퍼지 및 지능시스템학회 논문지 2004, Vol. 14, No. 2, pp. 156-163

유전알고리즘 기반의 멀티에이전트 시스템 조정 협동 기법

A Coordinated Collaboration Method of Multiagent Systems based on Genetic Algorithms

손봉기, 이건명
Bong-Ki Sohn, Keon-Myung Lee

충북대학교 전기전자컴퓨터공학부, 컴퓨터정보통신연구소, 첨단정보기술연구센터(AITrc)
School of Electric and Computer Engineering, Research Institute for Information Communication, Chungbuk National University and AITrc

요 약

이 논문에서는 다수의 에이전트, 외부 자원, 처리 작업이 존재하는 멀티에이전트 시스템 환경에서의 작업 처리를 위한 조정 협동 기법에 대해 기술한다. 에이전트는 특정 작업들을 실행할 수 있고, 외부 자원은 하나의 에이전트에 의해 사용되거나 여러 에이전트에 의해 공유될 수 있다. 작업은 에이전트에 의해 실행될 수 있는 여러 개의 부작업들로 구성된다. 여러 에이전트가 동일한 부작업을 실행할 수 있지만, 어떤 에이전트가 수행하느냐에 따라 비용이 달라질 수 있다. 따라서, 효율적인 작업 처리를 위해서는, 부작업을 적절한 에이전트에 할당하는 것과 각 에이전트에 할당된 부작업들에 대한 실행 순서를 정하는 스케쥴링과 같은 에이전트 조정 작업이 필요하다. 이 논문에서는 이러한 멀티에이전트 시스템 환경에서의 작업 처리를 위한 유전알고리즘(genetic algorithm) 기반의 조정 협동 기법을 제안한다. 또한, 실험을 통해 제안한 기법이 유용한 멀티에이전트 시스템 조정 협동 방법임을 보인다.

Abstract

This paper is concerned with coordinated collaboration of multiagent system in which there exist multiple agents which have their own set of skills to perform some tasks, multiple external resources which can be either used exclusively by an agent or shared by the specified number of agents at a time, and a set of tasks which consists of a collection of subtasks each of which can be carried out by an agent. Even though a subtask can be carried out by several agents, its processing cost may be different depending on which agent performs it. To process tasks, some coordination work is required such as allocating their constituent subtasks among competent agents and scheduling the allocated subtasks to determine their processing order at each agent. This paper proposes a genetic algorithm-based method to coordinate the agents to process tasks in the considered multiagent environments. It also presents some experiment results for the proposed method and shows that the proposed method is a useful coordination collaboration method of multiagent system.

Key words : 멀티에이전트시스템, 조정 협동, 유전알고리즘, 작업 할당, 작업 스케쥴링

1. 서 론

분산 정보 시스템(distributed information systems)의 정보 처리 작업은 분산된 서버들의 연속된 연산에 의해 이루어진다. 이러한 연산에는 파일, 통신 대역폭, 메모리 등과 같은 외부 자원이 필요하다. 또한, 산업 공정 시스템(industrial manufacturing system)의 작업 처리에도 다수의 워크스테이션(workstation)과 외부 자원이 필요하다. 워크스테이션은 외부 자원을 사용하여 연속적으로 작업 처리를 수행함으로써 제품을 생산한다[1]. 분산 정보 시스템과 산업 공정 시스템은 워크스테이션이나 서버에 해당하는 에이전트로 구성된 멀티에이전트 시스템으로 볼 수 있다. 즉, 에이전트는 외부 자원을 필요로 하고, 특정 작업을 실행할 수 있는 능력이 있다. 또한 에이전트는 다른 에이전트의 도움없이 한번에 하나의 부작업만 실행할 수 있다. 파일, 통신 대역폭, 주변장치와 같은 외부 자원은 한번에 하나의 에이전트에 의해 독점적으로 사용되거나 다수의 에이전트에 의해 공유될 수 있다. 작업은 부분적으로 순서화된 부작업들로 구성되는데, 부작업 간의 순서가 위상적인(topological) 부분순서를 위배하지 않는다면 작업은 어떤 순서로도 처리될 수 있다. 즉, 작업은 부분적으로 순서화된 부작업들로 구성된 미리 결정된 계획이라 볼 수 있다.

이 논문에서는 위에서 언급한 멀티에이전트 시스템 환경에서 협동하는 다수의 에이전트가 효율적으로 작업을 처리하기 위한 멀티에이전트 조정 협동(coordinated collaboration) 기법에 대해 다룬다. 조정 작업은 에이전트에 부작업을 분산

접수일자 : 2004년 1월 26일
완료일자 : 2004년 3월 28일
본 연구는 첨단정보연구센터(AITrc)를 통해서 과학재단 지원으로 수행된 것임.

시키는 작업 할당(task allocation)과 각 에이전트에 할당된 부작업의 실행 순서를 정하는 작업 스케쥴링(task scheduling) 문제로 나눌 수 있다. 두 가지 문제는 다항 함수 시간(polynomial time)에 최적해를 구할 수 없는 NP-complete로 알려져 있다.

이 논문에서는 멀티에이전트 환경에서의 에이전트 협동을 위한 유전알고리즘(genetic algorithm) 기반의 에이전트 조정 기법을 제안한다. 이 기법은 중개 에이전트(broker agent)가 전체 시스템의 작업 할당과 스케쥴링을 담당하는 중앙 조정(centralized coordination) 방법이다. 중개 에이전트는 2-계층 유전알고리즘(two-layer genetic algorithm)에 의한 결과에 따라 에이전트 조정을 수행한다. 2-계층 유전알고리즘은 작업 할당을 위한 상위 계층 유전알고리즘(upper layer genetic algorithm)과 상위 계층 유전알고리즘에 의해 제시된 작업 할당에 대하여 효율적인 작업 스케쥴을 생성하는 하위 계층 유전알고리즘(lower layer genetic algorithm)으로 구성된다.

이 논문의 구성은 다음과 같다. 2장에서는 멀티에이전트 시스템에서의 조정 협동 기법에 대해 살펴보고, 3장에서는 유전알고리즘 기반의 조정 협동 기법을 제안한다. 4장에서 제안한 기법의 실험 결과를 보이고, 5장에서 결론을 맺는다.

2. 에이전트 조정 협동 기법

멀티에이전트 시스템 내의 에이전트에 작업을 분산시키는 작업 할당은 멀티에이전트 조정의 중요한 부분이다. 작업 할당 방법에 대한 연구는 활발히 이루어져 왔는데, 크게 중앙 할당(centralized allocation)과 분산 할당(distributed allocation) 방법으로 분류할 수 있다[2,3].

중앙 할당에서는 두 가지 접근법이 있다. 첫 번째는 멀티에이전트 시스템의 구조가 계층적 포함 구조(hierarchical subordination organization)를 이루고 있어, 상위 계층의 에이전트가 하위 계층의 에이전트에게 작업을 할당할 수 있다[2]. 계층적 포함 구조는 작업 할당 방법이 시스템 구조에 내포되어 있어, 작업 할당의 유연성이 떨어질 수 있다. 두 번째는 모든 서비스에 대한 클라이언트의 요청과 서버의 입찰을 받아 비교하여 일치하는 서버로 클라이언트의 요청을 전송하는 중개 에이전트(broker agent)를 이용하는 작업 할당이다.

분산 할당은 에이전트가 필요한 서비스를 다른 에이전트에게 직접 요청하는 방법으로, 획득망(acquaintance network)과 계약망(contract net)에 의한 방법이 있다[2,3]. 획득망에 의한 방법에서는 각 에이전트가 자신이 알고 있는 에이전트에 대한 기능 테이블(skill table)을 가지고 있다고 가정한다. 다른 에이전트에 의해 처리되어야 할 작업이 있을 때, 에이전트는 기능 테이블을 이용해 순서대로 그 작업을 처리할 수 있는 에이전트에게 작업 처리 요청을 한다. 계약망에 의한 작업 할당은 다른 에이전트의 서비스를 필요로 하는 에이전트는 입찰 관리자(bidding manager)가 되고, 서비스를 제공할 수 있는 에이전트는 입찰자가 되어, 시장에서의 계약 절차와 같은 프로토콜에 의해 작업 할당이 이루어진다. 계약망에 의한 작업 할당은 다음과 같은 과정을 거쳐 이루어진다: 첫 번째 단계에서 입찰 관리자는 처리할 작업에 대한 정보를 미리 정해진 에이전트나 시스템의 모든 에이전트로 전송한다. 두 번째 단계에서 입찰자는 작업 정보에 기초하여 입찰 여부를 결정하고, 입찰할 수 있다면 입찰 관리자에게 입찰한다. 세 번째 단계에서는 입찰 관리자가 입찰자의 입찰을 받아들이고 평가하여 가장 좋은 입찰자와 계약을 맺는다. 마지막으로 계약을 맺게된 입찰자는 작업 실행 메시지를 입찰 관리자에 전송한다. 분산 할당은 작업부하, 자원의 가용성 등에 대한 현재 상태를 고려하여 어느 에이전트에게 작업을 할당할 것인가가 결정되는 장점이 있다. 그러나 중앙 할당과 비교할 때, 전체 시스템 성능을 고려하여 작업 할당하는 데는 한계가 있다.

부작업이 에이전트에 할당되면, 전체 시스템 성능을 최대화하도록 각 에이전트에 할당된 부작업의 실행 순서를 결정해야 한다. 특히, 에이전트가 제한적인 자원에 대해 경쟁하는 경우에는 더욱 신중한 작업 스케쥴링이 필요하다.

이 논문에서는 부작업의 실행 순서와 부작업 간의 내포된 선행 관계를 나타내는 멀티에이전트 시스템의 작업 처리 계획(plan)은 미리 정해져 있고, 부작업에 대한 작업 할당과 각 에이전트에서의 작업 스케쥴은 정해져 있지 않다고 가정한다. 작업의 실행 순서를 정하는 계획기법(planning)은 멀티에이전트 시스템에서 중요한 부분으로, 크게 중앙 계획기법, 부분 계획(partial plan)에 대한 중앙 조정 계획기법과 분산 계획기법이 있다[2]. 중앙 계획기법은 계획 에이전트(planner agent)가 모든 에이전트에 대한 전체적인 계획을 생성하고, 작업을 정적 또는 동적인 방법으로 에이전트에 할당한다. 동적 할당은 계약망이나 획득망에 의해 이루어지고, 정적 할당은 미리 정해진 에이전트의 기능에 따라 할당된다. 부분 계획에 대한 중앙 조정 계획기법은 각 에이전트가 자신의 목적 달성을 위한 부분 계획을 생성하여 조정자(coordinator)에게 전송한다. 조정자는 순서대로 작업을 정렬하거나, 에이전트 간 충돌을 피하기 위한 동기점(synchronization point)을 설정함으로써 모든 부분 계획을 하나의 전체적인 계획으로 통합한다. 분산 계획기법은 각 에이전트가 독립적으로 자신의 부분 계획을 생성하고, 다른 에이전트의 부분 계획과 목적에 관련된 정보만을 서로 교환한다. 교환된 정보를 이용하여 각 에이전트는 다른 에이전트의 부분 계획과 양립할 수 있도록 자신의 부분 계획을 수정한다. 부분 계획에 대한 분산된 조정 방법인 분산 계획기법에 대한 많은 연구가 진행되고 있다[4-6].

작업 할당과 스케쥴링 문제는 최적해를 보장하는 방법이 없는 NP-complete 문제로 알려져 있다[7]. 이 논문에서 고려하는 멀티에이전트 시스템은 작업을 처리 시 반응적으로 행동하지 않고, 충분한 시간을 가지고 행동한다고 가정한다. 즉, 효율적인 작업 할당과 스케쥴을 찾는데 충분한 시간이 있다는 것을 의미한다. 또한, 처리할 작업은 일괄적으로 미리 주어진다고 가정한다. 처리할 작업이 점진적으로 주어지는 멀티에이전트 시스템 환경에 제안한 에이전트 조정 기법을 적용하는 것에 대해서는 논문 후반부에서 다룬다. 이 논문에서는 에이전트들의 작업 처리를 조정하는 유전알고리즘 기반의 조정 협동 기법을 제안한다. 제안한 기법은 중개 에이전트가 에이전트와 작업 및 자원에 대한 모든 정보를 관리하고 감시하며, 유전알고리즘을 이용하여 조정된 작업 할당과 스케쥴을 생성하는 중앙 조정 기법이다. 유전알고리즘 기반의 접근법은 여러 가지 장점을 가진다. 첫째, 유전알고리즘은 항상 최적해는 아니더라도 탐색 과정을 거쳐 가능한 해를 찾을 수 있다. 둘째, 다른 방법에 의해 얻은 해가 유전알고리즘의 염색체(chromosome)로 표현되어 사용될 수 있다. 셋째, NP-complete 문제와 같은 많은 최적화 문제에서 유전알고리즘이 효율적인 해를 찾을 수 있다는 것이 실험적으로 알려져 있다[8].

이 논문에서의 작업 스케쥴링은 워크스테이션(에이전트)이 작업의 어떤 연산을 해야할 지를 미리 알고 있는 Job

Shop Scheduling 문제와 매우 유사하다. 스케쥴링 문제는 운용 과학(operation research)과 인공지능 분야에서, 우선순위 규칙(priority rule), Giffler-Thompson 규칙과 같은 휴리스틱(heuristics) 이나 유전 알고리즘, 타부 탐색, simulated annealing 등과 같은 메타 휴리스틱을 이용한 많은 연구가 이루어졌다[8-12]. 이 논문에서 고려하는 작업 스케쥴링 문제는 Job Shop Scheduling 문제와 유사지만, Job Shop Scheduling 문제가 외부 자원을 고려하지 않기 때문에 같은 것으로 볼 수 없다. 계획기법과 스케쥴링 문제에서 자원 제약조건이 있는 스케쥴링 문제에 대한 연구는 있지만 작업 할당 문제에서의 자원 제약은 다루지 않고 있다[13].

3. 유전알고리즘 기반의 멀티에이전트 시스템 조정 협동 기법

3.1. 작업 할당과 스케쥴링

멀티에이전트 시스템의 조정 작업은 부작업을 적절한 에이전트에 할당하는 것과 전체 시스템 성능을 최대화하는 각 에이전트에서의 부작업 스케쥴링 문제로 나눌 수 있다. 제안한 기법은 상위 계층 유전알고리즘이 후보 작업 할당을 생성하고, 하위계층 유전알고리즘이 상위계층 유전알고리즘에 의해 제시된 작업 할당에 대하여 효율적인 스케쥴을 생성하는 2-계층 유전알고리즘을 이용해 조정 작업을 수행한다. 여러 에이전트가 다른 비용으로 같은 부작업을 처리할 수 있기 때문에, 부작업 할당은 작업 부하, 외부 자원, 처리량 등을 고려하여 가장 적절한 에이전트에 할당되어야 한다. 외부 자원은 한번에 하나의 에이전트에 의해서만 사용되거나 여러 에이전트에 의해 공유될 수 있고, 부작업 실행에 여러 자원이 동시에 필요할 수 있다. 따라서, 부작업 간의 교착상태(deadlock)를 회피하도록 작업 스케쥴링이 이루어져야 한다[14].

제안한 기법의 기술을 위해 다음과 같은 표기법을 사용한다:

- $A=\{A_1, A_2, \ldots, A_m\}$: 에이전트 집합에 해당하며, A_i는 i번째 에이전트를 의미한다.
- $T=\{T_1, T_2, \ldots, T_n\}$: 작업 집합에 해당하며, T_i는 i번째 작업을, t_{ij}는 작업 T_i의 j번째 부작업을 의미한다.
- $PO_i=\{(t_{ij}, t_{ik}) | t_{ij}$는 t_{ik} 보다 이전에 실행되어야 한다$\}$: T_i의 부작업 간의 선행관계를 나타낸다.
- $R=\{R_1, R_2, \ldots, R_l\}$: 자원 집합에 해당하며 R_i는 i번째 자원을 의미한다.
- $C(R_i)$: R_i를 동시에 공유할 수 있는 에이전트 수를 나타낸다.
- $O=\{O_1, O_2, \ldots, O_l\}$: 전제시스템에서 실행되는 부작업 집합에 해당한다.
- $PT_i(O_j)$: A_i가 O_j를 실행하는데 소요되는 시간에 해당한다.
- $CA_i(O_i)$: O_i를 실행할 수 있는 에이전트 집합에 해당한다.
- $PR_i(O_j)$: T_i의 PO_i에서 O_j에 선행하는 부작업 집합에 해당한다.
- $N=\sum_{i=1}^{n} n_i$: 모든 작업의 부작업의 합을 나타내며 n_i는 T_i의 부작업 수를 의미한다.

조정 과정에 적용되는 목적함수(objective function)는 응용분야에 따라 다양하게 정의될 수 있다. 예를 들어, 작업 종료 시간의 합이나 최종 작업의 종료 시간이 조정 작업의 목적함수가 될 수 있다. 조정 작업은 적용된 목적함수를 최적화하도록 작업 할당과 스케쥴 생성 수행한다고 가정한다. 조정 과정을 거치면 작업과 에이전트 및 자원에 대한 스케쥴을 얻는다. 각 작업 T_i에 대한 스케쥴은 $(O_j, A_k, [t_l, t_m])$의 집합으로 표현되는데, 에이전트 A_k가 $[t_l, t_m]$동안 작업 T_i의 부작업 O_j를 실행한다는 것을 의미한다. 에이전트 A_i의 스케쥴은 $(T_j, O_k, [t_l, t_m])$의 집합으로 표현되는데, 에이전트 A_i는 작업 T_j의 부작업 O_k를 $[t_l, t_m]$동안 실행한다는 것을 의미한다. 자원 R_i의 사용 스케쥴은 $(A_j, T_k, O_l, [t_m, t_n])$의 집합으로 표현되는데, 에이전트 A_j가 $[t_m, t_n]$동안 T_k의 부작업 O_l을 실행하기 위해 R_i를 사용한다는 것을 의미한다.

목적함수 계산을 용이하게 하기 위해 에이전트 간의 작업 전송 시간은 무시할 수 있다고 가정한다. 작업은 작업 스케쥴에 따라 해당 에이전트에게 전송된다. 따라서, 에이전트 A_i가 할당된 T_j의 부작업을 실행한 후에, 작업 스케쥴에 따라 T_j를 처리하기 위해 다음 에이전트로 작업 T_j를 전송한다.

3.2. 멀티에이전트 시스템 조정 협동을 위한 유전알고리즘

제안한 조정 협동 기법은 상위계층 유전알고리즘에 의해 작업 할당이 제시되고, 하위계층 유전알고리즘이 제시된 작업 할당에 대하여 작업 스케쥴을 생성하는 2-계층 유전알고리즘에 의해 구현된다. 다음은 유전알고리즘 기반의 조정 협동 기법의 전체 프로시져이다.

procedure *Upper-Layer-GA-for-Task-Allocation*

단계 1. 가능한 작업 할당으로 이루어진 해집단을 초기화한다.
단계 2. Lower-Layer-GA-for-Task-Scheduling에 의해 각 후보 작업 할당의 적합도를 평가한다.
단계 3. 복제.교차.변이 유전연산자로 다음 세대의 해집단을 생성한다.
단계 4. Lower-Layer-GA-for-Task-Scheduling에 의해 각 후보 작업 할당의 적합도를 평가한다.
단계 5. 종료 조건과 일치하면, 가장 좋은 성능의 작업 할당과 에이전트.작업.자원에 대한 스케쥴을 반환하고, 그렇지 않으면 단계 3으로 간다.

procedure *Lower-Layer-GA-for-Task-Scheduling*

단계 1. Upper-Layer-GA-for-Task-Allocation에 의해 제시된 작업 할당과 자원을 고려하여 가능한 스케쥴로 이루어진 해집단을 초기화한다.
단계 2. 각 후보 스케쥴의 적합도를 평가한다.
단계 3. 복제.교차.변이 유전연산자를 이용하여 다음 세대의 해집단을 생성한다.
단계 4. 각 후보 스케쥴의 적합도를 평가한다.
단계 5. 미리 정해진 세대 수를 반복했다면, 가장 좋은 성능의 스케쥴을 반환하고, 그렇지 않으면 단계3으로 간다.

3.3. 작업 할당을 위한 상위계층 유전알고리즘

유전알고리즘을 구현하기 위해서는 후보해 인코딩 스키마, 해집단 초기화, 유전 연산자, 적합도 평가함수와 같은 기본적인 구성 요소를 정의해야 한다. 이 절에서는 작업 할당을 위한 상위계층 유전알고리즘의 구성 요소의 표현 방법에 대해

기술한다.

인코딩 스키마(Encoding Scheme)

각 부작업이 에이전트에 할당된 상태를 후보해로 표현한다. 후보해는 크기 N의 배열로 표현되는데, 각 배열 요소는 부작업 t_{ij}에 해당하고 값은 부작업을 실행할 수 있는 에이전트 집합 $CA(t_{ij})$ 중의 한 에이전트 식별자를 의미한다. 그림 1은 작업 할당에 대한 후보해를 표현한 예로서 부작업 t_{12}가 에이전트 A_2에 할당되었다는 것을 의미한다.

t_{11}	t_{12}	t_{13}	t_{21}	t_{22}	t_{31}	t_{32}
1	2	1	3	2	1	2

그림 1. 후보 작업 할당 인코딩

해집단 초기화(Population Initialization)

초기 해집단의 후보해는 각 부작업 t_{ij}을 실행할 수 있는 에이전트 집합 $CA(t_{ij})$에서 하나를 임으로 선택하여 생성한다.

유전 연산자(Genetic Operators)

복제(reproduction) 연산자는 하나의 후보해를 다음 세대로 복사하는 기본적인 연산자이다. 교차(crossover) 연산자는 이전 세대의 후보해 P_1, P_2로부터 새로운 두 후보해 C_1, C_2를 생성한다. 새로운 후보해를 생성하기 위해 0과 1로 임의로 초기화된 크기 N의 MASK 배열을 만든다. C_1은 MASK[i]=0이면 $P_1[i]$를 $C_1[i]$로 복사하고, 그렇지 않으면 $P_2[i]$를 $C_1[i]$로 복사한다. 다른 후보해 C_2도 유사한 방법으로 생성한다. 그림 2는 P_1과 P_2에 대해 교차 연산자를 적용하여 후보 작업 할당 C_1을 생성한 것이다.

	t_{11}	t_{12}	t_{13}	t_{21}	t_{22}	t_{31}	t_{32}
MASK	0	0	1	0	0	1	0
P_1	1	2	3	3	1	2	2
P_2	2	1	1	3	2	1	1
C_1	1	2	1	3	1	1	2

그림 2. 교차연산의 적용

변이(mutation) 연산자는 이전 세대의 후보해 P로부터 새로운 후보해 C를 생성하는 연산자다. 먼저 P를 C로 복사한 후, 변이 확률에 의해 i번째 요소를 선택하여 해당 부작업을 실행할 수 있는 에이전트 중 임의로 하나의 에이전트를 선택하여 그 식별자를 $C[i]$의 값으로 한다. 그림 3은 새로운 후보해 생성을 위해 변이 연산을 적용한 예로서, 짙은 색 부분이 변이 확률에 의해 선택되어 변경된 요소이다.

	t_{11}	t_{12}	t_{13}	t_{21}	t_{22}	t_{31}	t_{32}
C	1	2	3	3	1	2	2
P	2	2	3	1	1	2	3

그림 3. 변이 연산의 적용

적합도 함수(Fitness Function)

후보 작업 할당에 대해 Lower-Layer-GA-for-Task-Schedule을 적용하여 구해진 가장 좋은 스케쥴의 적합도가 후보 작업 할당의 적합도가 된다.

3.4. 작업 스케쥴링을 위한 하위계층 유전알고리즘

상위계층 유전알고리즘에 의해 제공되는 작업 할당에 대해 효율적인 작업 스케쥴을 생성하는 하위계층 유전알고리즘의 기본 구성요소들은 다음과 같다.

인코딩 스키마(Encoding Scheme)

하위계층 유전알고리즘의 후보해는 Upper-Layer-GA-for-Task-Allocation에 의해 제시된 작업 할당에 대한 가능한 스케쥴이 된다. 후보해는 부작업들의 순서화된 리스트로서, 부작업의 선행 제약이 지켜져야 하는 위상적인 리스트로 볼 수 있다. 그림 4는 그림 1의 작업 할당에 대응하는 하나의 스케쥴을 표현한 것이다. 그림 4의 (a)는 스케쥴의 순서화된 리스트 표현이고, (b)는 (a)로 표현된 스케쥴에 대한 각 에이전트의 부작업 실행 순서를 나타낸 것이다.

(t_{12}, A_2)	(t_{31}, A_1)	(t_{11}, A_1)	(t_{21}, A_3)	(t_{13}, A_1)	(t_{22}, A_2)	(t_{32}, A_2)

(a) 그림 1.의 작업 할당에 대한 스케쥴 표현

A_1: t_{31} t_{11} t_{13}
A_2: t_{12} t_{22} t_{32}
A_3: t_{21}

(b) (a)에 대한 각 에이전트의 부작업 실행 순서

그림 4. 작업 스케쥴 표현 및 선행관계

해집단 초기화(Population initialization)

초기 해집단은 가능한 후보 스케쥴로 구성되는데, 해집단 생성을 위한 프로시져는 다음과 같다. 하나의 부작업 t_{jk}를 1에서 N까지의 범위를 가지는 인덱스 i로 나타낸다.

procedure *Generate-a-Candidate*

단계 1. 공백의 배열을 생성한다.

단계 2. 범위 1에서 N까지 중에서 임의로 인덱스 c를 선택한다.

단계 3. 부작업 t_{ij}에 해당하는 c가 배열에 포함되어 있지 않고, $PR_i(t_{ij})$의 모든 부작업의 인덱스가 배열에 포함되어 있다면, c를 배열의 끝에 추가한다.

단계 4. 1부터 N까지의 모든 부작업의 인덱스가 배열에 포함되어 있다면, 후보해로 배열을 반환하고, 그렇지 않으면 단계 2로 간다.

유전 연산자(Genetic Operators)

새로운 후보 스케쥴을 생성하기 위해 복제.교차.변이 연산자가 사용된다. 교차 연산자는 순위 기반 선택(rank-based selection) 방법에 의해 해집단에서 두 순서화된 리스트 P_1, P_2를 선택한다[8]. 리스트 P_1에서 부작업 O_i를 임의로 선택하고, O_i와 선행관계의 부작업 또한 선택한다. P_1의 선택된 부작업을 새로운 배열의 같은 위치로 복사한다. 그런 다음, P_2에서 나타나는 순서대로 나머지 부작업을 새로운 배열에 복사한다. 그림 5는 교차 연산이 이루지는 방법을 보인 예이다. P_1에서 t_{13}이 선택되고, t_{13}보다 t_{11}이 먼저 실행되어야 한다는 선행 제약 조건이 있다. 따라서 P_1의 t_{11}과 t_{13}은 새로운 후보 스케쥴 C_1의 같은 위치에 복사된다.

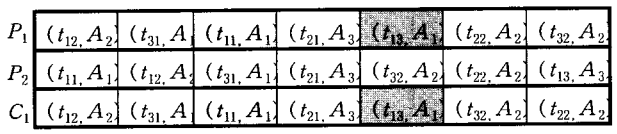

그림 5. 교차 연산의 적용

변이 연산은 먼저 해집단에서 선택된 하나의 순서화된 리스트를 새로운 리스트 $(O_1, O_2, \ldots, O_N)$로 복사하고, 리스트에서 임의의 위치 i를 선택한다. 부작업 O_i를 실행하는 에이전트에 의해 실행되는 가장 가까운 왼쪽과 오른쪽 부작업의 위치 인덱스 lmp와 rmp를 찾는다. 그런 다음 lmp+1에서 rmp-1의 범위에서 임의의 위치 p를 선택한다. 마지막으로, i번째 요소 O_i를 위치 p로 이동한다. 그림 6은 변이 연산자를 적용한 예를 보인 것이다. 여기서 t_{13}이 처음으로 선택되고, t_{13}보다 t_{11}이 먼저 처리되어야 한다는 제약 조건이 있다.

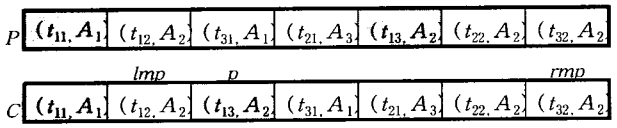

그림 6. 변이 연산의 적용

이러한 교차와 변이 연산자는 선행 제약조건을 보존한다는 것이 증명되었다[9]. 이러한 선행 조건 보존 연산자들은 염색체(chromosome)를 선행 제약조건이 포함된 작업 연산의 위상적 순서로 볼 수 있기 때문에 가능한 스케줄을 생성한다.

적합도 평가(Fitness Evaluation)

후보해의 적합도를 평가하기 위해서는 후보해에 해당하는 스케줄을 구성하는 부작업의 시작 시간과 종료 시간을 알아야 한다. 부작업의 시작 시간은 선행하는 부작업의 종료 시간과 그 부작업을 실행하는데 필요한 자원의 사용 가능 여부에 영향을 받는다.

만약 자원 R_j가 동시에 k개의 에이전트에 의해 공유될 수 있다면, R_j는 k개의 사용 트랙(usage tracks) $R_j.Tr_1, \ldots, R_j.Tr_k$를 가진다고 볼 수 있다. 각 트랙은 하나의 에이전트에 의해 점유될 수 있다. 자원 R_j는 각 트랙 $R_j.Tr_l$에 대한 개방 시간 변수(open time variable) $Op(R_j.Tr_l)$을 관리하는데, 이 변수는 트랙이 사용 가능하게 될 때를 나타낸다. 순서화된 리스트의 스케줄은 리스트의 부작업을 왼쪽에서 오른쪽으로 순서대로 나열함으로써 쉽게 얻어진다. 순서화된 리스트에 나타나는 순서에 따라, 부작업 O_i의 시작 시간 $O_i.start-time$ 은 다음과 같이 결정된다:

$O_i.start-time$=max{선행하는 부작업의 종료 시간, max{ O_i 실행에 필요한 각 자원에 대한 사용가능한 가장 빠른 시간}, max{ O_i와 같은 작업에 속하는 이미 스케줄된 부작업 종료 시간}}

에이전트 A_m에서의 O_i의 종료 시간은 다음과 같이 정해진다:

$$O_i.finish-time = O_i.start-time + PT_m(O_i)$$

$O_i.finish-time$이 결정되면, 에이전트 A_m이 점유하는 트랙 $R_j.Tr_l$의 개방 시간 $Op(R_j.Tr_l)$은 $O_i.finish-time$이 된다.

순서화된 리스트 즉, 후보해 L의 적합도 f(L)은 부작업의 시작 시간과 종료 시간으로부터 구할 수 있다. 응용분야에 따라 적용될 수 있는 적합도 함수의 몇가지 예는 다음과 같다:

$f(L) = \max O_i \in O\{O_i.finish-time\}$: 부작업 최종 종료 시각

$f(L) = \sum_{O_i \in O} O_i.finish-time$: 부작업 종료 시간의 합.

$f(L) = \sum_{O_i \in O} (O_i.finish-time - O_i.start-time)$: 부작업 실행 시간의 합.

4. 실험 및 분석

제안한 조정 협동 기법의 응용성을 보이기 위해, 다음과 같은 4개의 외부 자원과 5개의 에이전트로 구성된 멀티에이전트 시스템에 처리할 4개의 작업이 주어진 문제에 유전알고리즘 기반의 에이전트 조정 기법을 적용하였다.

$A=\{A_1, A_2, A_3, A_4, A_5\}$
$T=\{T_1, T_2, T_3, T_4\}$
$R=\{R_1, R_2, R_3, R_4\}$
$C(R_1)=2 \quad C(R_2)=3 \quad C(R_3)=1 \quad C(R_4)=4$
$T_1=\{O_2, O_4, O_6, O_7, O_1, O_5\}$
$PO_1=\{(O_2, O_6), (O_6, O_7), (O_2, O_1)\}$
$T_2=\{O_6, O_9, O_1, O_{10}, O_8\}$
$PO_2=\{(O_6, O_1), (O_1, O_{10}), (O_9, O_8)\}$
$T_3=\{O_2, O_4, O_5, O_7, O_9, O_8\}$
$PO_3=\{(O_4, O_7), (O_5, O_9)\}$
$T_4=\{O_1, O_4, O_3, O_8, O_7, O_2\}$
$PO_4=\{(O_1, O_3), (O_3, O_2), (O_4, O_7)\}$

$CO(A_1)=\{(O_1,2,\{R_1\}), (O_2,2,\{R_2,R_3\}), (O_4,4,\{R_3,R_4\}), (O_6,3,\{R_1\}), (O_7,4,\{R_2\}), (O_8,5,\{R_3\}), (O_9,6,\{R_4\}), (O_{10},5,\{R_3,R_4\})\}$

$CO(A_2)=\{(O_1,2,\{R_2\}), (O_2,3,\{R_1\}), (O_3,4,\{R_1,R_3\}), (O_4,4,\{R_1\}), (O_5,6,\{R_2,R_3\}), (O_6,5,\{R_2\}), (O_7,4,\{R_2,R_3\})\}$

$CO(A_3)=\{(O_4,5,\{R_1\}), (O_5,3,\{R_2\}), (O_6,4,\{R_3\}), (O_7,6,\{R_4\}), (O_8,3,\{R_1,R_2\}), (O_9,2,\{R_3,R_4\}), (O_{10},6,\{R_1,R_4\})\}$

$CO(A_4)=\{(O_1,4,\{R_4\}), (O_3,5,\{R_3\}), (O_5,3,\{R_1,R_2\}) (O_7,2,\{R_1\}), (O_9,6,\{R_2\}), (O_{10},5,\{R_3,R_4\})\}$

$CO(A_5)=\{(O_2,5,\{R_3,R_4\}), (O_3,4,\{R_1,R_2\}), (O_4,3,\{R_1,R_3,R_4\}), (O_6,6,\{R_1\}), (O_8,4,\{R_1\}), (O_{10},3,\{R_1\})\}$

유전알고리즘 기반의 조정 협동 기법을 위의 문제에 적용하여, 모든 부작업의 최종 종료 시간을 스케줄의 적합도로 채택한 실험의 결과는 아래와 같다. 실험에서 상위계층 유전알고리즘과 하위계층 유전알고리즘의 해집단 크기는 60, 세대 수는 각각 30과 25이고, 교차 비율은 각각 0.7과 0.9이다. 변이 연산자는 교차 연산자가 동일한 염색체를 선택할 때만 적용한다. 실험에서는 적합도 함수로서 최종작업의 종료시각을 사용하였다. 위 문제에 대한 작업 할당과 스케줄에 의해 멀티에이전트 시스템은 4개의 작업을 22 단위 시간에 완료한다.

작업 할당과 스케줄

$(O_i, A_j, [t_k, t_l])$은 해당 작업의 부작업 O_i이 에이전트 A_j에 할당되고 $[t_k, t_l]$동안에 실행되도록 스케줄링되었다는 것을 의미한다.

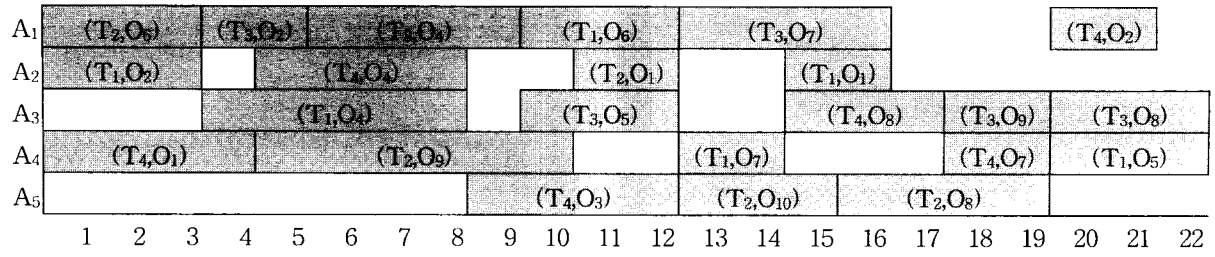

그림 7. 에이전트 스케쥴

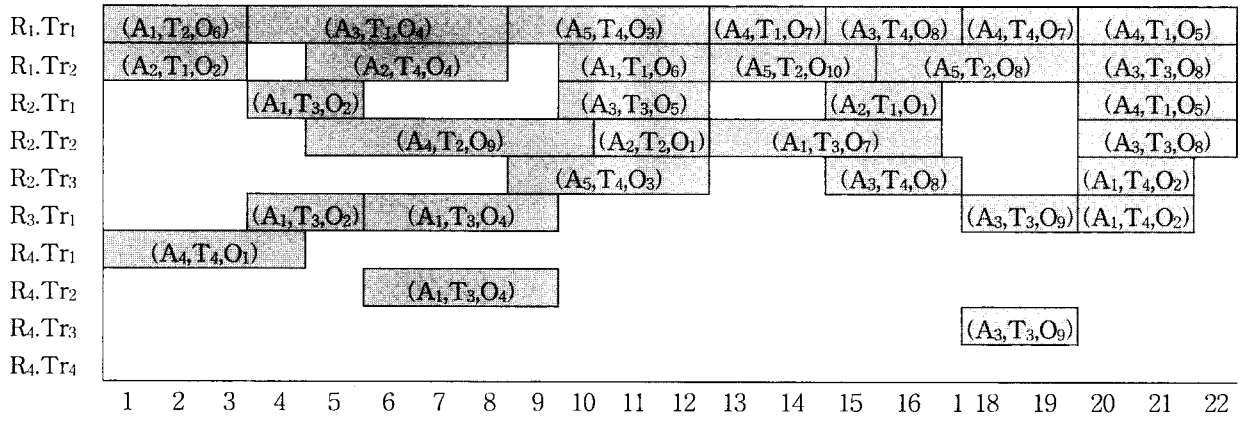

그림 8. 자원 스케쥴

T_1: ⟨($O_2, A_2, [0,3]$) ($O_4, A_3, [3,8]$) ($O_6, A_1, [9,12]$) ($O_7, A_4, [12,14]$) ($O_1, A_2, [14,16]$), ($O_5, A_4, [19,22]$)⟩
T_2: ⟨($O_6, A_1, [0,3]$) ($O_9, A_4, [4,10]$) ($O_1, A_2, [10,12]$) ($O_{10}, A_5, [12,15]$) ($O_8, A_5, [15,19]$)⟩
T_3: ⟨($O_2, A_1, [3,5]$) ($O_4, A_1, [5,9]$) ($O_5, A_3, [9,12]$) ($O_7, A_1, [12,16]$) ($O_9, A_3, [17,19]$) ($O_8, A_3, [19,22]$)⟩
T_4: ⟨($O_1, A_4, [0,4]$) ($O_4, A_2, [4,8]$) ($O_3, A_5, [8,12]$) ($O_8, A_3, [14,17]$) ($O_7, A_4, [17,19]$) ($O_2, A_1, [19,21]$)⟩

에이전트 스케쥴

($T_i, O_i, [t_k, t_l]$)는 작업 T_i의 부작업 O_i가 $[t_k, t_l]$동안 해당 에이전트로 스케쥴되었다는 것을 의미한다.

A_1: ⟨($T_2, O_6, [0,3]$) ($T_3, O_2, [3,5]$) ($T_3, O_4, [5,9]$) ($T_1, O_6, [9,12]$) ($T_3, O_7, [12,16]$) ($T_4, O_2, [19,21]$)⟩
A_2: ⟨($T_1, O_2, [0,3]$) ($T_4, O_4, [4,8]$) ($T_2, O_1, [10,12]$) ($T_1, O_1, [14,16]$)⟩
A_3: ⟨($T_1, O_4, [3,8]$) ($T_3, O_5, [9,12]$) ($T_4, O_8, [14,17]$) ($T_3, O_9, [17,19]$) ($T_3, O_8, [19,22]$)⟩
A_4: ⟨($T_4, O_1, [0,4]$) ($T_2, O_9, [4,10]$) ($T_1, O_7, [12,14]$) ($T_4, O_7, [17,19]$) ($T_1, O_5, [19,22]$)⟩
A_5: ⟨($T_4, O_3, [8,12]$) ($T_2, O_{10}, [12,15]$) ($T_2, O_8, [15,19]$)⟩

자원 스케쥴

($A_i, T_j, O_k, [t_l, t_m]$)은 시간 $[t_k, t_l]$동안 작업 T_j의 부작업 O_k을 실행하기 위해 에이전트 A_i에 의해 자원이 사용된다는 것을 의미하고, $R_a.Tr_b$는 자원 R_a의 트랙 Tr_b을 의미한다.

$R_1.Tr_1$: ⟨($A_1, T_2, O_6, [0,3]$) ($A_3, T_1, O_4, [3,8]$) ($A_5, T_4, O_3, [8,12]$) ($A_4, T_1, O_7, [12,14]$) ($A_3, T_4, O_8, [14,17]$) ($A_4, T_4, O_7, [17,19]$) ($A_4, T_1, O_5, [19,22]$)⟩
$R_1.Tr_2$: ⟨($A_2, T_1, O_2, [0,3]$) ($A_2, T_4, O_4, [4,8]$) ($A_1, T_1, O_6, [9,12]$) ($A_5, T_2, O_{10}, [12,15]$) ($A_5, T_2, O_8, [15,19]$) ($A_3, T_3, O_8, [19,22]$)⟩
$R_2.Tr_1$: ⟨($A_1, T_3, O_2, [3,5]$) ($A_3, T_3, O_5, [9,12]$) ($A_2, T_1, O_1, [14,16]$) ($A_4, T_1, O_5, [19,22]$)⟩
$R_2.Tr_2$: ⟨($A_4, T_2, O_9, [4,10]$) ($A_2, T_2, O_1, [10,12]$) ($A_1, T_3, O_7, [12,16]$) ($A_3, T_3, O_8, [19,22]$)⟩
$R_2.Tr_3$: ⟨($A_5, T_4, O_3, [8,12]$) ($A_3, T_4, O_8, [14,17]$) ($A_1, T_4, O_2, [19,21]$)⟩
$R_3.Tr_1$: ⟨($A_1, T_3, O_2, [3,5]$) ($A_1, T_3, O_4, [5,9]$) ($A_3, T_3, O_9, [17,19]$) ($A_1, T_4, O_2, [19,21]$)⟩
$R_4.Tr_1$: ⟨($A_4, T_4, O_1, [0,4]$)⟩
$R_4.Tr_2$: ⟨($A_1, T_3, O_4, [5,9]$)⟩
$R_4.Tr_3$: ⟨($A_3, T_3, O_9, [17,19]$)⟩
$R_4.Tr_4$: ⟨ ⟩

그림 7과 그림 8은 에이전트와 자원에 대한 스케쥴을 나타내는 간트 차트(Gantt chart)이다. 그림 9는 유전진화에 의한 최적해와 평균해를 보인다.

자원의 배타적 사용을 고려하지 않고 연산이 빠른 에이전트에서 수행될 때, 최적 makespan의 하한(lower bound)은 작업들의 연산 처리 시간의 합의 최대값 즉, $\max_{T_i \in T} \{ \sum_{t_{ij} \in T_i} \{ \min_{k \in CA(t_{ij})} PT_k(t_{ij}) \} \}$이다. 위 예에서 하한은 16이다. 자원 제약조건과 자원에 대한 에이전트들의 경쟁으로 인해, 실제 makespan은 하한보다 반드시 커야 한다. 그러므로, 유전알고리즘 기반의 에이전트 조정 기법이 최적해을 찾는다는 것을 보장할 수는 없을 지라도, 효율적인 작업 할당과 스케쥴을 찾을 수 있다.

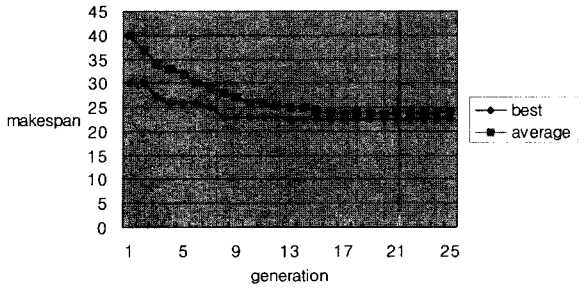

그림 9. 유전진화 과정의 최적 성능과 평균 성능

두번째 실험에서는 위의 실험환경에 대해서 아래와 같은 5개의 작업들에 대해서 제안한 방법을 적용하였다.

$T_1 = \{O_2, O_5, O_7, O_1, O_4\}$ $PO_1 = \{(O_5, O_7), (O_2, O_4)\}$
$T_2 = \{O_6, O_9, O_{10}, O_5, O_3, O_2\}$
$PO_2 = \{(O_6, O_5), (O_6, O_3), (O_5, O_2)\}$
$T_3 = \{(O_1, O_2, O_7, O_6, O_4, O_5\}$
$PO_3 = \{(O_1, O_7), (O_2, O_6), (O_7, O_5)\}$
$T_4 = \{O_2, O_4, O_6, O_8\}$ $PO_4 = \{(O_2, O_4), (O_2, O_8)\}$
$T_5 = \{O_3, O_6, O_9, O_{10}, O_1 O_4\}$
$PO_5 = \{(O_3, O_9), (O_9, O_1), (O_6, O_{4)}, (O_6, O_{10})\}$

첫 번째 실험에서와 같은 조건하에서 제안한 방법은 위 문제에 대해서 다음과 같이 24 단위시간을 갖는 작업할당과 스케쥴을 생성하였다.

작업 할당과 스케쥴

$T_1 = \{(O_2, A_1, [0,2]), (O_5, A_3, [2,5]), (O_7, A_4, [11,13]), (O_1, A_2, [14,16]), (O_4, A_2, [16,20])\}$
$T_2 = \{(O_6, A_1, [5,8]), (O_9, A_3, [8,10]), (O_{10}, A_5, [10,13]), (O_5, A_3, [13,16]), (O_3, A_5, [16,20]), (O_2, A_1, [20,22])\}$
$T_3 = \{(O_1, A_2, [0,2]), (O_2, A_2, [2,5]), (O_7, A_4, [7,9]), (O_6, A_2, [9,14]), (O_4, A_1, [16,20]), (O_5, A_3, [20,23])\}$
$T_4 = \{(O_2, A_1, [2,4]), (O_4, A_5, [4,7]), (O_6, A_1, [16,20]), (O_8, A_5, [20,24])\}$
$T_5 = \{(O_3, A_5, [0,4]), (O_6, A_1, [8,11]), (O_9, A_3, [11,13]), (O_{10}, A_5, [13,16]), (O_1, A_4, [16,20]), (O_4, A_2, [20,24])\}$

에이전트 스케쥴

A_1: ⟨$(T_1, O_2, [0,2])$ $(T_4, O_2, [2,4])$ $(T_2, O_6, [5,8])$ $(T_5, O_6, [8,11])$ $(T_4, O_6, [13,16])$ $(T_3, O_4, [16,20])$ $(T_2, O_2, [20,22])$⟩
A_2: ⟨$(T_3, O_1, [0,2])$ $(T_3, O_2, [2,5])$ $(T_3, O_6, [9,14])$ $(T_1, O_1, [14,16])$ $(T_1, O_4, [16,20])$ $(T_5, O_4, [20,24])$⟩
A_3: ⟨$(T_1, O_5, [2,5])$ $(T_2, O_9, [8,10])$ $(T_5, O_9, [11,13])$ $(T_2, O_5, [13,16])$ $(T_3, O_5, [20,23])$⟩
A_4: ⟨$(T_3, O_7, [7,9])$ $(T_1, O_7, [11,13])$ $(T_5, O_1, [16,20])$⟩
A_5: ⟨$(T_5, O_3, [0,4])$ $(T_4, O_4, [4,7])$ $(T_2, O_{10}, [10,13])$ $(T_5, O_{10}, [13,16])$ $(T_2, O_3, [16,20])$ $(T_4, O_8, [20,24])$⟩

자원 스케쥴

$R_1.Tr_1$: ⟨$(A_5, T_5, O_3, [0,4])$ $(A_5, T_4, O_4, [4,7])$ $(A_4, T_3, O_7, [7,9])$ $(A_5, T_2, O_{10}, [10,13])$ $(A_5, T_5, O_{10}, [13,16])$ $(A_5, T_2, O_3, [16,20])$ $(A_5, T_4, O_8, [20,24])$⟩
$R_1.Tr_2$: ⟨$(A_2, T_3, O_2, [2,5])$ $(A_1, T_2, O_6, [5,8])$ $(A_1, T_5, O_6, [8,11])$ $(A_4, T_1, O_7, [11,13])$ $(A_1, T_4, O_6, [13,16])$ $(A_2, T_1, O_4, [16,20])$ $(A_2, T_5, O_4, [20,24])$⟩
$R_2.Tr_1$: ⟨$(A_1, T_1, O_2, [0,2])$ $(A_1, T_4, O_2, [2,4])$ $(A_2, T_3, O_6, [9,14])$ $(A_5, T_2, O_3, [16,20])$⟩
$R_2.Tr_2$: ⟨$(A_5, T_5, O_3, [0,4])$ $(A_2, T_1, O_1, [14,16])$ $(A_3, T_3, O_5, [20,23])$⟩
$R_2.Tr_3$: ⟨$(A_2, T_3, O_1, [0,2])$ $(A_3, T_1, O_5, [2,5])$ $(A_3, T_2, O_5, [13,16])$ $(A_1, T_2, O_2, [20,22])$⟩
$R_3.Tr_1$: ⟨$(A_1, T_1, O_2, [0,2])$ $(A_1, T_4, O_2, [2,4])$ $(A_5, T_4, O_4, [4,7])$ $(A_3, T_2, O_9, [8,10])$ $(A_3, T_5, O_9, [11,13])$ $(A_1, T_3, O_4, [16,20])$ $(A_1, T_2, O_2, [20,22])$⟩
$R_4.Tr_1$: ⟨$(A_5, T_4, O_4, [4,7])$ $(A_4, T_5, O_1, [16,20])$⟩
$R_4.Tr_2$: ⟨$(A_3, T_2, O_9, [8,10])$⟩
$R_4.Tr_3$: ⟨$(A_3, T_5, O_9, [11,13])$⟩
$R_4.Tr_4$: ⟨$(A_1, T_3, O_4, [16,20])$⟩

5.결론 및 향후 과제

이 논문에서는 멀티에이전트 시스템에서의 협력 작업을 위한 유전알고리즘 기반의 에이전트 조정 기법을 제안하였다. 고려하는 멀티에이전트 시스템에서, 모든 작업은 주어지는 시점에 이미 부분적으로 순서화된 부작업들이 어떤 순서로 처리될 것인가에 대한 미리 결정된 계획을 가진다고 가정한다. 인공지능과 멀티에이전트 시스템에서 중요한 문제 중 하나인 계획기법은 그 자체로 복잡한 문제지만, 제안한 기법은 생성된 계획이 실행될 때 즉, 계획 다음 단계로써 작업 할당과 스케쥴링을 수행한다.

조정 작업은 에이전트에 부작업을 할당하는 것과 멀티에이전트 시스템의 전체적인 성능을 최대화하도록 각 에이전트에 할당된 부작업에 대한 스케쥴을 찾는 것이다. 제안한 기법은 한번에 하나의 에이전트에 의해 사용되거나 여러 에이전트에 의해 공유될 수 있고, 비선점(nonpreemptive)인 다수의 외부 자원이 존재하는 멀티에이전트 시스템 환경에서 효율적인 작업 할당과 작업 스케쥴을 생성한다. 에이전트가 동시에 다수의 자원을 필요로 할 수 있지만, 제안하는 기법은 유전 진화 동안 생성되는 염색체에 표현된 순서로 부작업을 스케쥴링하고, 부작업 실행에 필요한 모든 자원이 동시에 할당하기 때문에 교착 상태에 빠지지 않는다는 것을 보장한다. 실험을 통해 제안한 기법이 효율적인 작업 할당과 스케쥴을 생성한다는 것을 알 수 있었다.

에이전트는 스케쥴에 따라 부작업을 실행하는 동안 실패할 수 있다. 이런 경우, 실패한 부작업에 대한 재할당과 재스케쥴링이 필요하다. 예를 들어, 에이전트 A_i가 시간 t에 어떤 부작업 실행에 실패했다고 가정하자. 실패한 에이전트의 부작업만을 재스케쥴링하고 재할당 하는 것은 효율적이지 않다. 더욱이, 실패가 발생한 경우, 실행되지 않고 남아있는 모든 부작업을 재할당하고 재스케쥴링하기 위해 모든 에이전트를 정지시키는 것은 좋은 방법이 아니다. 이러한 예외 상황을 처리하기위해, 재할당과 재스케쥴링이 수행되는 기간을 나타내는 스케쥴링 윈도우(scheduling window) SW 개념을 도입한다. A_i를 제외한 모든 에이전트는, 스케쥴된 실행 시간이 [t, t+SW]와 부분적 또는 전적으로 겹치는 부작업을

현재의 스케쥴에 따라 실행한다. 실행되지 않고 남아있는 부작업의 효율적인 작업 할당과 스케쥴을 찾기위해, 제안한 유전 알고리즘 기반 조정 기법이 적용될 수 있다. 유전알고리즘 기반 접근법은 유전알고리즘이 탐색 과정에서 사용했던 가능한 해를 유지하고 있기 때문에 실패 처리(failure handling)에 장점을 가진다. 이러한 전략은 작업이 동적으로 발생하는 온라인 스케쥴링에도 적용될 수 있다. 즉, 특정 시간 간격의 스케쥴링 윈도우 크기를 결정하고, 윈도우 시간 간격동안 발생하는 작업을 수집한다. 그런 다음, 다음 윈도우 동안 수집된 작업에 대해 제안한 방법을 사용하여 효율적인 작업 할당과 스케쥴을 찾는다.

제안한 기법은 중개 에이전트가 모든 조정 작업을 담당하는 중앙 접근법으로써, 중개 에이전트와 다른 에이전트간의 통신 병목현상이 발생할 수 있다. 또한, 유전알고리즘의 고유한 특징 때문에, 제안한 기법은 효율적인 해를 제시하는데 있어 어는 정도의 시간이 필요하다. 그러므로, 제안한 기법은 짧은 반응 시간을 요구하는 문제에는 적용할 수 없지만, 작업이 일괄적으로 제공되거나 많은 처리시간을 요하는 제조공정 시스템이나 정보 처리 어플리케이션 분야에는 적절하게 적용될 수 있다.

참고문헌

[1] J. Blazewicz, K. Ecker, G. Schmit, J. Weglarz, Scheduling in Computer and Manufacturing Systems, Springer-Verlag, 1993.

[2] J. Ferber, Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, Addison Wesley Longman, 1999.

[3] G. Weiss, Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence(eds.), The MIT Press, 1999.

[4] K.S. Decker, V.R. Lesser, Designing a family of coordination algorithms, In ICMAS-95, (1995).

[5] M.E. desJardins, E.H. Durfee, C. Le Ortiz, Jr., M.J. Wolverton, A survey of Research in Distributed, Continual Planning, AI Magazine, Winter (2002).

[6] E.H. Durfee, V.R. Lesser, Partial global planning: A coordination framework for distributed hypothesis formation, IEEE Trans. on Systems, Man, and Cybernetics, KDE-1, (1991) 63-83.

[7] S. J. Russell, P. Norvig, Artificial Intelligence:A Morden Approach, Prentice Hall, 1995.

[8] M. Mitchell, An Introduction to Genetic Algorithms, The MIT Press, 1998.

[9] K. -M. Lee, T. Yamakawa, K. M. Lee, "Genetic algorithm approaches to job shop scheduling problems: An overview", Int. Journal of Knowledge-based Intelligent Engineering Systems, NO. 2, pp.72-85, 2000.

[10] M. Vazquez, L. D. Whitley, A Comparison of Genetic Algorithms for the Dynamic Job Shop Scheduling Problem, GECCO-2000, pp. 1011-1018, 2000.

[10] H.-L. Fang, Genetic Algorithms in Timetabling and Scheduling, Ph.D. Dissertation, Univ. of Edingburgh, 1994.

[11] J. Hurink, B. Jurisch, M. Tole, Tabu search for the job shop scheduling problems with multi-purpose machines, In Operations Research Spectrum, Vol. 15, pp.205-215, 1994.

[12] S. Kobayshi, I. Ono, M. Yamamura, An Efficient Genetic Algorithm for Job Shop Scheduling Problems, AI Magazine, Winter, 2002.

[13] P. Laborie, Algorithims for propagating resource constraints in AI planning and scheduling: Existing approaches and new results, Artificial Intelligence, Vol. 143, pp.151-188, 2003.

[14] H. M. Deitel, Operating Systems, Addison Wesley. 1990.

저 자 소 개

이건명(Keon-Myung Lee)

KAIST 전산학 학사, 석사, 박사 (1990, 1992, 1995)
프랑스, INSA de Lyon, Postdoc Fellow (1995-1996)
미국 PSI사, Staff Scientist (1996)
충북대학교 전기전자컴퓨터공학부 부교수(1996-현재)
Univ. of Colorado at Denver, 객원 교수(2001-2003)

관심분야 : 에이전트, 소프트컴퓨팅, 기계학습, 데이터마이닝, 바이오인포메틱스

손봉기(Bong-Ki Sohn)

서원대학교 전산학 학사(1998)
충북대학교 전산학 석사(2000)
충북대학교 박사과정 (2000-현재)

관심분야 : 에이전트 시스템, 기계학습, 워크플로우 시스템