 전체 금액의 (1-α) 이기 때문에, (1-α) 가 곱해진다. 캐피탈에서 대출받은 금액을 매 월 균등하게 상환한다(원금균등상환)고 가정할 때, 매달 캐피탈에 상환하는 금액은 잔존 금액인 (1-α) 을 대출 기간인 h 로 나눈 (1-α)/h 이다. 제약식 (5)는 다음 시점인 t+1 의 대출 한도를 계산하는데, 이는 현재 t 시점의 대출금 변경에 따른 대출금 한도에 의해 조정된다. 제약식 (6)은 차량 g 에 대한 t 시점의 재고비용을 계산한다. 차량의 재고 수량은 구입한 차량 대수와 수요량의 차이다. 제약식 (7)은 차량의 재고 수량이 회사의 보관 공간에 댈 수 있는 차량의 수 보다 작음을 의미한다. 제약식 (8)은 재고 차량 대수 $Y_{t,g}$ 에 대한 비부수 조건이며, 제약식 (9)는 마지막 시점인 T-h 에 재고가 0이 됨을 의미하고 있다.

2. Model simplification and constraint satisfaction

앞 장에서 설명한 문제는 장기 자동차렌탈 기업의 실제 의사결정 문제에서 얻어진 모형으로 실제 요구되는 모든 데이터 요소들이 모형의 모수로서 설정되어 있다. 이를 데이터보다는 보다 모형의 구조적인 측면에서 다음을 고려하여 요약할 수 있다. 원 문제에서 T-h 로 정의 되었던 시간대는 T 를 재정의 함으로써, T로 변경 되었다. 계약 시점과 차량의 인도시점의 차이인 τ 는 식의 단순화를 위

해 0으로 설정되었다. 기존의 목적함수에서 τ 는 렌탈 회사가 캐피탈 금융회사에 상환하는 금액과 렌탈 수수료를 받는 시점에 영향을 미치기 때문에 문제에 반영되어야 하지만, 목적식에서 전체 시간대에 대한 이윤을 계산하고, 발생한 모든 사건이 계산에서 고려된다면, 시점을 어떻게 구분하는지는 총 이윤 값의 계산에 큰 영향을 미치지 않는다. 또 원 문제의 세금 부분은 단순화된 모형의 목적식 (10)에서 $B_{t,g,c}$ 로 단순화될 수 있는데, 이는 세금은 세금율, 추정 차량 잔존가치, 차량 구매 비용과 같이 상수 값에 의해서 계산되며, 나중에 더해질 수 있기 때문이다. 이와 같이 목적식의 구조에는 영향을 미치지 않는 원 모수들을 모아서 하나의 모수로 단순화할 수 있다. 원 문제에서는 렌탈 회사가 재고 차량을 가지고 있을 수 있었지만, 단순화된 모형에서 렌탈 회사는 고객과의 계약으로 인해 필요한 차량 만을 주문함으로써 재고를 가지지 않는다. 이와 같은 방식으로 다음의 식 (10)-(13)을 얻는다.

$$\max \quad \sum_{t=1}^{T}\sum_{g=1}^{G}\sum_{c=1}^{C} B_{t,g,c}x_{t,g,c} \tag{10}$$

$$\text{s.t} \quad \sum_{g=1}^{G}[P^{I_{t,g}}x_{t,g,c} - \frac{1}{h}\sum_{\bar{t}=\rho}^{t-1} P^{I_{\bar{t},g}}x_{\bar{t},g,c}] \leqq \frac{C^{L_{t,c}}}{1-\alpha} \qquad \forall \ \ t,\ c \tag{11}$$

$$\frac{C^{L}_{t+1,c} - C^{L}_{t,c}}{1-\alpha} = \sum_{g=1}^{G}[\frac{1}{h}\sum_{\bar{t}=\rho}^{t-1} P^{I}_{\bar{t},g}\, x_{\bar{t},g,c} - P^{I}_{t,g}\, x_{t,g,c}] \qquad \forall \ \ t,\ c \tag{12}$$

$$\sum_{c=1}^{C} x_{t,g,c} = D_{t,g} \qquad \forall \ \ t,\ g \tag{13}$$

여기서 $B_{t,g,c}$는 다음과 같이 정의된다.

$$B_{t,g,c} = hR_{t,g} - (1-\alpha)P^{I_{t,g}}C^{R_{t,g,c}} - (1-\alpha b)P^{I_{t,g}} + P^{R_{t,g}} \tag{14}$$

이러한 새로운 모형에서 식 (14)는 세금을 제외한 원 문제의 목적식에 있었던 모든 요소들을 고려한다. 또 다른 차이는 식 (13)에서 렌탈 회사가 구입하는 차량의 수가 고객과의 계약을 이행하기 위해 필요한 차량 수와 일치한다는 것이다.

앞서 언급하였듯이, 차량 렌탈 문제는 제약식이 많은 수리모형으로, 동적계획법을 활용하여 문제를 해결하기 위해서는 제약식의 상태를 포함하는 상태 공간에 대한 복잡한 모형의 설정이 요구된다. 이 논문의 목적은 위 모형을 푸는데 있는 것이 아니라 제기되는 동적계획법과 제약식프로그래밍의 혼합 방식의 설명을 위해 위 모형을 사용하는 것이므로 설명을 위하여 제약식의 수를 줄여 문제를 단순화시킨 모형에 방법을 적용한다. 이는 뒤에서 설명되겠지만 각각의 제약식을 전통적인 방식의 동적계획법 적용할 때 각각의 데이터 구조체를 설계함이 필요한데 각각의 제약식에 따라 어떻게 데이터 구조체를 만들어 상태 공간에 포함시키는지는 연구의 목적과 부합하지 않기 때문에 본 연구에서 제안하는 방법과의 비교를 위해 목적에 부합하도록 모형을 구조적으로 단순화하여 사용한다. 이를 위해 원본 문제에서 수요에 대한 제약식을 제외한 다른 모든 제약식을 제거한 더 단순화된 모형을 생각해 보자. 다음 모형은 비록 몇몇 제약식이 제거되었지만 여전히 일반적인 제약식을 가진 구조적으로 유효한 문제이고 이를 어떻게 처리하는지를 비교 설명함으로써 제안하는 방법의 방식과 장점을 볼 수 있다.

$$\max \quad Z = \sum_{t=1}^{T}\sum_{g=1}^{G}\sum_{c=1}^{C} B_{t,g,c}x_{t,g,c} \tag{15}$$

$$\text{s.t} \quad \sum_{c=1}^{C} x_{t,g,c} = D_{t,g} \qquad \forall \ \ t,\ g \tag{16}$$

이 때,

$$B_{t,g,c} = hR_{t,g} - (1-\alpha)P^{I_{t,g}}C^{R_{t,g,c}} - (1-\alpha b)P^{I_{t,g}} + P^{R_{t,g}}$$

위의 문제는 단순 수리계획 문제로 볼 수 있지만 분리가능(separable)하기 때문에 동적계획법을 이용해 해를 찾을 수 있다. 이는 배낭채우기 문제를 동적계획법으로 푸는 것처럼 잘 알려진 접근 방법이다. 알고리즘에서, 최대화된 누적 합과 부분 목적 값은 변수값이 분기하는 과정마다 상태 값으로 저장된다. 여기서 연산의 핵심 아이디어는 알고리즘에서 상태를 어떻게 정의하는가 이다. 이 방법을 사용하면, 시간과 상태를 축으로 하는 값 (t, S_t)을 가지는 2차원의 상태 공간을 정의할 수 있다. 이 때 t 는 "시간" 혹은 "단계" 로 연산 단계를 나타낸다. S_t 는 "상

태"로 t시점의 변수 값을 대체하는 하는 값이다. 이러한 프레임 워크에서 계산하기 위해서는 의사결정 변수인 $x_{t,g,c}$가 선형적인 방법으로 배열되어야 한다. 동적 계획법에서 t는 의사결정 변수의 인덱스 트리플 (t,g,c)에 대응하고, 상태 S_t는 $x_{t,g,c}$의 값에 대응한다. 주의할 것은 표기 t가 원문제의 수리 모형과 동적 계획법 프레임워크 모두에서 나타나게 되는데, 맥락을 통해서 같은 심볼에 대해 의미를 구별할 수 있다. 원 문제에서 (t,g,c) 의 공간 $T\times G\times C$ 는 동적 계획법 모형에서 t 의 공간이 된다. S_t의 차원은 $\overline{D}=\max_{t,g}D_{t,g}$로 고정 된다. 따라서 DP 모형에서 상태 공간은 $(T\times G\times C)\times\overline{D}$가 된다. 우리의 목적은 각 (t,g,c)에 대해서 $x_{t,g,c}$를 결정하는 것이기 때문에, 동적 계획법은 상태 값 $V_t(S_t)$을 매 상태 S_t마다 원모형의 목적식을 부분적으로 합산함으로써 계산한다. 동적계획법 모형과 단순 모형의 차이를 요약하면 아래와 같다.

DP Model		Basic Model
t	$\leftrightarrow$	$(t,\ g,\ c)$
S_t	$\leftrightarrow$	$x_{t,g,c}$
$V_t(S_t)$	$\leftrightarrow$	$\sum B_{t,g,c}x_{t,g,c}$

동적 계획법 모형의 시간 t는 원래 문제의 인덱스인 $t=GC(t-1)+C(g-1)+c$에 의해 결정된다. 역으로, 인덱스는 차원을 반복적으로 나눔으로써 얻을 수 있다. 시간 t를 G'C 로 나눈 후, 이 몫에 1을 더한 값이 t 이고 나머지를 다시 C 로나눈 몫에 1을 더한 값이 g 이고 그 나머지가 c 가 된다. 이를 통해 인덱스 (t,g,c) 의 값을 얻는다. 상태 공간에서 벨만 방정식이 상태 정보 (t,S_t)와 이에 대응하는 상태 값 $V_t(S_t)$를 채우면서 작동한다. 보통은 벨만 방정식은 후진 방식의 방정식으로 표현되지만 여기서는 전진 형태의 벨만 방정식으로 표현한다. 이는 본 문제에서 후진이나 전진이나 대칭의 형태를 띄우기 때문이다. 그 식은 다음과 같다.

$$V_t(S_t)=\max_{a_t\in A_t}\{C_t(S_t,\ a_t)\ +\ V_{t-1}(S_{t-1}(S_t,\ a_t))\} \qquad (17)$$

식 (16)에서 S_t 가 가질 수 있는 최대값은 $D_{t,g}$ 로 고정되기 때문에, $S\in\{0,1,...,D_{t,g}\}$ 로 표현할 수 있다. 벨만 방정식은 상태 S_t 의 최적 가치는 $V_{t-1}(S_{t-1})$ 에 대해 현재 상태 S_t 에서 a_t 을 선택하여 C_t 를 t-1 상태 가치 V_{t-1}과의 합의 최대값으로 얻을 수 있다. 이는 가능한 A_t 의 원소 중 값을 최대화 시키는 a_t 을 선택함으로써 가능하다. 액션은 이전 상태인 S_{t-1} 에서 현재 상태 S_t 로 어떻게 도달하였는지에 대한 정보를 나타낸다. 우리의 동적 계획법 모형에서 액션은 현재 상태에 도달하는 마지막 단계의 상태로 계산된다. 즉 $a_t=S_{t-1}$ 로 정의함으로써 A_t 를 정의하는 조건을 만족하게 되며, 제약식 (16)이 A_t 에 대한 조건을 제공한다. 이윤 C_t 는 에 $x_{t,g,c}$ 의해서만 계산되기 때문에, 우리는 벨만 방정식 식 (16)을 다음과 같이 단순화 할 수 있다.

$$V_t(S_t)=\max_{S_{t-1}\in A_t}\{C_t(S_t)+V_{t-1}(S_{t-1}(S_t))\}$$

벨만 방정식의 계산은 문제의 최대 이윤을 보장한다. 문제를 풀기 위해 필요한 또 하나의 조건은 모형의 제약식 (16)을 어떻게 수행하느냐이다. 이 문제의 경우 단순한 최대화가 아닌, 제약식 (16)의 만족성을 탐색할 수 있는 최대화가 필요하기 때문에, 남은 과정은 제약식 (16)을 동적 계획법 프레임워크에서 사용될 수 있게 하는 것이다. 이에 대해 2가지 접근 방법을 시도해볼 수 있는데, 첫 번째는 동적 계획법을 푸는 과정에서 제약식 (16)을 무시하고 상태 값을 계산하는 단순한 개선 방법이다. 마지막 T 에서의 상태 S_T 에 대해 그 가치가 최대인 V_T 을 선택한다. 그리고 벨만 방정식에서 a_t 를 결정하는 단계에서 가능해를 유지하기 위해서, 선택된 a_t 가 가능해 조건을 위반하는지를 확인한다. 만약 가능해 조건을 위반하는 a_t 의 경우에는 이를 버리고, 다음으로 큰 V_T 로 이동한다. 이는 제약식을 만족하는 값을 찾을 때 까지 반복한다. 하지만 이러한 방법은 원하는 해를 구하는데 실패한다.

정리. 상기의 방법으로는 최적해를 보장할 수 없다.

증명. 우리의 상태 공간이 $N\times T$으로 정의된다고 하자. 이 경우에 T상태에 도달할 수 있는 경로의 N^{T-1}수는 개 이다. T시점에는 N개의 상태만을 가지기 때문에 제안된 방법은 N개의 가능한 결과를 가진다. T시점에서 종료되는 각 경로는 결정변수 값의 순서로써 각 단계에서 주어진 제약식을 무시하고 벨만 방정식을 계산함으로써 얻을 수 있다. 만약 벨만 방정식 과정에서 1개의 제약식을 위반하는 경우가 발생한다면, 해당 경로 전체는 불가능해가 된다. 만약 우리가 최대 N개의 경로가 모두 불가능해인

예시를 만들 수 있다면, 위의 방식이 불가능함을 보일 수 있다. 간단한 구성은 일부 t에 대해 제약 조건 $x_t = v$를 적용하고, v 이외의 값을 선택하는 경우에 대해서는 무한대의 이득을 그 외의 경우에는 0으로 놓는 것이다. 다른 모든 결정에 대해서는 이득이 0이라고 하자. 제약식을 만족하는 경로는 N^{T-2}개가 있기 때문에, 전체적으로 얻을 수 있는 이득은 0이 된다. 이 때, x_t에 대해서 v가 아닌 다른 값을 가지는 경로는 무한대의 이득을 얻게 된다. 그러므로, N개의 경로들은 모두 불가능해이다. ■

또 다른 합리적인 접근은 벨만 방정식을 계산할 때, $a_t = S_{t-1}$ 인 a_t 중 식 (16)을 만족하지 않는 a_t 을 고려대상에서 제외하는 것이다. 이는 벨만 방정식에서 후보 값들에 대한 추가적인 제약조건으로 이해될 수 있다. 이러한 제약 조건은 식 (16)의 좌변의 부분합에 대한 정보를 필요로 한다. 동적 계획법 모형에서 $x_{t,g,c}$ 를 S_t 로 인식하는 과정에서, $1 \le t_1 < \cdots < t_k = t < t_{k+1} < \cdots < t_n \le T$ 을 만족하는 경우에, 동적계획법 모형의 $t_1,\ldots,t_n$ 가 (t,g,c)와 대응한다고 하자. 이를 통해 벨만방정식의 단계 $k \in \{0,\ldots,t-1\}$ 에서 최대화 S_k^* 로 구성된 체인 $\left(S_1^*,\ S_2^*,\ldots,\ S_{t-1}^*,\ S_t\right)$ 을 정의할 수 있다. 이 제약식은 아래와 같다.

$$\sum_{i=1}^{t-1} S_i^* + S_t + \sum_{i=t+1}^{T} S_i = D_{t,g}$$

벨만방정식을 사용한 최대화 문제에서 우리는 최대값을 찾기 위해 $a_t \in A_t$를 만족하는 많은 대안들을 고려하게 되고, 앞에서 $a = S_{t-1}$이므로 이 과정은 최적 S_{t-1}를 찾는다. 이러한 과정에서 우리는 문제의 제약식을 위반하는 S_{t-1}들을 제거하고자 한다. 그리고 항상 현재 단계에 이르는 체인 상의 값을 계산할 수 있기 때문에 아래의 등식을 얻을 수 있다.

$$PS(S_{t-1}) = \sum_{i=1}^{t-1} S_i^* = (D_{t,g} - S_t) - \sum_{i=t+1}^{T} S_i$$

$PS(S_{t-1})$ 로 표현된 좌변의 부분합은 S_{t-1}에 의존하는데, 이는 최대화 체인이 어떠한 S_{t-1} 을 선택하는지에 따라 다른 값을 가지기 때문이다. 우변은 아직 알지 못하는 $(T-t+1)$ 개의 항들 즉 $S_t, S_{t+1},\ldots,S_T$ 가 있다. 개별 S_k 값은 상한과 하한을 가지기 때문에 우변의 값 역시 상한과 하한을 가지게 된다. 우리는 이 상한 $PS_{\max}$ 와 하한 $PS_{\min}$ 의 관계를 다음과 같이 표현할 수 있다.

$$PS_{\min} \le PS(S_{t-1}) \le PS_{\max}$$

이를 통해 벨만 방정식에서 불가능한 행동의 조건을 추론하는 식을 구할 수 있다. 즉 아래의 조건을 만족하는 A_t 에서의 S_{t-1}를 선택하고자 하는 행동들 중에 제거하여야 한다. 이를 한계일관성(Bound Consistency)라고 부르고 이렇게 얻어진 제약식은 No-good의 일종이 된다.

$$(PS(S_{t-1}) < PS_{\min}) \vee (PS(S_{t-1}) < PS_{\max})$$

다음으로 위 2개의 상한과 하한을 계산은 $0 \le x_{t,g,c} \le D_{t,g}$에서 다음을 얻는다.

$$PS_{\min} = -(n-k-1)D_{t,g} - S_t$$

$PS_{\max} = D_{t,g} - S_t$하지만 $PS(S_{t-1}) \ge 0$ 은 항상 성립하기 때문에 $PS(S_{t-1}) < PS_{\min}$은 무의미한 식이고 따라서 남은 식은 $PS(S_{t-1}) > PS_{\max}$로, 이 부등식을 만족하는 행동을 선택 대안에서 제외할 수 있다.

III. Computational frameworks

3. Methods in the DP framework

실현가능 조건을 보장할 수 있는 유일한 방법은 벨만 방정식을 푸는 모든 과정에서 가능해를 확인하고, 이를 만족하지 못하는 선택을 제외하는 것이다. 이를 위해서는 이전 상태에서 선택되었던 의사결정 변수들을 계속 추적하는 기능이 필요하다. 제약 조건의 위반을 확인하기 위해서 알고리즘은 현재 상태에서 선택된 체인 상의 모든 변수들 확인하고, 이 변수들이 제약식들을 위반하는지 여부를 확인하게 된다.

제약식을 처리하기 위해 전통적인 동적계획법 틀에서는 두 가지 방법을 생각해 볼 수 있다. 첫 번째 방법은 제약 조건을 평가하는데 필요한 모든 정보를 포함하는 상태 값에 대한 정보를 저장하는 것이다. 우리의 문제에서 상태 $V = (V_1, V_2)$로 표현된다고 하자. 이 때 V_1는 벨만 방정식에 의해 계산된 목적함수의 값이고, V_2는 식 (16)에서 이미 이전 단계의 벨만 방정식을 통해서 결정된 변수들의 값을 이용한 제약식 좌변의 부분합의 값이다. 동적 계획법 모형의 단계 t 에서 V_2는 각각의 (t,g) 인덱스에 대해, $GC(t-1) + C(g-1) + \tilde{c} \le t$ 인 경우 아래와 같이 계산된다.

$$\sum_{c=1}^{\tilde{c}} x_{t,g,c}$$

이는 알고리즘이 각각의 (t,g) 에 대해 $V_t(t,S_t)$ 값을 저장하므로 $T \times G$ 크기의 행렬이 동적 계획법의 단계

(t,S_t) 마다 계속 저장해야 함을 의미한다. 하지만 이러한 접근은 저장되는 행렬의 크기가 커지는 경우에 문제가 될 수 있다. 문제에서 값의 저장을 위한 행렬은 동적 계획법의 각 단계 (t,S_t) 에 대해 $T\times G$ 의 크기이고, 이때 $D_{\max}$ 가 $D_{t,g}$ 의 최대값일 때는 $(T\times G\times C)\times D_{\max}$ 번의 계산이 이루어지기 때문에 전체 행렬의 크기는 $(T\times G\times C)\times D_{\max}\times(T\times G)$ 로 대단히 커질 수 있다. 다행히 이런 행렬의 크기를 줄이는 것이 가능한데, 이는 서로 다른 (t,g) 에 대해서는 값을 계산할 필요가 없고, 주어진 t 와 g 의 루프 내에서의 c 가 변화됨에 따른 부분합에 대해서만 계산이 수행되기 때문이다. 이를 통해 다른 (t,g) 로의 변경 이후 $c=1$ 이 되는 경우에, 이 $T\times G$ 크기의 행렬은 0으로 다시 초기화되어 재 사용될 수 있다. 이는 연산과정에서 크기가 $T\times G\times C\times D_{\max}$ 인 행렬 1개만을 필요로 함을 의미한다. 매 연산과정에서 벨만 방정식의 평가가 이루어지기 전에, 알고리즘은 가능한 S_{t-1} 에 대해 행렬의 부분 합을 계산하고, 이 값이 상한인 $D_{t,g}-S_t$ 보다 큰 경우에는 해당 S_{t-1} 값을 A_t 에서 제외시킨다. 벨만 방정식의 평가 후에 행렬은 최대화에 의해 선택된 부분합과 S_t 에 의해 현재 과정 t 에 대한 새로운 부분합으로 갱신된다. 다음 알고리즘은 일반적인 동적 계획법의 주 루틴을 보여준다. 가치를 저장하는 $V_t(S_t)$ 는 벡터로 첫 항은 일반적인 목적함수 식의 누적값을 보관하고 둘째 항은 제약식의 부분합을 보관하는 두 개의 항을 갖는다.

```
Algorithm: ByState
for t∈ {1,2,...,T} do
    for s∈{0,1,...,D_max} do
        v, p ← Bellman(t, s)
        V_t(s)← v
        pred(t,s)← p
```

두 번째 방법은 부분합을 계속 기록하는 대신에 벨만 방정식의 매 단계에서 얻어진 모든 최대화 해들을 기록하고 이 기록을 통해 앞 단계에서의 의사결정 결과를 이용하여 현 단계의 벨만 방정식의 후보 선택들을 얻기 위한 부분합의 계산에 사용하게 된다. 보통 동적계획법 알고리즘은 연산 후에 해를 찾기 위해 벨만 방정식의 최대값을 만드는 행동들, 즉 여기서 다루는 문제에서는 변수들의 값을 저장하기 때문에, 이 변수값들을 이용하여 원하는 수식을 계산하는 것은 어렵지 않다. 때문에 기존의동적 계획법 알고리즘과 본 연구에서 제안하는 방법의 유일한 차이점은 제약식을 적용하기 위하여 이와같은 변수값 정보를 사용하고 부분 합을 구성하는 것이다. 모든 S_k 가 $x_{t,g,c}$ 에 대응하기 때문에 관련된 값을 선택하고, 부분합을 계산하는 것은 가능하다. 제약식 (16)에 대한 부분합을 구하기 위해서는 t 로부터 (t,g,c) 인덱스를 계산해야 한다. 예를 들어 $\bar{t}$가 $(t,g,1)$에 해당하는 상태라고 하면, $\sum_{k=\bar{t}}^{t-1} S_k$ 은 제약식 (16)의 부분합이다. 이러한 방법으로 알고리즘은 벨만 방정식의 A_t 를 계산함으로써 제약식을 만족하지 않는 S_{t-1} 를 선별하고 제외시킨다. 즉 첫째 방법과의 차이는 부분합의 값을 상태로서 기록을 하느냐 아니면 필요에 따라 부분합을 다른 정보를 이용하여 계산하여 제약식을 구현하느냐의 문제로 일반적인 공간-저장의 트레이드오프 관계로 볼 수 있다. 다음의 알고리즘은 이 두번째 방법에서 접근 가능한 행동의 집합 A_t를 결정하는 과정을 보여준다. 함수 traceback에 의해 과거의 벨만 방정식의 풀이에서 얻은 최대해를 추출해내고 함수 violate_constraints에 의해 주어진 제약식을 위배하면 해당 행동을 A_t 집합에서 배제하고 있다.

```
Algorithm: Calculate A_t
A = []
(t_1, g_1, c_1)← t
for a∈{0,1,...,D_{t_1,g_1}}
    pth← traceback(t−1, a)
    if violate_constraints(t, s, pth) then
        continue
    A ← {a}∪ A
return A
```

4. A method using CP

앞에서 제약식을 다루기 위해 전통적인 동적 계획법 프레임워크를 사용하는 방법을 소개하였다. 제약식을 만족하지 못하는 벨만 방정식의 연속적인 최대값 체인과 같은 부분해를 필터링하는 방법을 사용하였다. 하지만 이 방법은 문제가 다수의 복잡한 제약식을 포함하는 경우에, 모형 관리자에게 이를 필터링할 수 있는 알고리즘을 구축해야 하는 어려운 과제를 남기게 된다. 또 모형 관리자는 데이터 구조 및 솔루션 메커니즘을 설계하기위한 알고리즘을 구성해야 한다. 이러한 알고리즘의 복잡성은 연구단계에서는 받아들여질 수 있지만, 실제에서 복잡한 문제를 해결하기에 적합한 방법은 아니다. 대신에 어떤 형태의

제약식들이 모형에 포함되던 이를 특별한 알고리즘을 설계하는 단계없이 일반적으로 작동하는 방식이 더 권장되고 따라서 이를 위한 일반적인 솔버가 필요하다. 그 한 가지 접근은 제약식프로그래밍(CP)의 해결틀을 사용하여 제약식을 만족시키는 행동을 추출하는 것이다.

벨만 방정식에서 $V_t(S_t)$ 을 계산하기 위해서는 최대화 문제를 풀어야 한다. 이 때 최대화를 위한 행동은 실현가능 집합인 A_t에서 선택된다. 이 실현가능 집합은 제약식(16)을 만족하는 부분해들을 포함한다. 이전 장에서 제안되었던 과정에서, 실현가능한 행동 집합은 다음과 같이 표현될 수 있다.

$$A_t = \left\{ S_{t-1} | \sum_{c=1}^{C} x_{(t,g,c)} = D_{(t,g)},\ \forall\ t,g \right\}$$

앞 장에서 설명된 바와 같이, t 단계의 동적 계획법 계산에서 S_{t-1} 의 각 값에 대해 $t-1$ 단계 이전에서 벨만 방정식을 풀어서 나온 최적 부분해의 체인 $\left(S_1^*, ..., S_{t-2}^*, S_{t-1}\right)$ 을 특정할 수 있다. 그리고 S_k^* 값들은 원 문제의 변수들 $x_{t,g,c}$ 의 값에 해당하여 위 체인은 원문제의 부분해를 구성하고 제약식의 부분합을 계산하는데 사용된다. 그리고 계산의 단계가 T 에 도달할 때까지 변수 $x_{t,g,c}$ 들의 완전한 값을 모르므로 제약식의 만족여부를 결정할 수 없다. 그러나 부분해는 제약식의 일탈에 대한 의미있는 정보를 포함하여 이를 이용하면 제약식을 일탈케 하는 행동을 찾아낼 수 있다. 예로써 다음의 식이 있다고 하자.

$$X_1 + X_2 = 2$$

여기서 $X_2 \in \{0,1,2\}$ 이고 $X_1 = 3$ 을 강제하는 행위가 적용되었다고 하자. 그러면 이 정보로부터 설명 X_2 의 값이 결정되지 않은 상태임에도 변수는 X_2 는 위 등식을 만족하는 해가 없다는 사실을 발견할 수 있다. 그러면 $X_1 = 3$ 은 주어진 식을 거짓으로 만드는 값이기 때문에 X_1 의 선택에서 제외되어야 한다. 이와 같은 확산법(Propagation)을 통해 주어진 제약식을 위배하는 변수값을 대안들을 제거하게 된다.

제약식프로그래밍의 사용은 다수의 제약식을 강하게 제약되는 상황에서 계산상의 더 큰 이점을 지닌다. 즉 많은 제약식에 대한 통합적 추론을 통해 각각의 제약식에서 얻게 되는 추론보다 더 많은 정보를 추론할 수 있다는 점이다. 예를 들어 $X_1 \in \{1\}$, $X_2 \in \{0,\ 1,\ 2\}$, $X_3 \in \{0,\ 1,\ 2\}$을 변수로 가지는 아래의 두 제약식을 가지고 있다고 하자.

$$X_1 + X_2 = 2, \quad X_2 + X_3 \leq 2$$

변수들의 값을 보면 X_1 의 값은 1로 고정되어 있지만, X_2 와 X_3 은 결정되지 않은 상태이다. 그러면 $X_1 \in \{1\}$ 이라는 사실을 이용해 첫 번째 제약식에 대입하면 $X_2 = 1$ 이 되어야 하고 이를 통해 $X_2 \in \{1\}$ 라는 사실을 얻는다.

추론을 통해 제약식들을 만족시키는지를 확인할 수 있고 만약 제약식을 만족시키지 못하면 그 행동은 A_t 에서 제거된다. 제약식프로그램의 솔버를 사용한다는 것은 앞 장의 일반적인 방법보다 훨씬 강력한 추론을 바탕으로 행동의 대안들을 훨씬 빠른 속도로 축소해 나갈 수 있다는 것이다. 본질적으로 앞 장에서 제시한 한계일관성 방법은 제약식프로그래밍에서 사용하는 확산법 중의 한 기법을 이용한 것이다.

알고리즘 1은 제약식프로그래밍을 이용한 본 연구의 주 루틴 부분을 보여준다. 알고리즘의 줄 1-8은 상태 공간을 구성하고, 함수 Bellman 은 벨만 방정식을 풀어서 나온 최적 상태 가치와 최적이 되게 만드는 행동, 여기서는 바로 이전의 상태 값 S_{t-1}을 되돌려주는데 이를 각각 $V(S_t)$ 와 $pred$ 리스트에 기록한다. 줄 10에서는 마지막 단계 T 에 저장되어 있는 모든 상태 값을 비교하여, 그 중 최대값을 찾는다. 이 값이 문제에서 계산되는 최대 이익이 된다. 줄 11-17은 최적해 $x_{t,g,c}$ 을 찾기 위해, 매 단계 t마다 $pred$ 리스트에 저장되어 있는 벨만 방정식에서 최대 값을 만드는 행동 $a_t = S_{t-1}$ 를 추적하여 결정한다. 알고리즘 2는 현재 단계 t 와 상태 s 를 입력으로 함수 Bellman의 계산을 보여준다. 이는 벨만 방정식의 최대화 문제를 푸는 것으로 $t = 0$ 일 때는 단순히 그 단계의 수익이 상태의 가치로 돌려준다. 단계 t 가 1 이상일 때는 벨만 방정식에서처럼 함수 $Action(s,t)$ 이 돌려주는 모든 후보 행동들의 리스트에 있는 각 행동에 대해 함수 $satisfiable(t,s,pth)$ 로 제약식을 위반하는지 여부를 판단한다. 즉 이 함수가 제약식들의 변수가 택할 수 있는 값들의 집합을 내보내는데 이 중 그 집합이 공집합이 되는 변수가 있다면 제약식을 만족하지 못하는 것이 된다. 이를 통해 대응하는 모든 제약식을 만족하는 행동들의 집합이 접근가능한 행동들(admissible actions)이 된다.

주 루틴이 실행되기 전에, 제약식 (16)이 제약식프로그래밍에서 요구하는 문법에 따라 제약식프로그래밍 솔버에 추가되고, 모든 (t,g,c) 에 대해 변수 $x_{t,g,c}$ 들이 제약식프로그래밍에서 정의한 도메인 제약식으로 추가된다. 초기 상태에서는 모든 제약식 (16)이 만족되는 것으로 나타나는데 이는 어느 제약식도 위배 상태가 나타나지 않기

때문이다. 즉 제약식을 만족하는 실현가능해를 이루는 변수들 $x_{t,g,c}$ 의 값이 적어도 1개는 존재하기 때문이다.

Algorithm 1: $main$

```
for t∈{1,2,...,T} do
    (t̂, ĝ, ĉ) ← convert t
    for s∈{0,1,...,D_{t̂,ĝ}} do
        v,p← Bellman(t,s);
        V(S_t)← v;
        pred(S_t)← p;
    end
end
s*← arg max_{S_T}(S_T);
v*← V(s*);
x*_{T,G,C} ← s*;
p̄← pred(S_T);
for t∈{T−1,...,0} do
    (t̂, ĝ, ĉ) ← convert t;
    x_{t̂,ĝ,ĉ} ← p̄;
    p̄← pred(p̄) ;
end
return v*,x*;
```

Algorithm 2: $Bellman(t,s)$

```
if t = 0 then
    pred(s)← −1;
    return C(s,−1),−1;
end
A← Action(t,s)
if A = ∅ then
    return −∞ ,−1;
end
a*← argmax_{a∈A}{C_t(s,a)+ V_{t−1}(s,a)}
v*← C_t(s,a)+ V_{t−1}(s,a)
return v*,a*;
```

Algorithm 3: $Action(t,\ s)$

```
A ← ∅ ;
for a∈{0,...,D̄} do
    pth ← tracing back from S_{t−1} = a;
    If Satisfiable(t,s,pth) then
        A ← A ∪ {a}
    end
end
return A;
```

Algorithm 4: $Satisfiable(t,s,pth)$

```
add value-fixing constraints to the CP solver using pth;
call the CP solver and propagate;
return no empty sets in domains;
```

주 루틴이 실행되는 과정에서, satisfiable은 주어진 상태 경로로 부터 주변 제약식 변수를 식별하고, 변수에 대한 제약식을 고정시킬 수 있는 값을 생성한다. 예를 들어 동적계획법 모형에서 $T=3,\ G=2,\ C=4,\ \overline{D}=2$ 이고, 현재 시점이 $t=7$ 이라고 하자. 그럼 $(t,g,c)=(1,2,3)$ 가 $t=7$ 에 해당한다. $S_7=1$인 경우 $V_7(1)$를 계산하기 위해 행동 $a_6=S_6=2$ 를 고려하고, 상태 $S_6=2$ 이 되는 과정까지 벨만 방정식에서 체인 $S_1=S_2=S_3=1,\ S_4=2,\ S_5=1$ 로 결정된 경우를 가정해 보자. 이 경우에 고정되는 값은 S_5 와 S_6 에 해당하는 변수에 대해 $x_{1,2,1}=1$ 과 $x_{1,2,2}=2$ 이고, 이 값이 제약식프로그래밍 솔버에 추가된다. 그러면 솔버는 그 제약식들을 추가하게 되어 솔버가 갖는 제약식은 다음과 같이 된다.

$$x_{1,2,1}\in\{0,1,2\},\ x_{1,2,2}\in\{0,1,2\},\ x_{1,2,3}\in\{0,1,2\},\ x_{1,2,4}\in\{0,1,2\}$$

$$x_{1,2,1}+x_{1,2,2}+x_{1,2,3}+x_{1,2,4}=2$$

$$x_{1,2,1}=1,\ x_{1,2,2}=2$$

제약식프로그래밍 솔버는 추론을 통해 각 변수들의 도메인을 줄여간다. 그리고 결국은 제약식 확산법에 의해 도메인을 $dom(x_{1,2,3})=\varnothing$ 과 $dom(x_{1,2,4})=\varnothing$ 로 축소한다. 왜냐하면 먼저 마지막 추가된 제약식 $x_{1,2,1}=1$과 $x_{1,2,2}=2$에 의해 두 변수는 $x_{1,2,3}\in\{1\}$과 $x_{1,2,2}\in\{2\}$로 축소되고 다음 제약식 $1+2+x_{1,2,3}+x_{1,2,4}=2$에서 변수 $x_{1,2,3}$과 $x_{1,2,4}$의 도메인에 있는 어떤 값도 제약식을 만족시키지 못하여 도메인에서 제거되어 두 집합이 공집합이 되어버린다. 이를 통해 $a_6=2$이 적절한 행동이 아님을 알 수 있고, 나아가 벨만 방정식 계산시 A_7의 후보 행동에서 빠지게 된다. Fig. 1은 제약식프로그래밍에서 정의된 제약식들을 보여준다. 제약식프로그래밍 솔버로 Prolog로 구현된 경우로 문법은 Prolog의 문법을 따르고 있다. 보이는바와 같이 제약식프로그래밍을 사용하는 경우 전통적인 방법과는 달리 제약식의 상태를 표시하기 위한 데이터 구조체나 제약식의 부분해를 계산하기 위한 코드 등이 필요하지 않고 원래 문제가 정의된 형태와 유사한 방식으로 정의되고 있는 것을 볼 수 있다. 이는 앞의 방법과는 다르게 모형의 해를 구하기 위한 구현의 단계에서 용이성과 에러 회피성에서 중요한 장점을 지닌다.

```
:- use_module(library(clpfd)).
constraint(0, 0, [X0, X1, X2, X3, X4, X5, X6, X7, X8, X9]) :-
    [X0, X1, X2, X3, X4, X5, X6, X7, X8, X9] ins 0..13,
    sum([X0, X1, X2, X3, X4, X5, X6, X7, X8, X9], #=, 13),
    label([X0, X1, X2, X3, X4, X5, X6, X7, X8, X9]).
constraint(0, 1, [X10, X11, X12, X13, X14, X15, X16, X17, X18, X19]) :-
    [X10, X11, X12, X13, X14, X15, X16, X17, X18, X19] ins 0..2,
    sum([X10, X11, X12, X13, X14, X15, X16, X17, X18, X19], #=, 2),
    label([X10, X11, X12, X13, X14, X15, X16, X17, X18, X19]).
constraint(0, 2, [X20, X21, X22, X23, X24, X25, X26, X27, X28, X29]) :-
    [X20, X21, X22, X23, X24, X25, X26, X27, X28, X29] ins 0..2,
    sum([X20, X21, X22, X23, X24, X25, X26, X27, X28, X29], #=, 2),
    label([X20, X21, X22, X23, X24, X25, X26, X27, X28, X29]).
constraint(0, 3, [X30, X31, X32, X33, X34, X35, X36, X37, X38, X39]) :-
    [X30, X31, X32, X33, X34, X35, X36, X37, X38, X39] ins 0..6,
    sum([X30, X31, X32, X33, X34, X35, X36, X37, X38, X39], #=, 6),
    label([X30, X31, X32, X33, X34, X35, X36, X37, X38, X39]).
.....
```

Fig. 1. An example of constraint definition for the constraint programming

이렇게 정의된 제약식은 앞에서 보여준 동적계획법을 구현한 부분의 Satifiable 함수에 의해 호출되고 이는 제약식 프로그래밍 솔버에서 해풀이 과정을 거쳐 그 결과가 동적계획법 부분으로 넘어옴으로써 계산이 진행된다.

5. Computation and discussion

제안된 방법론의 성과를 비교하기 위해서, 실제 데이터를 기반으로 무작위의 데이터가 생성되었다. 실제 데이터에서 차량의 최대 수요량은 17대이기 때문에 데이터의 차량 대수는 0과 17사이의 값을 가지는 균등분포를 통해 생성되었다. 자동차의 구입 가격은 천2백만원에서 1억4천만원 사이에서 발생되었다. 렌탈 비용을 계산하기 위해, 이익율은 5~20%사이에서 무작위로 생성한다. 렌탈 비용은 전체 계약 금액을 계약기간으로 나눈 값이다.

위와같이 무작위로 만들어진 인스턴스에 대해 앞 장에서 제시한 세가지 알고리즘에 대한 코드를 구현하였다. 먼저 ByState는 동적 계획법의 상태 공간을 확장하여 제약식의 중간 상태를 저장함으로써 단계별로 저장된 값들을 이용해 No-good을 생성함으로써 제약식을 강제하는 코드이고, ByPath는 ByState와 마찬가지 No-good을 생성하나 제약식의 중간 상태를 저장함 없이 이전 단계에 계산된 변수들의 값을 이용해 제약식에 있는 표현을 계산하고 이를 통해 얻어진 값을 이용하여 제약식을 강제하는 코드이다. 사실상 No-good은 제약식프로그래밍에서 온 개념으로 전통적인 동적계획법에서는 이에 대한 명시적인 개념이 없다. 단지 제약식프로그래밍의 가장 단순한 추론인 한계일관성(bound consistency)을 앞의 두가지 코드에 적용하였다.

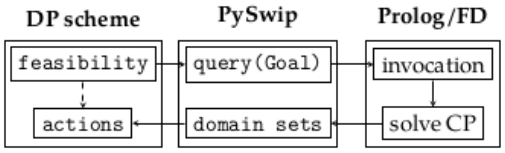

Fig. 2. Architecture of ByCP

마지막으로 ByCP는 앞의 코드와는 전혀 다르게 상태공간은 문제의 목적함수식 값에 해당하는 값만을 보관하고 제약식에 대한 정보는 제약식프로그래밍 솔버로 이전한다. 모든 제약식의 정보는 이 솔버에서 변환되고 제약식 확장법과 레이블링(labeling)에 의해 제약식이 강제된다. 확장법은 한계일관성에 의한 추론 외에도 다른 추론 방식들도 사용되고 레이블링은 탐색을 이용하여 불완전한 추론을 보완하여 완전추론을 만들게 된다. 이 코드들은 Python 3.6에 의해 구현되었고 특히 ByCP의 경우 제약식 프로그래밍 솔버로 SWI-Prolog가 포함하고 있는 FD 솔버를 이용하고 있다. ByCP의 구현을 위해 Python과 SWI-Prolog 간의 인터페이스가 필요하고 이를 위해 Python의 라이브러리 중의 하나인 PySwip을 이용하였다. 이 라이브러리를 통해 프롤로그의 개체를 설정하고 프롤로그 파트에서 각 t 에 대해 최대 $D_{\max}$만큼의 프롤

로그 엔진을 생성하고 각각에 대해 FD 솔버를 가동하여 각각의 행동에 대해 제약식의 위배성을 탐지한다. 이 모든 과정을 프롤로그 언어를 통해 구현하였다. Fig. 2는 코드의 구조를 보여주고 있다.

Table 2. Comparison of computation times

(T, G, C)	D_{max}	Size	ByState	ByPath	ByCP
20,5,5	10	5000	0.1361	7.2501	19.8921
			0.1355	7.3214	19.9183
			0.1359	7.3566	19.9411
			0.1358	7.3425	19.9175
			0.1455	7.4693	19.8976
30,5,5	10	7500	0.2155	18.8287	37.3533
			0.2216	18.8232	37.4764
			0.2207	19.0221	37.4142
			0.2203	19.1860	37.4815
			0.2139	19.1654	37.5396
40,5,5	10	10000	0.3047	36.9679	59.2711
			0.3052	37.4027	59.3312
			0.3125	37.5194	59.2787
			0.3048	37.5880	59.3237
			0.3059	37.4161	59.2680
50,5,5	10	12500	0.3716	59.8460	82.5413
			0.3744	60.3836	82.9630
			0.3748	61.3037	83.2753
			0.3738	61.8591	82.6510
			0.3754	60.6192	82.4793

Table 2는 3가지 방법의 계산시간을 보여준다. 첫 열은 인스턴스 사이즈이고, 두 번째 열은 해당 인스턴스의 최대 수요를 나타낸다. 표의 결과에서 보듯이 ByState의 연산이 가장 빠르다. 이는 이 방법이 해를 찾는데 가장 효과인 개선방법이기 때문이다. 상태와 시간 간의 상충관계를 고려하였을 때, 이 ByState는 실현가능 해 조건을 검사하는 과정의 모든 중간 값을 저장한다. 계산은 저장된 값들의 비교에 불과하니 계산시간을 절약할 수 있다. 이와는 대조적으로 ByPath와 ByCP는 매 단계 벨만 방정식의 계산에서 필요로 하는 값을 중간 계산을 통해서 도출하게 된다. 또 이 과정에서 같은 계산을 중복해서 계산하는 부분이 발생하고, 이는 전체적으로 계산 시간을 증가시킨다.

ByPath와 ByCP을 비교하면, ByCP는 ByPath에 비해 2가지 추가적인 연산을 필요로 한다. 두 방법 모두 이전 상태의 값을 검색하는 과정은 동일하지만, 실현가능해 조건 확인의 의존성에서 차이가 있다. ByPath는 제약식의 좌변에 해당하는 표현을 계산하고, 이를 우변의 표현식과 비교한다. ByCP는 이 비교 과정에 제약식프로그래밍 솔버를 이용한다. 먼저 계산된 중간 값일수록 더 자주 평가되기 때문에, 인스턴스의 크기가 커짐에 따라서 계산시간은 비선형적으로 증가할 수 있다. 하지만 ByPath 와 ByCP는 향후의 개선 방향에 차이를 가진다. ByPath 는 ByState 의 서로 반대쪽 극단에 있는 방법이다. 즉 ByState 중간 계산을 줄이기 위해 모든 계산의 결과를 메모리에 저장하는 방법이고 ByPath 는 저장되는 양을 줄이기 위해 모든 중간과정을 다시 계산하는 것이다. 서로 극단에 있기에 추가적인 개선이 어렵다. 메모리 자원을 절약하느냐 계산 자원을 절약하느냐의 문제는 어느 하나가 좋다기 보다는 알고리즘을 사용하는 맥락에 따라 장단점이 달라지지만 일반적으로 동적계획법을 사용하는 문제의 경우 대규모의 문제인 경우가 많기 때문에 메모리 자원을 사용하여 계산 자원을 절약하는 방식을 취한다. 그런 측면에서 ByState가 일반적인 경우라 볼 수 있지만 메모리에 필요한 데이터의 저장을 위한 데이터 구조의 설계를 수반한다. ByCP는 제약식프로그래밍을 사용하기 때문에 제약식의 상태에 대한 정보가 FD 솔버에 저장되는 것으로 볼 수 있다. 현재로서는 ByCP 코드가 완전하고 효율적인 구현을 하지 못한 측면이 있고 여전히 해결해야 하는 많은 이슈들이 존재하고 있어 더 많은 연구와 알고리즘에 대한 개선이 필요한 상황이다.

Fig. 3은 이 세가지 알고리즘들의 계산시간을 보여주고 있다. Table 2 에서 보여준 통계량을 평균하여 문제의 크기에 따른 평균적인 계산시간이 어떻게 변화하는지를 보여준다. 문제의 크기는 변수 수의 증가와 연결되지만 계산시간과 밀접한 관련을 갖는 것은 제약식 수의 증가이다. 그림에 따르면 ByState는 문제의 크기가 증가해도 계산시간의 변화가 없다. 당연히 상태공간의 확장에 따른 계산량의 증가는 거의 미미하기 때문이다. 반면 ByPath와 ByCP는 제약식의 증가에 따라 계산 시간이 증가한다. 당연히 제약식의 표현에서 더 많은 변수 더 많은 제약식 표현을 계산하기 위해 이전 단계의 값들을 더 많이 참조하고 이를 이용해 더 많이 수적 계산을 하기 때문이다. ByPath 와 ByCP 를 비교하면 계산량의 증가 속도는 거의 같다는 것을 알 수 있다. 단지 제약식프로그래밍 솔버는 더 강한 일관성을 얻기 위해 더 많은 계산을 필요로 하기때문에 ByCP의 계산 시간이 더 많다는 것을 알수 있다. 흥미로운 것은 그 계산 시간의 차이가 문제의 크기가 증가함에 따라 증가하지 않고 거의 상수에 해당할 정도로 변화가 없다는 것이다. 이는 더 강한 일관성을 단지 고정된 시간만큼의 계산 추가로 얻을 수 있음을 의미한다고 해석할 수 있다.

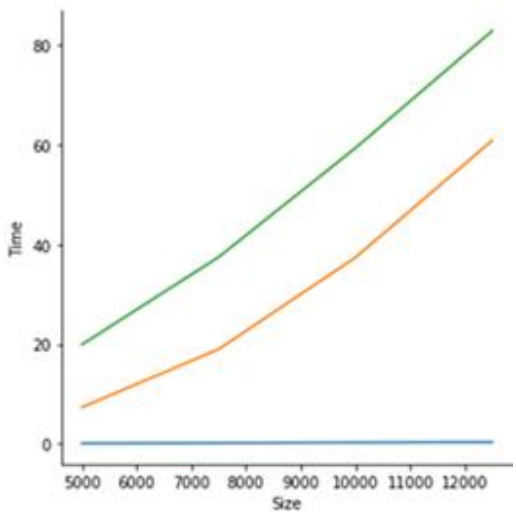

Fig. 3. The relationship of size and computational time (ByCP, ByPath and ByState from top to bottom)

ByCP의 장점은 사용성이다. 결과에서 보듯이 ByState의 계산 속도가 가장 빠르다. 이는 그 구현 방식이 가장 머쉰코드의 양식에 가까워 머쉰의 입장에 가장 불필요한 계산을 절약한 방식이기 때문이다. 하지만 일반적인 머쉰코드들이 지닌 약점처럼 시스템적인 설계 방법이 없기 때문에 주어진 제약들에 대한 상태를의 정의가 대규모로 이루어지면 이를 구현하는 것이 거의 불가능하다. 하지만 ByCP에서는 CP 솔버를 만족하는지에 대한 자세한 계산을 통해, 원래 문제의 제약식들을 변형하지 않고 반영할 수 있으므로 인간의 표현 양식에 더 가까우므로 모형을 구현하는데 훨씬 큰 장점을 지닌다.

또 한가지 장점은 완전 일관성(full consistency)를 보장한다는 것이다. 일관성은 제약식프로그래밍에서 중요시하는 개념으로 주어진 제약식을 위배하는 모든 변수의 값들을 제거하였을때 완전 일관성이 이루어진다. ByState 같은 경우 알고리즘의 설계에 한계 일관성 같은 간단한 추론 정도 포함할 수 있을뿐 여러 제약식들이 구조적인 특성에서 나올 수 있는 여러 추론들을 포함하는 것이 불가능하다. 예를들어 레이블링 같은 추론을 포함할 수 없다. 일반적으로 제약식프로그래밍은 각 변수들을 정의하고 이 변수들이 가질수 있는 도메인을 주고 이 위에 제약식을 표현하여 모형을 구성한다. 그리고 제약식 확산법을 통해 각 도메인에서 제약식을 위배할 수 있는 값들을 제거하는 과정을 거치게 되어 최종적으로 모든 도메인의 값들을 선택하면 모든 제약식이 성립하게 되도록 변수의 도메인을 축소하게 된다. 결국 더 이상 제약식을 위배할 수 있는 값이 어느 변수의 도메인에 포함되어 있지 않은 상태를 완전 일관성이 보장되는 상태이다. 이 상태에 도달하면 각 변수에 대해 그 도메인의 어떤 값을 선택하더라도 제약식이 성립하는 실현가능해를 조합할 수 있다. 즉 ByCP 방식은 제약식프로그래밍 이론에 근거한 많은 추론 방법의 도움을 받을 수 있다. ByCP에서 사용하는 FD 솔버에 입력되는 제약식은 세가지 형태로 얘기할 수 있다. 먼저 두가지는 앞에서 설명한 변수의 도메인과 문제에서의 제약식에 해당하는 식이고 마지막으로 레이블링(labeling)식이다. 이 마지막 식이 솔버에서 완전 일관성을 보장해주는 역할을 하고 있다. 제약식프로그래밍의 관점에서 ByState와 ByPath 두가지 알고리즘을 보면 이는 단지 여러 일관성 중에 한계 일관성(bound consistency)만을 이용하여 No-good을 생성한 알고리즘이고 따라서 일관성을 아주 협소하게 사용하고 있다. 이 방법은 완전 일관성을 보장해주지 못하기 때문에 해 공간에 여전히 제약식을 위배할 수 있는 해가 존재할 가능성을 배제할 수 없다. 반면 ByCP에서는 제약식 확산법과 함께 레이블링을 사용하므로 완전 일관성을 보장한다. 이같은 완전 일관성의 보장을 위해 솔버는 더 많은 계산을 하게되고 그 결과로 더 오랜 계산시간을 요구하고 있다.

IV. Conclusions

이 연구는 제약식이 있는 동적계획법 모형을 풀기위한 방법으로 전통적인 동적계획법의 틀과 제약식프로그래밍의 혼합한 방식을 제안하고 장기 차량 렌터카의 운영을 위한 금융 의사결정 문제의 풀이를 위하여 문제를 제약식이 있는 동적 계획법 모형을 생각한다. 그리고 이 이선형 문제를 풀기 위한 방법으로서 전통적인 동적 계획법의 방법과 제약식프로그래밍을 이용한 혼합 방법을 제시하였다. 무엇보다도 제안된 방법을 명확히 보여주기 위해 문제를 단순화한 모형을 이용하여 제안된 방법이 어떻게 문제를 풀어나가는지를 보여주고 이를 전통적인 방법을 확장하여 적용했을 때와 비교하여 설명하였다. 마지막으로 실제 상황을 기반으로 무작위적으로 인스턴스를 발생하여 전통적인 방식과 제안된 방식을 계산적으로 비교하고 새로운 방식에 대한 의미와 장점에 대해 논의하였다. 그 장점으로는, 첫째, 전통적인 방식에 따른 제약식의 상황을 상태 공간에서 정의함과 달리 제약식을 제약식프로그래밍 솔버에 맡김으로써 문제의 해결을 논리적으로 동적계획법과 제약식프로그래밍으로 분리하여 상태 공간에서의 복잡한 설계를 피할 수 있게 하였다. 둘째로 제약식프로그래밍의 사용은 상태 변수 설계의 회피로 사용자에게

친숙한 표현을 직접적으로 사용할 수 있게 하는 장점을 지닌다. 세째, 제약식프로그래밍 방법은 완전 일관성을 보장함으로 전통적인 방식이 불완전 일관성으로 인해 실현 가능해를 찾지 못하는 경우를 방지한다.

제안된 방식에 대한 현 단계의 연구는 여러 한계점과 추후 연구에 대한 이슈들을 제기할 수 있다. 먼저 단 하나의 문제 특히 그 문제를 단순화한 모형에 대해서만 적용하여 얻어진 결과에 대해서만 논하여 일반화를 위해서는 보다 더 많은 문제들에 대해 적용해 보아야 할 것이다. 물론 하나의 대단위 응용 문제에 대해 적용한 연구는 여러 가지 노력의 한계가 있을 수 있다. 이는 하이브리드 방식의 문제라기 보다는 전통적인 방식이 매 제약식별로 데이터구조의 설계를 요구하는 관계로 이를 구현하는데 많은 노력이 들어가는 것이 사실이다. 둘째로 현재의 하이브리드 방식의 구현은 두 가지 문제 해결기를 단순히 연결한 상태이다. 일반적인 동적계획법의 방식은 병렬처리가 가능하므로 두 해결기의 연결 방식을 병렬처리로 확장하는 방식의 연구를 통하여 하이브리드 방식의 계산을 개선할 수 있다. 세째로 제약식프로그래밍에 제약식 처리에서 어느 정도의 일관성을 이용한 추론 방법을 사용할 것인지에 대한 연구가 필요하다. 현재는 SWI-PROLOG의 FD 솔버가 기본으로 주는 추론만을 사용하였다. 또 FD 솔버가 구현하고 있는 여러 제약식 논리술부(predicate) 중 어떤 것들을 사용하는 것이 효과적일지에 대한 논의가 필요하다. 제약식프로그래밍에서는 각 논리술부에 따라 독특한 추론 방법이 정의될 수 있기 때문에 하이브리드 방식의 각 단계에서 완전 추론과 불완전 추론의 선택이 있을 수 있고 불완전 추론의 경우에는 각 추론의 정도에 대한 사용상의 가이드를 제시할 필요가 있다. 보다 완전한 추론을 선택하는 경우 계산상의 중첩되는 부분이 있을 수 있어 계산 속도를 떨어뜨릴 수 있고 보다 느슨한 추론은 최적해의 탐색에 실패할 수 있다. 그 사용 가이드를 통해 동적계획법의 틀 안에서 효율적으로 작동하는 제약식프로그래밍 솔버의 설계가 요구된다. 이것이 가능하면 이 연구의 계산 결과에서 보았던 계산 시간의 격차를 줄일 수 있을 것으로 판단된다.

본 연구는 제약식이 있는 동적계획법 모형을 푸는 새로운 방식을 제안하였다는 점에서 의의가 있고, 학술적인 측면에서, 이는 동적계획법의 열거 방식과 정수계획법이나 인공지능 기법에서 사용되는 일반적인 열거 방식들의 차이에 대한 생각들을 다시금 해보고 제약식이 부가됨으로써 동적계획법의 열거 방식을 가능하게 하는 문제의 조건에 대한 연구에 한걸음 다가서게 시도로서의 의미를 지닌다. 산업적으로 많은 설계와 계획의 문제들이 최적화 혹은 이에 준하는 모형으로 설정되고 이를 수치적으로 계산하는 의사결정 방식이 요구되는데 특이 이에 많이 사용되는 동적계획법과 강화학습 등의 인공지능 방법론을 개선함으로써 4차 산업혁명에 따른 자동화의 구조를 고도화하는데 기여한다고 생각된다.

REFERENCES

[1] H. H. Kim, H. J. Kim, “A comparison of fund-operational decision making in long-term car rental business using optimization,” Journal of the Korean Production and Operation Mangement Society, Vol. 29, No.2, pp 189-208, May 2018. DOI:10.21131/kopoms.29.2.201805.189

[2] D. P. Bertsekas, “Dynamic programming and optimal control,” Athena scientific, 1995.

[3] Y. Bukchin, T. Raviv, “Constraint programming for solving various assembly line balancing problems,” Omega, Vol. 1, No. 78, pp57-68, July 2018. DOI:10.1016/j.omega.2017.06.008

[4] J. Castro, “A stochastic programming approach to cash management in banking,” European Journal of Operational Research, Vol. 192, No. 3, pp. 963-974, February 2009. DOI:10.1016/j.ejor.2007.10.015

[5] R. E. Chatwin, “Continuous-time airline overbooking with time-dependent fares and refunds,” Transportation Science, Vol. 33, No. 2, pp. 182-191, May 1999. DOI:10.1287/trsc.33.2.182

[6] J. Chen, J. Wang, and P. C. Bell, “Lease expiration management for a single lease term in the apartment industry,” European Journal of Operational Research, Vol. 238, No. 1, pp. 233-244, October 2014. DOI:10.1016/j.ejor.2014.03.025

[7] N. P. Faisca, K. I. Kouramas, P. M. Saraiva, B. Rustem, and E. N. Pistikopoulos, “A multi-parametric programming approach for constrained dynamic programming problems,” Optimization Letters, Vol. 2, No. 2, pp267-280, March 2008. DOI:10.1007/s11590-007-0056-3

[8] E. A. Feinberg and A. Shwartz, “Constrained discounted dynamic programming,” Mathematics of operations research, Vol. 21, No. 4, pp922-945, November 1996. DOI:10.1287/moor.21.4.922

[9] J. G. Kallberg, R. W. White, and W. T. Ziemba, “Short term financial planning under uncertainty,” Management Science, Vol. 28, No. 6, pp. 670-682, June 1982. DOI:10.1287/mnsc.28.6.670

[10] D. Kizilay, P. Van Hentenryck, D. T. Eliiyi, “Constraint programming models for integrated container terminal operations,” European Journal of Operations Research, Vol. 286, No. 3, pp945-962, November 2020. DOI:10.1016/j.ejor.2020.04.025

[11] Y. Lan, M. O. Ball, I. Z. Karaesmen, J. X. Zhang, and G. X.

Liu, “Analysis of seat allocation and overbooking decisions with hybrid information,” European Journal of Operational Research, Vol. 240, No.2, pp. 493-504, January 2015. DOI:10.1016/j.ejor.2014.07.021

[12] B. Liu, C. Cheng, S. Wang, S. Liao, K. W. Chau, X. Wu, and W Li, “Parallel chance-constrained dynamic programming for cascade hydropower system operation,” Energy, Vol. 165, pp752-767, December 2018. DOI:10.1016/j.energy.2018.09.140

[13] J. I. McGill, and G. J. Van Ryzin, “Revenue management: Research overview and prospects,” Transportation science, Vol. 33, No. 2, pp. 233-256, May 1999. DOI:10.1287/trsc.33.2.233

[14] M. Ono, M. Pavone, Y. Kuwata, and J. Balaram, “Chance-constrained dynamic programming with application to risk-aware robotic space exploration,” Autonomous Robots, Vol. 39, No. 4, pp555-571, December 2014. DOI:10.1007/s10514-015-9467-7

[15] Y. E. Orgler, “An unequal-period model for cash management decisions,” Management Science, Vol. 16, No. 2, pp. B-77, October 1969. DOI:10.1287/mnsc.16.2.B77

[16] A. A. Robichek, D. Teichroew, and J. M. Jones, “Optimal short term financing decision,” Management Science, Vol. 12, No. 1, pp. 1-36, September 1965. DOI:10.1287/mnsc.12.1.1

[17] F. Rossi, P. Van Beek, and T. Walsh, “Constraint programming,” In Foundations of Artificial Intelligence. Elsevier. Vol 3, pp. 181-211, January 2008. DOI:10.1016/S1574-6526(07)03004-0

[18] C. Schulte, and M. Carlsson, “Finite domain constraint programming systems.” In Foundations of Artificial Intelligence. Elsevier. Vol. 2, 495-526, January 2006. DOI:10.1016/S1574-6526(06)80018-0

[19] B. C. Smith, J. F. Leimkuhler, and R. M. Darrow, “Yield management at American airlines,” interfaces, Vol. 22, No. 1, pp. 8-31, February 1992. DOI:10.1287/inte.22.1.8

[20] V. Srinivasan, and Y. H. Kim, “Deterministic cash flow management: state of the art and research directions,” Omega, Vol. 14, No. 2, pp. 145-166, January 1986. DOI:10.1016/0305-0483(86)90017-4

[21] J. Subramanian, S. Stidham Jr, and C. J. Lautenbacher, “Airline yield management with overbooking, cancellations, and no-shows,” Transportation science, Vol. 33, No. 2, pp. 147-167, May 1999. DOI:10.1287/trsc.33.2.147

[22] C. Zhang, C. Guo, and S. Yi, “Airline overbooking problem with uncertain no-shows.” Journal of Applied Mathematics, Vol. 2014, January 2014. DOI:10.1155/2014/304217

Authors

Tae Joon Park received the M.S. degree in Business Administration from Yonsei University in 2016 and has been in PhD. degree program since 2016. He is interested in machine learning algorithms, logistics optimization, and new technologies in the logistics industry.

Dr. Hak-Jin Kim received the B.S. and M.S. degrees in Business Administration from Yonsei University, Korea, the M.S. degree in Math from University of Illinois, Urbana-Champaign and the PhD. degree in Operations Research from Carnegie-Mellon University, Pittsburgh, U.S.A. in 2001. Dr. Kim is currently a Professor of School of Business at Yonsei University, Seoul, Korea. He is interested in the optimization theory, the constraint programming, the auction theory, network scheduling and algorithmic analysis for the variety of application problems.

Jinhee Kim received Bachelor’s degree of Business Administration and Statistics from Sookmyung Women’s University, Seoul, Korea, in 2019. She is recently in Master’s course of Business Administration at Yonsei University. She is interested in management science and data analytics.