한국지능정보시스템학회논문지 제10권 제2호 2004년 11월(pp.165~187)

워렌 추상기계와 한정도메인 제약식프로그램의 구조를 이용한 혼합형 문제해결기 구현에 대한 탐색적 연구

김학진
연세대학교 경영대학
(hakjin@yonsei.ac.kr)

제약식 프로그램과 최적화는 상이한 배경에서 출발하였지만 현실의 동일한 의사결정 문제의 해법을 시도해 왔다는데서 공통점을 가진다. 이 논문은 이 두가지 학문영역의 성과들을 하나의 문제풀이 틀에 통합하는 시도로서 혼합형 문제해결기의 구조를 제시한다. 특히 워렌의 추상기계와 한정도메인 제약식 프로그램의 문제해결기의 구조를 이용하여 최적화 특히 선형계획법의 문제해결기을 결합시키는 한 해법을 통합 모형의 틀을 통해 구체적인 구현의 단계로 제시한다. 또한 그 구현시 해결해야 할 문제들을 제시하고 그 해법을 논의한다.

논문접수일 : 2004년 8월 게재확정일 : 2004년 11월 교신저자 : 김학진

1. 서론

제약만족(Constraint Satisfaction, CS)은 전통적으로 컴퓨터과학에서 인공지능의 한 분야이다. 일상에서 실제로 닥치게 되는 많은 문제들이 수리 논리적인 구조를 띄고 있고 많은 부분이 수리논리식으로 표현될 수 있음으로 해서 이런 수리논리식으로 표현된 제약식들을 어떻게 효과적으로 풀 수 있는지를 이론적으로 다룬다. 이 분야가 관심을 갖는 문제들의 예로써 크로스워드 퍼즐, 체스의 N-퀸 문제, 암호 문제와 같은 퍼즐 문제로부터 보다 실제적인 지도의 색칠문제 같은 조합문제(Combinatorial Problem) 등이 있고 이를 효과적으로 풀기 위해 다양한 형태의 일관성(Consistency) 이론과 여러 효과적인 탐색기법을 제시하는 등 그 이론을 구축한다. 이와 같은 제약만족 문제(Constraint Satisfaction Problem, CSP)는 프로그래밍 언어적으로 프롤로그(Prolog)를 사용하게 되는데 그 주된 이유는 이 언어가 추론과 탐색에 적합하도록 설계되었기 때문이다. 하지만 이 언어가 지닌 추론 능력은 식의 대치와 같은 원시적인 추론 방법에 의존하고 있어 제약만족 문제를 풀기에는 한계가 있고 제약만족 이론의 다양한 추론 방법을 반영하지 못한다. 따라서 기존의 프롤로그 바탕에 제약만족식의 강력한 추론엔진을 더한 새로운 프로그래밍 언어 패러다임이 나타나게 되는데, 이를 제약식 프로그래밍(Constraint Programming, CP)이라 한다. 즉 제약만족 이론의 프로그래밍 언어 시스템으로 반영된 것이 제약식 프로그래밍이다. 이런 인공지능 이론의 진화 과정을 통해 제약만족 이론은 단순한 퍼즐 문제를 푸는 데에서 벗어나 본격적으로

현장의 문제를 푸는 정도로 발전하게 된다. 현장에서 벌어지는 복잡하게 얽혀있는 갖가지 작업일정계획 문제 등은 기존의 최적화 분야에서 다루어지기 어렵게 여겨진 문제들로 제약만족 추론엔진과 유연한 제약식 프로그래밍 언어의 표현을 통해 보다 쉽게 해결될 가능성이 열리게 되었다.

이 논문은 제약만족식의 추론엔진인 CP 문제해결기 중에 현실 문제 풀이에 가장 유용한 것으로 알려진 한정도메인 제약식 프로그래밍(Finite Domain Constraint Programming, FD)에 집중한다. FD는 기본적으로 이산형 문제에 효과적인 CP로서, 현실 세계에서 나타나는 이산형과 연속형의 혼합 시스템 문제를 풀기 위해서는 연속형 시스템을 다룰 수 있는 문제해결기의 기능이 필요하다. 이에 대한 해결로서 OR분야의 최적화 문제해결기와의 결합을 통해서 혼합형 시스템을 구축함이 제기되었다. 다음 절에서는 이와 같은 FD와 OR의 문제해결기의 결합에 대해 지금까지 어떻게 연구되어 왔는지 좀더 자세히 살펴보게 될 것이다. 이 논문에서는 Hooker가 제시한 새로운 모델링 언어의 틀 안에서 FD와 LP솔버를 결합한 혼합형 방법에 기초하여 논의를 전개한다. 즉 논문의 초점은, 주로 CP시스템을 구현하는데 사용되는 프롤로그, 좀더 정확하게 말하면 워렌의 추상기계(Warren Abstract Machine, WAM)와 CP 문제해결기가 만나 어떻게 FD 시스템이 구현되는지를 보여주고, 이 위에 Hooker의 혼합형 방법의 가장 기본적인 추론을 반영하는 조건제약식을 통하여 FD 시스템이 OR의 최적화 솔버와 어떻게 결합되어 혼합형 문제해결기로 구현되는 지를 보여주는데 맞추어져 있다. 특히 혼합형 문제해결기의 구현시 고려되어야 할 구체적인 이슈들을 시스템 구현의 입장에서 제시하고 이 문제들이 WAM, FD 문제해결기의 구조와 최적화 문제해결기와의 결합 구조를 통해서 어떻게 해결되는지를 제시한다.

이를 위해 이 논문은 다음과 같이 구성된다. 둘째 절에 혼합형 문제해결기 기술에 대한 짧은 문헌연구가 따르고 셋째에 이 논문에서 이야기하고자 하는 혼합형 문제해결기의 기본적인 개념에 대해 언급한다. 특히 본 혼합형 모형과 접근방법에서 혼합형 조건 제약식이 제안되고 그것이 어떻게 보다 복잡한 제약식을 해결하는 근간이 되는지를 설명한다. 셋째와 네째에서 혼합형 문제해결기를 구축하기 위해 이 논문에서 기저로 삼는 WAM 구조와 FD 문제해결기 구조에 대해 살펴본다. 그리고 다섯째로 이 논문에서 말하고자 하는 혼합형 문제해결기를 WAM과 FD위에 구축할 때 제기되는 문제들을 살펴보고 그 해결책을 제시한다. 마지막으로 결론과 앞으로의 연구방향에 대한 제안으로 글을 맺는다.

2. 문헌연구

제약식 프로그래밍과 정수계획법(The Integer Programming, IP) 혹은 선형계획법(The Linear Programming, LP)을 통합한 혼합형 문제해결기에 대한 아이디어는 최근 여러 논문에서 제시된 바 있다(Smith et al. 1995; Darby-Dowman and Little, 1998). 그 논문에 따르면 각 사용하는 방법들이 갖고 있는 특징들에 따라 실험적인 결과들을 도출해 낸 바 있다. 특히 정수계획법은 좋은 완화문제(Relaxation Problem)을 갖는 경우에 효과적인 것으로 나타나 완화문제의 한계값이 약하거나 그 모형언어(Modeling Lanauage)에서의 약점

으로 인해 크기가 큰 모형을 설정할 시에 문제 풀이에 많은 어려운 점들을 내포한다. 제약식 프로그래밍(Constraint Programming, CP)은 기존의 최적화 모형과는 달리 풍부한 표현 제약식들을 가지고 있어 모형의 표현에 있어 크기가 상대적으로 작은 모형을 생산해낼 수 있고, 고도로 제약이 된 문제에서 강점을 드러낸다. 하지만 CP 완화법은 상대적으로 각 제약식의 국지적 특징들을 이용하였기 때문에 IP에 비해 전국적이지 못하다.

이러한 서로의 약점을 보완하기 위해 앞의 두 가지 접근법을 통합하는 노력이 있어왔다. (Beringer and De Backer, 1995)는 CP의 한계 확산법(Bounds Propagation)과 고정 변수를 이용하여 CP와 LP를 결합하는 방법에 대해 생각해 본 적이 있다. (Rodosek et al. 1997)은 변수 도메인에 있는 값을 추려내고 값의 범위를 한정짓기 위해 한 탐색나무(Search Tree)에서 CP 문제해결기와 LP 문제해결기를 함께 사용하였다. 탐색나무의 각 노드에서 풀어야하는 하위문제는 몇 가지 경우에 실현 불가능성을 규정하는데, 확산법(Propagation)으로 한 변수의 도메인을 공집합하거나, LP 문제해결기를 통해 완화문제가 실현 불가능함이 판명되거나, 완화문제의 목적함수 값이 현재 최적값보다 열등할 때가 그러하다. 그들의 아이디어는 문제를 CP로써 모형화하고 이 모형에 대해 체계적으로 "그림자" IP모형을 설정할 수 있는 방법을 과정으로 만들었다. 그 방법은 재 설정된 수식 제약식과 **alldifferent** 제약식 등을 포함한다. 그 위에 모형기(Modeler)가 구축되어 제약식들에 어떤 문제해결기를 - CP, LP 혹은 두가지를 동시에 - 이용할지를 결정한다. 상징형 제약식을 보다 잘 다루기 위한 결합을 목적으로 하는 연구도 진행되었다. (Hajian et al. 1996; Hajian, 1996)은 IP 문제해결기가 보다 잘 다룰 수 있는 방법을 보였다. 그리고 이를 통해 **alldifferent** 제약식을 선형 모형에서 어떻게 다룰 지를 제시하였다. Bockmayr와 Kasper(Brockmyr and Kasper, 1998)은 CP와 IP를 통합하기 위해 흥미로운 통합 틀을 선보였다. 그 틀에서 몇 가지의 통합이 가능한데, 어떻게 상징형 제약식이 절면 제약식처럼 IP에 통합이 가능한지, 선형 부등식 체계가 어떻게 상징형 제약식으로써 CP로 통합될 수 있는지를 보였고, 선형 부등식과 변수의 도메인이 같은 제약식 저장소(Constranit Store)에 저장될지를 논의했다.

이상의 CP와 최적화 문제해결기의 통합을 위해 택해진 방법들은 주로 모형 언어 상에서 두가지의 접근방식을 채택하고 있다. 첫째로, 서로 다른 모형언어를 가진 문제해결기들을 병렬로 연결하여 문제를 푼다. 이때 문제는 CP와 최적화 두 모형으로 동시에 표현되어서 문제해결기의 풀이 과정에서 나오는 정보는 다른 문제해결기에서 이용될 때 그에 해당하는 모형으로 표현되어야 한다. 따라서 이 방법은 진정한 의미의 통합이라기보다는 두가지 모형 두가지 문제해결기를 문제풀이를 위해 연결하여 필요에 따라 다른 형태의 문제해결기로 문제를 해결하려는 시도로 볼 수 있다. 둘째는 CP와 IP의 모형을 어느 하나에 종속을 시키는 방법이다. 이 방법 또한 각 문제해결기가 채택하고 있는 모형 언어의 장점들을 포기하게 됨으로 통합의 효과를 극대화하지 못한다.

위의 접근방법과는 다르게 통합의 수준을 모형 언어에서부터 다루는 방법이 있다. 즉 각 문제해결기에서 사용해야 하는 모형 언어의 장점들을 모두 포용하는 새로운 모형 언어를 만들어 이 모

형 언어에 적합한 문제해결기를 주어진 두가지 문제해결기의 기술을 결합하여 만든다. 이때 제시된 새 모형 언어의 구조로부터 두 문제해결기 기술이 결합되는 양상이 결정된다(Hooker and Osorio, 1999; Hooker et al. 2001; Kim and Hooker, 2002). 국내에서 발간된 논문(김학진, 2003)은 문제해결기의 결합이 모형 언어로부터 시작되어야 함을 강조하고 두 문제해결기의 기술이 결합되는 방식에 대한 아이디어를 제시하고 있다. 본 논문은 이 아이디어를 구체적으로 워렌 추상기계와 한정도메인 문제풀이기의 구조를 이용하여 문제해결기의 결합을 적용할 때에 나오는 이슈들을 제시하고 특히 가장 기본적인 형태의 제약식을 구현함을 통해 그 통합의 구체적인 해법을 모색한다.

다음 절에서는 이와 같은 접근방법이 갖는 핵심적인 개념과 아이디어를 제시하고 이 접근방법에 의거한 새로운 혼합형 문제해결기의 구현에 대해 이 후의 절에서 논의한다.

3. FD-LP 혼합형 접근방법

이 절에서는 FD-LP 혼합형 접근방법을 모형 언어적인 측면에서 이 방법이 문제를 서술하는데 전통적인 수리계획법적인 방법보다 얼마나 풍부한지를 보여주고 이 모형이 근본적으로 가장 간단한 구조의 혼합형 조건제약식으로 환원됨을 보여준다. 그리고 모형을 풀기 위한 알고리즘이 어떻게 되는지를 보여준다.

가장 기본적인 형태의 혼합형 모형은 다음과 같이 설정된다.

$$\begin{aligned} &\min\ cx \\ &\text{s.t. } h_i(y) \rightarrow A^i x \geq b^i,\ i \in I \\ &\quad x \in R^n,\ y \in D \end{aligned} \tag{1}$$

여기서 y는 이산형 변수의 벡터이고 x 는 연속형 변수의 벡터이다. $h_i(y)$는 y만을 포함하는 제약식이다. 어떤 y의 값이 이 제약식을 만족하면 TRUE의 값을 갖는다. 이는 일반적인 형태로 만약 조건없이 선형부등식이 충족되기 위해서는 다음과 같은 형식으로 표현될 수 있다.

$$(0=0) \rightarrow Ax \geq b$$

마찬가지로 선형부등식이 부과되지 않는 이산형 변수의 제약식도 역시 형식상 다음으로 표현된다.

$$\neg h_i(y) \rightarrow (1=0)$$

이 형식은 또한 논리적인 OR를 나타내는 제약식을 표현하는데도 편리하다. 예를 들어 고정비용함수를 나타낼 때

$$f(x) = \begin{cases} 0, & x=0 \text{ 일때} \\ F+cx, & x>0 \text{ 일때} \end{cases}$$

이는 연속형 변수를 도입함으로써 다음과 같은 형태로 변형된다.

$$\begin{aligned} &(y=0) \rightarrow x=z=0 \\ &(y=1) \rightarrow (x>0, z=F+cx) \end{aligned}$$

y는 {0,1}을 도메인으로 가지는 이산형 변수이다. 좀더 복잡한 형태의 혼합형 모형을 생각한다면 모형 설정에 유용한 도구는 변수첨자이다. 예를 들어 할당문제에서 작업자 k를 작업 j에 할당하는데 드는 비용이 c_{ij} 라면 목적함수는 $\Sigma_j c_{j,y(j)}$

와 같이 나타낼 수 있다. 이때 y_j는 작업 j에 할당된 작업자를 나타내는 이산형 변수이다. $c_{j,y(j)}$의 값은 사실상 벡터 변수 $y=(y_1,\ldots,y_n)$의 함수이고 $g_j(y)$와 같은 형태로 쓸 수 있는데 이때 g_j는 y_j의 값에만 의존한다. 이와 같은 $g_j(y)$를 변동 상수(variable constant)라고 볼 수 있고 마찬가지로 연속형 변수의 첨자를 y의 함수로 보았을 때 변동첨자(variable subscript)라고 부른다. 이런 모형 설정의 장치들을 사용해서 다음과 같은 혼합형 모형이 가능하다.

$$\begin{aligned}&\min\ cx\\&\text{s.t.}\ \ h_i(y)\rightarrow\sum_{k\in K_i(y)}a_{ik}(y)x_{j_{ik}(y)}\geq b_i(y),\ \ i\in I\\&\qquad x\in R^n,\ y\in D\end{aligned}\qquad(2)$$

여기서 $a_{ik}(y)$와 $b_i(y)$는 변동상수이고 $j_{ik}(y)$는 변동첨자이다. 덧붙여 위의 모형은 $K_i(y)$라는 변동 인덱스집합을 보여주고 있다.

위의 복잡한 형태의 모형은 기본적인 형태의 충분히 많은 혼합형 조건제약식의 형태로 분해된다. 예를 들어, $y(j)\in\{1,\ldots,n\}$이라고 하고 다음의 제약식들이 모형에 추가되면

$$(y(j)=k)\rightarrow(z_j=c_{jk}),\ \ \forall j,k\in\{1,\ldots,n\}$$

제약식 $z\geq\sum_j c_{j,y(j)}$은 $z\geq\sum_j z_j$로 표현되어 모형에 추가된다. 물론 식(2)를 이 같은 방식으로 분해하는 것은 비효율적이다. 혼합형 문제해결기가 각각의 모형설정의 장치인 변동상수, 변동첨자, 그리고 변동 인덱스집합을 $y(j)$ 값이 변화되고 고정됨에 따라 해당하는 조건제약식의 결과를 LP 문제해결기에 강제하도록 구현되어야 한다. 하지만 위의 예에서 보여주는 것처럼 원칙상 혼합형 접근방법에서 다양한 형태의 제약식이 혼합형 조건제약식으로 환원되고 이 모형을 푸는 혼합형 문제해결기의 입장에서 조건제약식을 어떻게 다루느냐는 보다 완전한 문제해결기의 구현에 필수적인 요소이다.

다음 예들은 혼합형 접근방법으로 표현된 모형들이다. 첫째로 판매원 경로 문제 (traveling salesman problem)는

$$\begin{aligned}&\min\ z\\&\text{s.t.}\ \ z\geq\sum_{j=1}^{n}c_{y(j),y(j+1)}\\&\qquad\text{all-different}\{y(1),\ldots,y(n)\}\end{aligned}$$

이때 $y(j)\in\{1,\ldots,n\}$이고 $y(n+1)=y(1)$이다. 그리고 $y(j)$는 판매원이 방문하는 j번째 도시이다. 혹은 다른 제약식을 사용하여

$$\begin{aligned}&\min\ z\\&\text{s.t.}\ \ z\geq\sum_{j=1}^{n}c_{j,y(j)}\\&\qquad\text{cycle}\{y(1),\ldots,y(n)\}\end{aligned}$$

이라 할 때 $y(j)$는 도시 j이후에 방문하는 도시이다. 그리고 제약식은 판매원이 순회하는 경로를 나타낸다. 즉 도시 $q_1,\ldots,q_n$을 차례로 방문할 때 이들 도시는 다 다른 도시를 나타내고 $q_1=1$이고 $j<n$에 대해 $q_{j+1}=y(q_j)$이다. 또 다른 예로 지체시간을 최소화하는 한 기계 일정계획 문제는 다음과 같이 표현된다.

$$\begin{aligned}&\min\ \sum_j z_j\\&\text{s.t.}\ \ z_j\geq t_j+d_{y(j)}-f_{y(j)},\ \ \forall j\\&\qquad t_{j+1}\geq t_j+d_{y(j)}+s_{y(j),y(j+1)},\ \ \forall j<n\\&\qquad\text{all-different}\{y(1),\ldots,y(n)\}\\&\qquad t_1=0\end{aligned}$$

여기서 t_j는 j번째 계획된 작업이 시작되는 시간이고, d_i는 작업 i의 경과시간, s_{ik}는 작업 i가 작업 k의 선행 작업일 때의 셋업시간, 그리고 f_i는 작업 i의 만료 시간이다. 변동 인덱스집합을 사용하는 예로 자원제약 조건하에서의 1 기계 문제가 있다. 작업들이 일렬로 이루어지기 보다는 전체 자원 소비율에서 일정한 한도을 초과하지 않는 범위에서 동시에 작업이 이루어지는 경우가 있다. 만약 작업 i가 r_i의 비율로 자원을 소비하면 어느 한 순간의 프로세스에 있는 모든 작업들 i에 대해 r_i의 합은 R을 초과하지 않는다. 정수계획 문제로 모형을 만드는데 어려움이 있지만 혼합형 모형은 변동 인덱스 집합으로 쉽게 설정된다.

$$\begin{aligned} \min \ & \sum_i z_i \\ \text{s.t.} \ & z_i \geq t_i + d_i - f_i, \ \forall i \\ & \sum_{k:t_k \leq t_i \leq t_k + d_k} r_k \leq R, \ \forall i \\ & t_i \geq 0, \ \forall i \end{aligned}$$

여기서 t_i는 작업 i의 시작 시간이고 z_i는 작업 i의 지체시간이다. [그림 1]은 과정 언어로 표현된 주어진 혼합형 모형을 푸는 일반 알고리즘이다.

1: $\overline{D_j}$: 이산형 변수 y_j의 현재 도메인
2: $L_i(x): A^i \geq b^i$
3: A : 도메인 벡터 $\overline{D} = (\overline{D_j})$로 정의된 활성 노드들의 집합. 최초로는 {D}이다.
4: $\overline{z}$: 최적값에 대한 상계, 최초로 무한대 값으로 설정된다.
5:
6: **while** $A \neq \varnothing$ **do**
7: A로부터 노드 $\overline{D}$를 제거한다.
8: 확산법을 적용한다.
9: $L \Leftarrow \cup_{i \in I} \{L_i(x) \mid L_i(x)$는 $\overline{D}$에 의해 강제된다$\}$
10: $z \Leftarrow \min\{cx \mid L\}$
11: **if** $z < \infty$ **then**
12: **if** 모든 $\overline{D_j}$는 원소가 하나인 집합 **then**
13: **if** $z = -\infty$ **then** stop (무계 문제) **end if**
14: **if** $z < \overline{z}$ **then**
15: $\overline{z} \Leftarrow z$
16: $\overline{x} \Leftarrow \arg\min\{cx \mid L\}$
17: $\overline{y} \Leftarrow (d_1, \ldots, d_n)$ 이때 $\overline{D_j} = \{d_j\}$
18: **end if**
19: **else**
20: 가지치기를 해서 노드 $\overline{D}^1, \ldots, \overline{D}^K$를 A에 추가 이때 $\overline{D}_j^k \subset \overline{D}_j$
21: **end if**
22: **end if**
23: **end while**

[그림 1] 혼합형 접근방법의 일반적인 알고리즘

4. 워렌의 추상기계(The Warren Abstract Machine)

알레인 콜머라우어(Alain Colmerauer)가 프롤로그를 고안해낸 이후, 프롤로그는 로버트 코발스키(Robert Kowalski)의 논리 프로그래밍(Logic Programming) 개념을 구현하는 그 기본 구현체가 되어왔다. 최초의 구현은 포오트란을 이용해서 이루어졌고, 그 뒤 에딘버그 대학에서 DEC-10 Prolog라는 시스템이 만들어졌으며 이는 이 후의 현대의 모든 프롤로그의 기본적인 근간을 이루는 시스템이 되었다. 워렌의 추상기계는 이 에딘버그에서 개발된 시스템의 아이디어를 추상기계로 추상화한 시스템이다. 따라서 WAM은 현대의 프롤로그 시스템의 기본 구조를 이루는 엔진이라고 할 수 있다(Ait-Kaci, 1999).

WAM의 기본 구조는 모든 논리 프로그래밍을 이루는 요소들을 저장할 수 있는 메모리 구조체와 그 위에서 그 메모리 구조체를 조작하는 명령어 집합으로 이루어진다. 좀더 자세히 말해서 그 메모리 구조체는 3가지의 스택(stack) 구조로서 이루어졌다. HEAP, E-STACK, TRAIL로 표시될 수 있는 세 개의 스택이다. 만약 어떤 논리 프로그램이 WAM의 구현체에 주어지게 된다면 모든 용어(term)에 대한 선언과 정의는 HEAP 스택에 저장되게 된다. 논리 프로그래밍의 계산 모형은 선언된 문장(statement)을 일정한 제어를 통해서 연역적으로 해석, 즉 문장의 선언된 정의에 따라 해석하는 것이기 때문에, 각 선언의 정의에 대한 해석이 이루어지는 그 환경을 정의할 필요가 있다. 그래서 이 환경은 E-STACK이라는 저장소에 저장하게 된다. 이 환경은 프로그램의 계산에서 한 용어가 호출될 때 그 용어를 구성하는 서술어(Predicate)의 원자(atom), 전달되는 인자들과 그 인자들의 수 등으로 이루어진다. 논리 프로그래밍은 단순한 연역적인 해석으로는 해결될 수 없는 문제를 해결하기 위해 이에 더하여 비결정주의 방식 (Nondeterminism)을 그 계산 모형에 도입한다. 예를 들어 논리 프로그램에서 한 서술어가 OR의 관계를 갖는 논리식(disjunction)으로 이루어진 경우 이 식의 환경을 기억하는 E-STACK에는 선택점(choice point)이 입력된다. 이 선택점은 여러 대안을 열거 탐색하기 위해 사용된다. 만약 한 선택점에서 하나의 대안이 선택되고 그것을 따라 연역적인 해석을 계속해 나갈 때 결국에 가서는 이 대안으로 말미암아 논리 불일치의 상태에 도달한다면 지금까지 해온 해석들을 모두 돌이키고 그 대안이 선택되었던 선택점으로 되돌아오는 백트랙킹(backtracking)의 연산을 해야한다. 이때 대안 선택을 통해 이루어진 모든 해석은 E-STACK에 차례로 저장되고 백트랙킹은 이 스택을 거꾸로 거슬러 올라가 대안 선택의 선택점에 까지 도달하게 되어 다른 대안을 선택하도록 하게 만든다. E-STACK을 거슬러 올라가는 동안 시스템의 상태를 대안이 선택되기 전의 상태로 환원되어야 하는데 이에 필요한 정보는 TRAIL이라는 스택에 저장된다. TRAIL에는 선택점 이후로 이루어지는 모든 해석들을 어떻게 되돌리는지에 대한 정보들이 차례로 쌓이게 되고 백트랙킹의 과정을 통해 이것이 하나하나 출력되어 상태를 되돌이키는데 이용되는 것이다. 따라서 논리 프로그래밍의 비결정주의적인 연산은 E-STACK과 TRAIL의 협조적인 방식을 통해 WAM에서 구현된다.

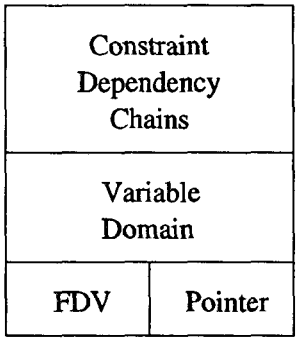
5. 제약식 프로그래밍 솔버의 구현체

전통적으로 제약식 프로그래밍은 프롤로그 구현체 위에 구현되어 왔다. 우선 그 한 이유로는 제약식 프로그래밍이 논리 프로그래밍 분야로부터 유래되었기 때문이기도 하지만 이보다 더 강한 이유로 프롤로그가 그 언어의 중심에 비결정주의 연산을 자연스럽게 구현하고 있기 때문이라고 볼 수 있다. 즉 제약식 프로그래밍은 NP-hard의 조합문제(combinatorial problems)들을 대상으로 하고 있고 그 문제들을 푸는데 비결정주의 연산이 중심적인 역할을 하기 때문에 프롤로그에 이미 구축된 WAM 구조는 제약식 프로그래밍 솔버를 구축하는데 커다란 도움이 된다.

한정도메인 제약식 프로그래밍(Finite-Domain Constraint Programming, FD) 문제해결기를 구축하는데 gprolog에서는 C-STACK을 도입한다. (Codognet & Diaz 1995, 1996, 2001; Diaz 2002; Diaz & Codognet 2000) 그리고 이 제약식 프로그래밍에서 설정되는 FD 제약식 문제를 표현하기 위해 세가지 객체가 도입되는데 FD-Frame, C-Frame 그리고 A-Frame이 그것이다. FD 제약식은 FD 자체적인 원자와 변수들을 결합하여 FD 서술어로 표현되고 이것이 문제해결기에 주어지게 된다. 주어지는 FD 문제는 보통 다음과 같은 형태를 띄게 된다.

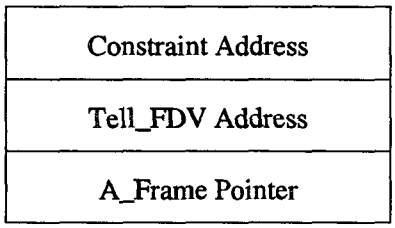
```
fd_domain([S,M],1,9),
fd_domain([E,N,D,O,R,Y],0,9),
fd_all_different([S,E,N,D,M,O,R,Y]),
(1000*S+100*E+10*N+D)+(1000*M+100*O+10*R+E)
#= 10000*M+1000*O+100*N+10*E+Y,
fd_labeling([S,E,N,D,M,O,R,Y]).
```

위의 예에서 **fd_domain**, **fd_all_different**, **fd_labeling**, 산술 연산자들 +, * 그리고 관계 연산자 #=은 FD 문제해결기의 라이브러리에서 이미 구축된 원자들이고 이것들이 FD 변수들 S,E,N,D,M,O,R,Y과 결합되어 서술어를 이루어 문제를 형성한다. 문제에서 보는 것처럼 모든 변수들은 FD 변수들로 이루어져, FD-Frame의 형태로 C-STACK에 저장된다. 즉, [그림 2]에 나타난 것처럼 크게 네 가지 요소가 있고 그 중 첫 번째 요소로 FDV는 이 구조체의 유형을 나타내는 마스크이다. 즉 그 구조체가 FDV값을 갖고 있으면 이것이 FD 변수임을 나타낸다. 이와함께 Pointer는 유니피케이션(Unification)을 위해 사용되는 주소값이다. 무엇보다도 중요한 정보를 포함하고 있는 부분은 Variable Domain과 Constraint Dependency Chains 부분이다. 전자는 해당 FD 변수가 갖는 값을 나타내는 도메인이다. 보통은 하한값과 상한값을 나태내는 구간의 형태로 표현하는 경우가 있고 이때는 주로 부분일관성(partial consistency) 기법을 확산법으로 사용할 때 이러한 표현을 사용한다. 또 다른 방법으로 모든 대안값을 포함한 집합으로 나타낼 수 있는데 이런 표현방식은 주로 완전일관성(full consistency) 기법을 사용할 때 사용된다. 후자 즉 Constraint Dependency Chains는 다음에 설명할 제약식을 나타내는 C-Frame의 주소를 체인으로 모아놓은 것으로 즉 현재의 FD-Frame에 해당하는 변수를 포함하고 있는 모든 제약식의 주소를 모아둔다. 이것을 통해 변수의 도메인이 바뀌었을 때 이 체인에 있는 제약식의 주소를 알고 그 제약식 위에 확산법을 적용해 변수의 도메인의 변화를 다른 변수의 도메인의 변화로 확장해 나간다. 이 체인은 여러가지 형태의 체인으로 이루어져서 각 형태마다 독특한 확산법을 적용한다.

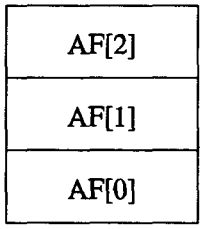


[그림 2] FD-Frame 구조

문제의 FD 제약식은 C-Frame을 통해서 표현이 된다. [그림 3]이 보여주는 것처럼 이는 세가지 요소로 구성된다. 첫째 요소 Constraint Address는 실제 그 제약식이 어떻게 계산분해되어 변수의 도메인에 영향을 끼쳐야 하는 지를 정의하고 있는 그 제약식만의 확산법이 함수의 형태로 프로그램 코드 스택에 저장되게 되고 이 함수의 주소가 바로 C-Frame의 첫째 요소로 저장된다. 둘째 요소 Tell_FDV Address는 그 제약식이 확산법에 의해 계산분해될 때 영향을 끼치는 변수의 FD-Frame



[그림 3] C-Frame



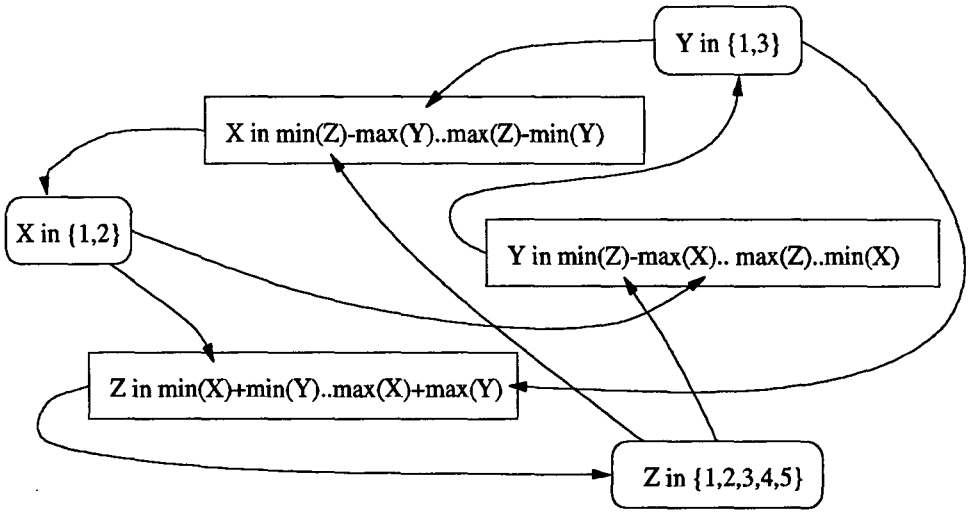
[그림 4] A-Frame

의 주소가 저장된다. 마지막으로 주어진 문제에서 FD 제약식이 프롤로그의 서술어로 정의되었을 때 그 서술어의 인자들이 [그림 4]에서와 같이 A-Frame으로 저장되는데 이 주소가 A_Frame Poninter에 기록된다. 이 제약식의 확산법이 가동되었을 때 시스템은 Constraint Address에서 지시하는 코드를 실행하게 된다. 이때 필요한 인자를 구하기 위해 해당하는 A-Frame에서 정보를 얻게 되는데 이는 A_Frame Pointer를 통해서 얻어진다.

이 같은 세가지 형태의 프레임은 FD 문제해결기에서 확산법을 통해 가장 기본적인 연산을 수행할 수 있도록 하는 변수-제약식 네트워크를 견고하게 구성하여 C-STACK에 생성한다. [그림 5]는 FD 변수 X, Y, Z가 주어지고 제약식 X+Y=Z가 주어졌을 때 변수-제약식 네트워크가 어떻게 형성되는지를 보여준다. 일단 주어진 제약식은 보통 원시제약식(primitive constraint)라고 부르는 여러 개의 기본적인 제약식으로 분해된다. 우리의 예에서는

- X in min(Z)-max(Y)..max(Z)-min(Y)
- Y in min(Z)-max(Y)..max(Z)-min(X)
- Z in min(X)+min(Y)..max(X)+max(Y)

로 분해된다. 만약 변수들의 각각의 도메인이 {1,2}, {1,2}, {1,2,3,4,5}라고 한다면 제약식으로부터 Z=1의 값은 결코 성립할 수 없고 확산법에 의해 이는 변수 Z의 도메인으로부터 제거된다. 물론 Z=5의 값 역시 성립하지 않기 때문에 제거될 수 있지만 이것이 한꺼번에 일어날지 아니면 시간상의 차이를 두고 일어날 지는 확산법의 형태에 따라 달라질 수 있다. 일반적으로 확산법에서 적용되는 확산 정도를 나타내는 개념으로 제약만족



[그림 5] FD 변수-제약식 네트워크

이론(Constraint Satisfaction)의 일관성기법(consistency technique)이 사용되는데 여러 가지 형태의 기법들이 있다.

제약식 프로그램은 WAM구조 위에서 움직이는데 이는 앞에서 언급한 데이터 구조들이 WAM 상의 프롤로그 구조체에 연결됨을 의미한다. 이 연결은 직접적으로 프롤로그 변수와 FD 변수 그리고 프롤로그 서술어와 FD 제약식간에 만들어진다. 즉 제약식 프로그램 문제가 프롤로그의 서술어 형식으로 주어지면 WAM은 해당 프롤로그 구조체를 만들고 이것이 제약식 프로그램에 속하는지를 따져본다. 만약 그렇다면 WAM은 FD 문제해결기를 통해 해당하는 FD의 구조체들을 생성하여 프롤로그 구조체에 연결해 준다. 이 후에 FD 문제해결기가 WAM의 도움으로 FD 구조체를 통해서 문제풀이를 담당한다. 이때의 WAM의 도움은 선택점의 생성, 선택대안의 채택, 백트랙킹 등으로 프롤로그 구조체와 FD 구조체의 연결이 이를 가능하게 한다.

6. 혼합형 문제해결기의 구현

6.1 어떻게 작동되는가.

FD 문제해결기를 LP 문제해결기와 통합하기 위해 필요한 것은 혼합형 제약식이라는 새로운 형태의 제약식이다. 혼합형 제약식은 문제의 모형언어와 하부의 문제해결기 구현에서 FD와 LP를 결합하는 역할을 지니고 있다. 새로운 혼합형 문제해결기는 이런 새로운 제약식을 구현할 수 있는 틀을 제공해야 한다. 단순히 프롤로그 수준에서 이미 구현된 라이브러리를 이용해 gprolog의 FD 문제해결기를 확장하는 것으로는 이 목표를 이룰 수 없다. 첫째로 외부의 LP 문제해결기를 FD 문제해결기에 연결할 수 있어야 하는데 혼합형 문제해결기의 구현을 위해 이 논문에서 다루는 gprolog에는 외부의 시스템과 연결할 수 있는 인터페이스가 있다. 이 기능을 통해 gprolog는 사용자가 원하는 프롤로그 서술어를 구축하게 한다.

또한 gprolog의 FD 문제해결기는 내부에 작은 매크로 언어를 포함하여 사용자가 FD 제약식을 설계할 수 있는 기능도 제공한다. 문제는 혼합형 문제를 다루기 위해 혼합형 제약식을 어떻게 구현을 하고, 이 제약식을 FD 문제해결기와 LP 문제해결기와 어떻게 연결하며 서로간의 상호작용이 일어나게 하는가, 그리고 WAM 구조를 이용해서 어떻게 비결정주의적인 연산에서 일어나는 백트랙킹을 제어할 것인가 하는 것이다.

gprolog는 C수준에서의 인터페이스를 지닌다. C 수준에서 이진함수가 선언되면 이 것은 :-foreign(Templates, Options) 형태의 프롤로그 명령어로 선언이 되면서 선언된 C 이진함수를 프롤로그의 서술어로서 사용할 수 있다. 그리고 하부에 C로 선언된 여러 함수들을 통해 데이터를 주고 받으며 서술어의 움직임을 통제한다[자세한 것은 gprolog매뉴얼을 참조]. 따라서 이 인터페이스를 통해 LP 문제해결기의 함수들을 C 이진함수로 포장하고 이를 프롤로그 서술어로 정의함으로써 프롤로그 수준에서 LP 문제해결기를 통제할 수 있게 한다.

혼합형 제약식의 구현은 FD 제약식이 구현되는 방식과 유사하다. 왜냐하면 프롤로그 시스템에서 동일한 형태를 띄게 되고 또한 계산시 제약식이 다루어지는 것 역시 FD 문제해결기의 틀 안에서 다루어지기 때문이다. 일반적으로 FD 문제해결기에서 이루어지는 제약식 확산법(propagation method)이 혼합형 제약식에서도 동일하게 적용된다. 다른 점이 있다면 확산을 통해 얻어지는 것이 다르고 이것이 LP 문제해결기를 통제하게 된다는 것이다. 따라서 FD 문제해결기의 확산법을 적용하기 위해 혼합형 제약식도 FD 제약식과 마찬가지로 C-STACK에 구현된다. 하지만 혼합형 제약식은 FD 문제해결기에 내재하는, 사용자 정의 FD 제약식을 만드는, 매크로 언어로는 정의될 수 없다. 그 이유는 혼합형 제약식은 확산법에서 계산을 위한 분해(resolution)시 다른 방식으로 이루어지고 이로 인해 가동되는 계산 방식이 FD 제약식과는 다르게 이루어지는데 FD의 매크로 언어에서는 FD 제약식을 위한 고정된 방식이 존재하기 때문이다. 그래서 혼합형 문제해결기의 구현에서는 혼합형 제약식의 나름대로의 방식이 구현되어야 한다.

마지막 이슈는 혼합형 문제해결기의 백크랙킹에 대한 부분이다. FD 문제해결기에서의 백트랙킹은 WAM 구조에서 관장한다. WAM은 선택점의 스택을 가지고 있고 각 변수들의 도메인에 대한 정보를 어떻게 추적할 것인지에 대한 정보를 가지고 있다. FD 프로그램을 계산하는 과정에서 비결정주의 연산에 의해 선택점을 생성하고 한 대안을 선택 계산을 진행하다가 그 선택 대안에 대해 해가 존재하지 않는 FALSE로 결론이 나면 그 대안을 생성한 선택점으로 되돌아가 다른 대안을 선택한다. 이때 선택점으로 되돌아가는 백트랙킹에서 필요한 정보는 여러 대안을 생성하는 그 선택점과 한 대안을 선택한 이후 FD의 확산법의 과정에서 WAM의 유니피케이션(Unification)을 통해 각 변수의 도메인이 축소될 때 변화되기 이전의 상태들이다. 이 정보들은 선택점의 경우 E-STACK에, 도메인의 변화 이전의 상태는 TRAIL이라는 스택에 저장이 되었다가 백트랙킹 시 팝-아웃(pop-out)을 통해 선택점 당시의 상태로 회복된다.

6.2 혼합형 조건제약식의 계산 분해 과정 (resolution)

FD-LP 통합 혼합형 모형의 가장 기본적인 혼합형 제약식은 조건제약식이다. 주로 논리의 추론을 나타내는데 사용이 되는 제약식으로 그 기본 형태는 조건과 결과 그리고 그 둘을 잇는 추론 연산자이다. 즉,

$$h(y) \rightarrow Ax \geq b \qquad (3)$$

의 형태를 지니다. 이때 조건 부분인 h(y)는 FD 제약식이고 결과 부분은 LP의 전형적인 선형부등식 군이다. 혼합형 문제해결기가 이와 같은 제약식을 다루는 과정은 FD 제약식을 FD 문제해결기에서 다루는 과정과 다르다. 논리적으로 어느 해 (y,x)가 $h(y)$에서 TRUE의 값을 가지고 부등식이 만족되지 못하면 이 조건 제약식은 만족되지 못한다.

일반적으로 FD 문제해결기는 논리적인 제약식을 평가하기 위해 불완전 계산분해과정(incomplete resolution)을 이용한다. 완전한 계산분해과정이 없기 때문이 아니라 이런 과정은 계산적으로 너무 비용이 많이 들기 때문이다. 대신 불완전한 부분을 모든 가능한 해를 조합하여 열거하는 열거법(Enumeration)을 이용해서 해결한다. 이러한 열거는 탐색 가지치기를 통해서 이루어지고 이는 또한 논리 프로그래밍의 비결정주의 연산과 맥이 닿아있다. 계산 분해시 FD 제약식을 평가하는 과정에서 되도록이면 제약식을 거짓으로 만드는 FD 변수값의 조합을 제거하는데 초점이 맞춰지는데 이는 열거를 할 해의 수를 줄이기 위해 필수적인 과정이다. 이 과정이 확산법을 통해서 이루어지는 것은 마치 정수계획법에서 절면을 생성해서 실현 불가능 해를 제거하는 완화법과 같은 역할을 한다. 즉 확산법은 각 선택점에서 한 대안이 선정되면 각 FD 제약식을 통해 사전에 정의된 일관성 기법(consistency technique)에 따라 FD 제약식을 만족시킬 수 없는 변수의 조합을 살펴보고 이에 따라 항상 제약식을 만족시킬 수 없는 값을 도메인에서 제거한다. 만약 한 변수의 도메인에서 모든 값이 제거되어서 공집합을 갖게 되면 선택된 대안으로부터 FALSE 값이 추론된 것이고 선택점으로의 백트랙킹이 일어난다.

혼합형 조건 제약식 (3)에 대한 간단한 구현으로 $h(y)$를 단순히 FD 제약식으로 간주하고 FD 문제해결기 부분에서 $h(y)$의 평가로 TRUE의 값을 갖을 시, 이를 C 인터페이스를 통해 LP 문제해결기에 결과에 해당하는 선형 부등식을 추가하여 선형 부등식을 강제하는 것으로 볼 수 있다. 그러나 이런 구현은 이때의 $h(y)$의 움직임이 다른 FD 제약식과는 다르게 이루어져야 하기 때문에 이 방식은 성립하지 않는다. <표 1>은 이것이 어떻게 다른지를 보여준다.

$h(y)$를 FD 제약식으로 구현을 했을 때, 이는

<표 1> 혼합형 조건식과 FD 제약식의 움직임의 차이

평가 결과	FD 제 약 식	혼합형 조건식의 h(y)
TRUE 값을 가질 때	계산을 계속 수행	선형 부등식을 LP에 강제
FALSE 값을 가질 때	백트랙킹을 수행	계산을 계속 수행
값이 결정 되지 않을 때	계산을 계속 수행	계산을 계속 수행

FD 시스템에서 변수-제약식 네트워크의 한 노드가 되고 변수 도메인의 변화로 세가지 가능한 상태를 만날 수 있다. 즉 TRUE, FALSE 그리고 값이 결정되지 않는 경우들이다. 첫째 경우는 C 인터페이스를 통해서 선형 부등식을 LP에 부가하여 강제하지만, 둘째의 경우 FD 문제해결기에서는 백트랙킹이 일어난다. 하지만 혼합형 조건제약식이 처리되어야 하는 방식은 논리 조건식에서 보듯이 조건이 거짓일 경우 그 결과는 어떤 제약도 주지 않으므로 이 경우 선형 부등식을 무시하고 다음 제약식의 평가로 넘어가는 것이어야 하는데 여기서 불일치가 발생한다. 이 문제를 회피할 한 방법으로 기존의 FD 제약식에서 계산 분해되는 방식이 다른 새로운 확산법을 갖는 새로운 FD 제약식을 만드는 방법이 있을 수 있다. 즉 조건부에 있는 FD 변수가 모두 값이 확정될 때까지 이 제약식의 계산 분해을 미루는 식의 확산법을 갖는 제약식이다. 하지만 이 경우 모든 FD 제약식에 대해 각각 새로운 확산법을 갖는 제약식을 만들어야 한다는 문제가 있다.

가장 합리적인 방법으로 이 논문에서는 이진 FD 변수와 리이피케이션(Reification) 연산자를 이용하는 방법을 제안한다. 식 (3)은 다음과 같은 동일한 두 쌍의 제약식으로 표현된다.

$$h(y) \leftrightarrow B \tag{4}$$

$$B \rightarrow Ax \geq b \tag{5}$$

첫번째 제약식에서 $h(y)$는 이진 FD 변수 B로 재정의되고(reify) 둘째 제약식에서 B의 값에따라 선형 부등식이 강제된다. gprolog에서는 "h(y) #<=> B"와 같이 등치 연산자 "#<=>"를 이용하여 표현된다. 이와 같은 표현의 장점은 혼합형 조건제약식의 조건 부분에 모든 FD 제약식으로 확장될 수 있도록 하는 방법이라는 것이다. 또한 리이피케이션 제약식 (4)는 $h(y)$의 값이 FALSE로 평가되었을 때 $h(y)$는 전체 리이피케이션 제약식의 부분식이기 때문에 시스템이 백트랙킹을 하지는 않는다. 백트랙킹은 $h(y)$의 값이 B 값과 충돌이 일어날 때 전체 제약식의 값이 FALSE가 되므로 발생한다. 그러나 이진 변수 B는 단지 재정의를 위해서 쓰여진 것이라 제약식 (4)와 (5) 이외에는 나타나지 않으므로 미리 값이 확정되는 법이 없고 확산법에 의해 항상 $h(y)$와 동일한 값으로 결정된다. 이 의미는 리이피케이션 제약식의 양부분이 어떤 값을 갖던 간에 리이피케이션 제약식은 항상 TRUE의 값을 갖는다는 것이다. 따라서 이 제약식을 통해서는 백트랙킹이 일어나지 않는다. 제약식 (5)이 혼합형 문제해결기가 구축해야하는 혼합형 이진 조건제약식이다. 따라서 이 제약식에서의 움직임이 앞에서와 같이 FD 제약식의 확산법에서의 움직임과 상충되지 않기 위해 특별한 구현이 필요하다. 그 방법은 FD 문제해결기에서 B의 값이 FALSE가 되더라도 전체 제약식의 값이 TRUE가 되도록 강제로 구현하는 방법에 의해서다. 다음 절에서 이 제약식이 어떻게 구현되는지를 자세히 설명한다.

6.3 혼합형 이진 조건제약식의 구현

제약식 (5)의 구조는 크게 두 부분으로 나뉜다. 첫째 부분은 프롤로그의 서술어 선언부분이다. 이 서술어는 모델링 언어의 한 요소로서 자연스럽게 문제를 정의하는데 사용된다.

```
/* Prolog predicate */
Conditional(B, Inequality) :-
        fd_tell(conditional(B, Inequality)).
```

```
/* C function corresponding to predicate conditional */
Bool conditional(B, Inequality) {
        Return create_conditional(B, Inequality);
}
```

서술어 **fd_tell**은 그 인자로 있는 서술어가 FD 제약식으로 사용된다는 것을 말하고 그 인자 서술어에 상응하는 WAM 수준의 C-함수 **Bool conditional()**을 호출한다. 이 지점에서 시스템은 WAM 수준으로 내려가고 그 명령어들은 다음과 같은 형태이다.

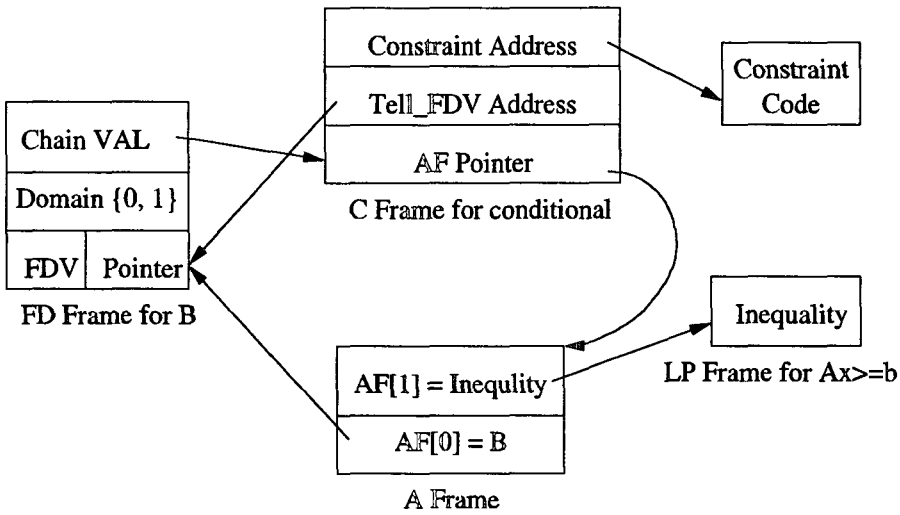
```
Bool create_cond(B,Inequality) {
   LP = Create_LP_Frame(Inequality);
   AF = Create_A_Frame(B, LP);
   CF = Create_C_Frame(AF, exec_cond, B);
   Add_Constraint_Dependency(B, chain_val, CF);
   return exec_cond(B, Inequality);
}

Bool exec_cond(B, Inequality) {
   If (Is_Grounded_Fd_Variable(B)) value_B = Min(B);
   else return TRUE;
   return C_Cond(value_B, Inequality);
}

Bool C_Cond(value_B, Inequality) {
   If (value_B)
       Return Enforce_Linear_System(Inequality);
   Return TRUE;
}
```

이와 같이 제약식의 구현인 함수 **conditional**은 WAM 수준에서 세 단계의 명령어로 구성된다. **Create_cond**는 (5)을 표현하는데 필요한 데이터 구조를 메모리에서 할당하여 만들어낸다. **exec_cond**는 제약식에 대한 확장법이 가동되어 계산분해할 때 호출되는 것으로 계산분해의 조건을 명시한다. **C_Cond**는 혼합형 이진 조건식의 경우 변수 B가 TRUE로 평가되었을 때 시스템이 선형 부등식을 강제하는 함수를 호출하는 것을 보여주는데 일반적으로 **exec_cond**에서 계산분해의 조건을 통과 했을 때 어떤 식으로 계산분해가 이루어지는 지를 정의한다.

[그림 6]은 조건 제약식의 데이터 구조를 나타낸다. 일단 프롤로그로 쓰여진 모델이 **fd_domain_bool(B)**로 이진 FD 변수 B를 선언하면 FD-Frame이 C-STACK에 생성된다. 조건 제약식에 대한 프롤로그의 선언으로 명령어 conditional과 create_cond가 실행된다. 이때 **create_cond**가 선형 부등식에 대한 데이터를 저장할 LP-Frame을 생성하고 서술어 **conditional**의 인자가 되는 정보를 A-Frame에 저장을 한다. 현재 혼합형 이진 조건제약식의 경우 두 개의 인자를 갖게 되어 A-Frame은 크게 두 개의 항목으로 구성되어있다. 다음으로 필요한 작업은 제약식의 구현에 필요한 C-Frame이다. 이는 **Create_C_Frame** 명령어을 통해 이루어지는데 두 번째 인자로 실제 코드 스택에서 확산법이 이 제약식에 대해 이루어질 때 어떤 행위가 뒤따라야 하는 지에 대한 코드의 주소를 전달하는 것으로써 **exec_cond** 명령어가 전달된다.

C-Frame이 만들어지고 다음 단계로 변수-제약식 네트워크를 구성하기 위해 제약식에 대한 객체가 FD 이진 변수에 연결이 되어야 한다. FD 변수의 데이터 구조에서 의존성 사슬 중에 Chain_VAL이라고 불리는 부분에 연결된다. 이것은 **exec_cond**에서 정의된 확산법의 계산분해의 조건과 일치되는 의존성 사슬에 제약식이 연결됨을 보여준다. Add_Constraint_Dependency가 그 명령어이다. 마지막으로 LP-Frame은 선형 부등식에 대한 정보를 포함하는 데이터 구조로 다음과 같이 구성된다.

- Nb_Linear_Cstr: 주어진 혼합형 조건제약식에 의해 강제되어야 하는 선형 부등식의 개수를 나타낸다.



[그림 6] 혼합형 조건제약식의 데이터 구조

- Arry_Operations: 선형부등식의 관계연산자를 나타내는 정수를 포함한 벡터이다.
- Arry_RHS: 선형부등식의 우측상수를 포함하는 벡터이다.
- Arry_Nb_Nonzeros: 각 선형부등식의 계수 중 0이 아닌 계수들의 개수를 나타내는 숫자들의 벡터들이다.
- Arry_Column_Indices: 0이 아닌 계수들에 해당하는 열의 인덱스를 포함하는 벡터이다.
- Arry_Coeff: 전체 선형 부등식의 0이 아닌 계수들을 포함하는 벡터이다.

6.4 백트랙킹

혼합형 문제해결기에서 백트랙킹을 구현하는데 해결되어야 할 이슈들에 대한 논의에 앞서 먼저 FD 문제해결기에서의 백트랙킹을 재검토해 본다. 앞에서 언급한 것처럼 논리 프로그램에서 완전한 계산분해에 대한 연산이 비싸고 언트랙터블(untractable)이라는 사실 때문에 비결정주의 연산을 하게되고 이때 백트랙킹이 발생한다. 논리 프로그램에서의 백트랙킹은 간단하다. 단지 유니피케이션(unification)이라는 연산을 통해서 상태가 변화될 때 변화되기 이전의 상태를 TRAIL에 저장하고 백트랙킹시 이를 팝업(pop-up)하여 상태를 회복시키면 되기 때문이다. 전술한 바와 같이 이는 WAM에서 E-STACK과 TRAIL의 조화로 이루어진다. FD 문제해결기의 경우 논리 프로그래밍의 개념 위에 구현되었기 때문에 백트랙킹을 WAM의 구조에 의존하고 따라서 그 전체적인 윤곽은 동일하다. 모든 FD 변수는 각자의 도메인을 가지고 있고 도메인이 확산법에 의해 축소될 때마다 문제해결기는 축소이전의 상태를 TRAIL에 보전한다. 논리 프로그램과 상화에서 다른 점은 같은 변수에 대한 도메인의 축소가 한 번의 확산법의 실행에서 여러 번 있을 수 있는데 이 현상은 논리 프로그램의 백트랙킹에 수정을 요구한다. 왜냐하면 TRAIL에는 확산법이 적용되기 이전의 오직 하나의 상태만 저

장할 필요가 있기 때문이다. 즉, 시스템은 한 선택점에서 여러 개의 선택 대안을 갖는데 한 선택 대안이 선택된 후 이것이 실패로 기각된다면 선택점으로 돌아가야하고 이 때 새로운 대안을 선택하기 전에 시스템의 상태를 선택 이전의 상태로 회복시켜야 한다. 따라서 이는 선택 이전의 상태를 TRAIL에 저장할 것을 요구한다. 이를 구현하기 위해 스탬핑 방법(stamping scheme)이 적용된다. 즉 변수의 데이터 구조에 그 도메인이 처음 확산법이 적용된 이후로 몇 번의 변화과정을 거쳤는지에 대한 정보를 스탬프로 저장해서 오직 처음 도메인 변화되었을 때만 변화 이전의 상태를 TRAIL에 저장한다.

혼합형 문제해결기는 바로 FD 문제해결기 위에서 LP 문제해결기의 도움으로 이루어진 한 연장으로 볼 수 있기 때문에 백트랙킹도 FD 문제해결기와 동일하다. 즉 혼합형 문제해결기는 FD 제약식을 다룰 수 있어야한다. 이와 함께 혼합형 조건제약식의 요구조건도 충족해야 한다. 필요한 것은 LP 문제해결기의 강제된 선형 부등식들이 저장된 LP 문제의 데이터베이스를 어떻게 WAM의 백트랙킹의 골격에 연결시키는가이다. 다시 선택점에 대한 논의로 돌아가서 한 대안이 선택되고 이에 따라 확산법에 의해 혼합형 조건제약식이 계산분해되면 그 결과로 LP 문제의 데이터베이스에 선형 부등식이 추가된다. 따라서 백트랙킹을 위해서 추가되기 전에 어떤 선형 부등식이 데이터베이스에 있는지에 대한 정보가 TRAIL에 저장되어야 하는 대상이다. 그래서 나타나게 되는 문제는 세가지로 요약된다. 1) 실제 선형부등식은 FD 문제해결기 시스템이 아니라 LP 시스템에 저장이 된다. 또 이 논문에서 혼합형 문제해결기의 구현은 LP를 라이브러리로 사용하기 때문에 LP의 내부 데이터 구조에 접근할 수 없다. 2) TRAIL에 저장해야 하는 정보는 FD 문제해결기에서와 같은 몇 개의 상수가 아니고 LP 시스템에 있는 데이터베이스의 상태이다. 3) 일반적으로 FD 문제해결기의 WAM 구조는 이미 전제되고 있는 데이터 형태만를 추적하도록 설정되어있기 때문에 이와 다른 형태의 데이터를 저장하려면 수정을 요구한다.

이런 문제들을 해결하기 위한 아이디어는 기본적으로 LP 내부의 데이터베이스의 선형부등식의 상태를 모사하는 구조체를 만들어 그 안에 LP 문제의 데이터베이스에 관한 정보를 유지하는 것이다. 그 구조체는 LP-Frame을 확장함으로써 가능하다. LP-Frame에 다음과 같은 정보를 포함하는 필드를 추가한다.

- Enforce_Flag: 이것은 LP-Frame에 있는 선형부등식의 정보가 LP 문제해결기로 강제되었는지를 나타내는 플래그이다.
- Prev_Cstr: 현 LP-Frame에서 표현하는 선형부등식이 LP 문제해결기에 강제되어 있을 때 LP 문제해결기의 문제 데이터베이스에 이 LP-Frame의 선형부등식 보다 앞서서 강제된 선형 부등식들을 표현한 LP-Frame을 지시하는 포인터이다.
- Order: 여기에 저장되는 숫자는 LP-Frame이 몇 번째로 LP 문제해결기에 강제되었는지를 나타낸다.

Prev_Cstr 포인터를 통해서 링크리스트(linked list)가 형성이 되고 이것은 LP 문제해결기의 문제 데이터베이스에 저장되어 있는 선형 부등식의 순서를 모사하고 있다. 즉 확산법에 의해 선형부등식이 LP 데이터베이스에 추가될 때마다 그 순서를 위의 필드에 기록하고 저장한다. 나중에 백

크랙킹에 의해 선형 부등식을 LP 데이터베이스에서 제거할 때 필요한 정보들은 LP-Frame의 위의 필드로부터 얻어지고 LP 문제해결기의 함수들을 통해 LP 데이터베이스에서 제거될 수 있다. 이를 통해 첫 번째와 두 번째의 문제가 해결이 된다. 마지막 문제에 대해서는 함수 어드레스를 WAM에 저장함으로써 해결한다. 즉 혼합형 백트랙킹은 FD에서와 같이 단순한 데이터의 복원 이외에 복잡한 명령들을 수행해야 한다. 이것은 함수를 통해 정의가 되고 이 함수의 어드레스가 WAM의 TRAIL에 저장이 됨으로써 나중에 백트랙킹을 할 때 이 함수를 호출한다. 호출된 이 함수는 LP-Frame을 통해 제공되는 정보를 통해 백트랙킹에 필요한 명령을 수행하게 되는 것이다.

6.5 혼합형 문제해결기의 최적화 계산 모형

FD 문제해결기의 최적화 계산 모형은 WAM 위에서 실패-추동 접근방법(fail-driven approach)에 의해 최적화 문제를 푼다. 그 이유는 프롤로그에는 과정 언어의 반복적인 루프가 구현되어 있지 않고 반복적인 계산을 재귀(recursion)에 의지하고 있기 때문이다. 그러나 계산상의 이유로 반복적인 계산에 재귀는 많은 자원을 소비한다. 이런 이유에서 프롤로그는 비결정주의 연산에서 실패를 통해 다른 대안으로 옮아가는 메커니즘을 이용해 과정 언어의 루프와 같은 연산을 한다. 이 절에서는 혼합형 문제해결기의 최적해 찾는 계산 모형을 제시하기에 앞서 먼저 이해되어야 할 FD의 최적화에서 해를 찾아가는 메커니즘을 설명하기로 한다.

FD 시스템은 기본적으로 어떤 제약식들의 집합이 주어졌을 때 이 모든 제약식을 만족하는 해를 찾는 계산 모형이다. 그래서 최적화 계산 모형은 이 위에 제약식을 추가함으로써 구현된다. 즉 시스템은 제약식을 만족하는 한 해를 발견하면 그리고 비결정주의 연산에 의해 다른 해를 발견할 수 있는 대안이 남아있으면 연산을 그만둘 것인지를 묻게 되는데, 이때의 답변이 fail이면 백트랙킹을 통해 다른 대안을 탐색하게 된다. 이것이 비결정주의 연산에 의한 실패-추동 접근방법이고 최적화 모형은 목적함수 값을 나타내는 변수를 도입, 해를 찾을 때마다 이에 대한 한계를 제약식으로 부가함으로써 기존의 해가 갖고 있는 목적함수 값보다 더 향상된 값을 갖는 해을 도출해 낸다. 다음은 gprolog에 구현된 FD 최적화 계산 모형이다.

```
fd_minimize(Goal, Var) :-
    fd_max_integer(Inf),
    g_assign('$cur_min', Inf),
    repeat,
    g_read('$cur_min', B),
    B1 is B-1,
    (   fd_domain(Var, 0, B1),
        '$call'(Goal, fd_minimize, 2, true) ->
        fd_min(Var, C),
        g_assign('$cur_min', C),
        fail,
    ;   !,
        Var = B,
        '$call'(Goal, fd_minimize, 2, true)
    ).
```

변수 Goal은 FD의 제약식 만족문제이다. 변수 Var는 최적화 목적함수 값을 나타내는 변수이다. FD 제약식 만족문제의 예로 다음과 같이 표현된다.

```
my_goal :-
    fd_domain([Y1,Y2],2,3),
    fd_domain(Y3,1,10),
    Y1 + Y2 #=< Y3,
    fd_labeling([Y3,Y2,Y1]).
```

이때 최소화 문제는 서술어 **fd_minimize (my_goal,Y3)**과 같다. 즉 **my_goal**을 만족하며 **Y3**의 값을 최소화하는 해를 찾는 것이다. FD의 계산 모형에서 **'$cur_min'**은 현재까지의 목적함수식을 저장하는 전역변수이다. 이 값보다 1이 작은 값으로 **B1**이 설정되고 서술어 **fd_domain (Var,0,B1)**을 통해 목적함수 값의 상계를 설정하게 된다. 물론 이때 설정되었다는 말은 과정 언어의 변수에 그 값이 계산되어 저장되었다는 뜻과는 다르다. 유니피케이션(unification)에 의해 일단 **'$cur_min'** 전역변수의 값이 달라지면 자동적으로 **B1**이 그 값보다 1이 작은 값으로 바뀐다. 또 **fd_domain(Var,0,B1)**은 사실상 **Var**라는 변수가 **B1**의 값에 의해 변한다는 제약식을 FD 제약식 문제의 변수-제약식 네트워크에 추가하는 역할을 하기 때문에 **B1** 값의 변화에 따라 확산법에 의해 **Var**의 도메인을 한정한다. 실패-추동 접근방법은 **repeat** 서술어에 의해 선언되고 있다. **repeat** 이후에 **fail**을 만날 때까지의 과정에서 비결정주의 연산에 의해 여러 선택점이 만들어지게 되는데 전술한 바와 같이 **fail**을 만날 때마다 다음의 대안으로 계산이 옮겨간다. 그 대안은 FD 제약식 문제로 주어지는 **Goal**의 서술어 **fd_labeling**에 의해 만들어진다. 우리의 예인 **my_goal**에서 볼 수 있는 것처럼 **fd_labeling ([Y3,Y2,Y1])**은 리스트의 끝 **Y1**부터 **Y3**까지 그 도메인에 있는 값 각각마다 대안을 노드로 하는 탐색나무를 만든다. 그리고 **fail**을 만날 때마다 백트랙킹을 통해서 **Var**의 도메인의 값이 바뀌면서 다른 대안으로 계산이 옮아간다.

혼합형 최적화 문제의 계산 모형을 만들기 위해 이 FD 최적화의 계산 모형을 수정하게 하는 두 가지 문제가 존재한다. (1) 위의 **fd_minimize**는 사실상 문제를 두 번에 걸쳐 풀고 있다. 첫째로 최적의 목적함수 값을 구하기 위해 실패-추동 접근방법에 의해, 둘째 모든 대안이 다 소진된 이후 변수 **Var**를 변수 **B**, 즉 찾아진 최적의 목적함수 값에 같게 놓은 후로 제약식을 만족하는 해를 찾는다. 이는 계산 모형의 연산자 ; 이후에 나타나는 서술어들로부터 알 수 있다. 이것이 가능한 것은 **fd_labeling**에서 각 변수의 값이 확정되면서 일어나는 확산법의 계산량이 상대적으로 작기 때문에 두 번의 연산이 큰 부담이 되지 않는다. 또한 **Var=B**라는 서술어을 통해 대안의 대부분이 FALSE로 제거되어 나가기 때문에 계산량이 작다. 반면 혼합형 모형에 따른 문제해결기의 계산에서는 목적함수의 값이 LP 부분에서 LP 문제해결기에 의해 계산이 되기 때문에 이는 직접적으로 FD 부분의 변수-제약식 네트워크에 영향을 미치지 못해 결국은 확산법에 의한 계산량을 줄여줌이 미미하다. (2) FD 문제해결기의 계산 모형은 확산법과 비결정주의 연산을 위해 선택점을 생성하는 열거 서술어 **fd_labeling**을 밀접하게 결합시키고 있다. 즉 **my_goal**처럼 문제를 설정하는 모형 언어로부터 변수-제약식 네트워크가 메모리에 설정되고 열거 서술어에서 변수에 대한 한 값이 고정될 때마다 확산법은 그 네트워트 위에서 변수의 도메인들을 줄여나가며 결국은 각 변수의 해를 찾아낸다. 즉 선택점에서 대안의 선택은 확산법의 시동으로 직접 연결되어 도메인의 축소까지 한 스텝에 이루어진다. 혼합형 문제해결기의 경우, 확산법에 의한 도메인 축소 후에 LP 문제를 푸는 또 한 단계의 계산을 필요로 하므로 기존의 FD에 설정된 **fd_labeling**을 직접 이용해서는 이루어질 수 없고 혼합형 문제해결기 나름의 열거 서술어가 필요하게 한다. 이를 해결하기 위해 제안되는 계산 모형은 프롤로그로 다음과

같이 표현된다.

```
hyb_labeling([]) :-
hyb_labeling([X|Ys]) :-
    fd_labeling(X),
    hyb_solve_lp(Z0) ->
    hyb_update(Z0),
    hyb_labeling(Ys).
hyb_minimize(Goal, Var) :-
    repeat,
    hyb_init_lp,
    ( '$call'(Goal, hyb_minimize, 2, true),
      fail,
    ; !,
      hyb_finish_lp,
      hyb_print_solution(Var)
    ).
```

여기서 보는 바와 같이 **hyb_labeling**은 재귀식으로 이루어져있고 LP 문제 풀이가 포함되어 있다. 이 계산 모형에서 첫째 문제에 대한 해법으로는 일반적인 과정 언어에서와 같이 확산법과 LP 문제풀이마다 그 해와 목적함수 값을 메모리에 저장하고 향상된 해가 구해질 때마다 이를 업데이트해 준다. 서술어 **hyb_update**는 이를 보여준다. 그리고 **hyb_minimize**에서 실패-추동 접근방법에 의해 모든 대안의 탐색이 끝나면 **hyb_print_solution** 서술어을 통해 해를 사용자에게 보여준다. 두번째 문제는 전술한바와 같이 **hyb_labeling**이라는 혼합형 문제해결기 나름의 열거법을 정의함으로써 해결된다.

7. 결론과 앞으로의 연구 방향

이 논문에서는 혼합형 방법의 가장 기본적이지만 일반적인 추론의 형태를 담고 있는 제약식인 조건제약식을 근간으로 WAM, FD 그리고 LP 문제해결기의 구조가 어떻게 결합할 수 있으며 이 때 발생하는 문제들을 살펴보고 이에 대한 해결책을 제시하였다. 앞으로 이루어질 수 있는 연구의 방향을 살펴보면 다음과 같다. 첫째로 여기서 논의된 것은 가장 기본적인 형태의 제약식 만을 고려하였으므로 논의된 문제해결기는 실제 문제를 풀기에는 불완전한 형태이다. 현실 문제를 풀기 위해서는 특히 혼합형 방법의 모델 언어에서 사용되는 여러 제약식들의 구현이 따라야한다. 그 예를 들어보면 혼합형 element, piecewise_linear 같은 것들이 있다. 둘째로 제약식의 형태외에 제약만족 이론에서 제기되는 다양한 형태의 탐색기법들이 혼합형 문제해결기 시스템에서 구현 가능하다. 특히 기반이 되는 WAM 구조는 탐색의 기본인 백트랙킹을 실현하는 스택으로 이루어져 정보의 생성과 저장의 순위를 바꿈으로써 다양한 탐색기법을 구현하기에 적당한 구조이다. 셋째는 좀더 일반적으로 여기서는 제시한 FD와 LP 문제해결기의 결합이 아닌 다양한 형태의 CP와 최적화 문제해결기의 결합을 모색해 볼 수 있다.

참고문헌

김학진, "제약식 프로그래밍과 최적화 솔버로부터 혼합솔버의 구현에 대하여" Information Systems Review, Vol.5, No.2(2003), 203-217.

Ait-Kaci, Hassan, "Warren's Abstract Machine: A Tutorial Reconstruction", Intelligent Software Group (Simon Fraser University), 1999, http://www.isg.sfu.ca/~hak

Beringer, H. and B. De Backer, "Combinatorial problem solving in constraint logic programming with cooperating solvers", Logic Programming: Formal Methods and Practical Applications (C. Beierle and L. Plumer eds.), Studies in Computer Science and Artificial Intelligence, Elsevier, 1995.

Bockmayr, A. and T. Kasper, "Branch-and-infer: A unifying framework for integer and finite domain constraint programming", INFORMS J. Computing, Vol.10, No.3(1998), 287-300.

Carlson, B., M. Carlsson and D. Diaz, "Entailment of Finite Domain Constraints", International Conference on Logic Programming (ICLP), Santa Margherita, Italy, 1994.

Codognet, P. and D. Diaz, "wamcc: Compiling Prolog to C", International Conference on Logic Programming (ICLP), Tokyo, Japan, 1995.

Codognet P. and D. Diaz, "Compiling Constraints in clp(FD)", Journal of Logic Programming, Vol. 27, No. 3(1996), 1-47.

Codognet, P. and D. Diaz, "Design. and Implementation of the GNU Prolog System", Journal of Functional and Logic Programming, Vol. 2001, No. 6(2001).

Darby-Dowman, K. and J. Little, "Properties of some combinatorial optimization problems and their effect on the performance of integer programming and constraint logic programming", INFORMS J. Computing, Vol.10, No.3(1998), 276-286.

De Backer, B. and H. Beringer, "Intelligent backtracking for CLP languages: an application to CLP(R)", Logic Programming, Proceedings of the 1991 International Symposium (V. Saraswat and K. Ueda eds.), The MIT Press, San Diego, 1991, 405-419.

Diaz, Diniel, "GNU Prolog: A Native Prolog Compiler with Constraint Solving over Finite Domains", September, 2002, http://pauillac.inria.fr/~diaz/gnu-prolog.

Diaz, D. and P. Codognet, "The GNU Prolog System and its Implementation", ACM Symposium on Applied Computing (SAC), Villa Olmo, Como, Italy, 2000.

Diaz, D. and P. Codognet, "GNU Prolog: Beyond Compiling Prolog to C", Practical Aspects of Declarative Languages (PADL), Boston, 2000.

Dincbas, M., P. Van Hentenryck, H. Simonis, A. Aggoun and F. Berthier, "The Constraint Logic Programming Language CHIP", Proceedings of International Conference on Fifth Generation Computer Systems (FGCS-88), Tokyo, 1988, 693-702.

Hajian, M., R. Rodosek and B. Richards, "Introduction of a new class of variables to discrete and integer programming problems", Baltzer Journals, 1996.

Hajian, M., "Dis-equality constraint in linear/integer programming", Technical Report, IC-Parc, 1996.

Hooker, J. and M. Osorio, "Mixed Logical/Linear Programming," Discrete Applied Mathematics, Vol.96-97 (1999), 395-442.

Hooker, J., Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction, Wiley, New York, 2000.

Hooker, J., Hak-Jin Kim and Greger Ottoson, "A Declarative Modeling Framework that Integrates Solution Methods", Annals of Operations Research, Vol.104 (2001), 141-161.

Kim, Hak-Jin and J. Hooker, "Solving fixed-charge network flow problems with a hybrid optimization and constraint programming approach," Annals of Operations Research, Vol.115 (2002), 95-124.

Marriott, K. and P. Stuckey, Programming with Constraints: An Introduction, MIT Press, 1998.

Milano, Michela(ed.), Constraint and Integer Programming: Toward a Unified Methodology, Kluwer Academic Publishers, 2004.

Rodosek, R., M. Wallace and M. Hajian, "A new approach to integrating mixed integer programming and constraint logic programming", Baltzer Journals, 1997.

Tsang, E., Foundations of Constraint Satisfaction, Academic Press, 1993.

Abstract

On an Implementation of a Hybrid Solver Based on Warren Abstract Machine and Finite Domain Constraint Programming Solver Structures

Hak-Jin Kim*

Constraint Programming in AI and Optimization in OR started and have grown in different backgrounds to solve common decision-making problems in real world. This paper tries to integrate results from those different fields by suggesting a hybrid solver as an integration framework. Starting with an integrating modeling language, a way to implement a hybrid solver will be discussed using Warren's abstract machine and an finite domain constraint programming solver structures. This paper will also propose some issues rising when implementing the hybrid solver and provide their solutions.

Key words : 제약만족화, 제약식 프로그래밍, 최적화, 정수계획법, 선형계획법, 솔버 기술, WAM, gprolog, 소프트웨어

* Business Administration, Yonsei University