Simulated Annealing Algorithm의 변형을 지원하기 위한 객체지향 프레임워크 설계
Designing an Object-Oriented Framework for the Variants of Simulated Annealing Algorithm

정영일(연세대학교 컴퓨터•산업공학부)
유제석(연세대학교 컴퓨터•산업공학부)
전 진(고려대학교 정보통신기술공동연구소)
김창욱(연세대학교 컴퓨터•산업공학부)

Abstract

Today, meta-heuristic algorithms have been much attention by researcher because they have the power of solving combinational optimization problems efficiently. As the result, many variants of a meta-heuristic algorithm (e.g., simulated annealing) have been proposed for specific application domains. However, there are few efforts to classify them into a unified software framework, which is believed to provide the users with the reusability of the software, thereby significantly reducing the development time of algorithms.

In this paper, we present an object-oriented framework to be used as a general tool for efficiently developing variants of simulated annealing algorithm. The interface classes in the framework achieve the modulization of the algorithm, and the users are allowed to specialize some of the classes appropriate for solving their problems. The core of the framework is Algorithm Configuration Pattern (ACP) which facilitates creating user-specific variants flexibly. Finally, we summarize our experiences and discuss future research topics.

1. 서론

대형 수리문제 중 특히 NP-hard의 대형 최적화 문제는 문제의 크기가 커짐에 따라 컴퓨터 용량과 계산시간의 한계를 갖는다. 이러한 문제에 대해서 빠른 시간에 근사최적해를 찾을 수 있는 발견적(heuristic) 기법에 관한 연구가 주로 이루어지고 있다. 하지만 발견적 기법의 연구는 해결하고자 하는 문제마다 각기 그 특성에 맞추어 개발해야 하는 어려움이 있다. 이런 특정문제가 갖는 정보에 크게 구속되지 않고 다양한 문제에 적용 가능한 상위수준의 발견적 기법이 바로 메타휴리스틱(metaheuristic)이다.

메타휴리스틱 기법으로는 Genetic Algorithm, SA(Simulated Annealing), Tabu Search 가 있으며 이들 기법들은 개념과 이론이 단순하고 해공간의 탐색성능이 우수하여 공학, 자연과학, 경영학, 사회과학 등의 최적화 분야와 의사결정분야에 응용이 가능하기 때문에 오늘날 많은 관심의 대상이 되어 왔다. 이러한 메타휴리스틱 알고리즘의 많은 기법들 또한, 구체적인 응용 도메인에 있어서 제안되어 왔지만, 사용자로 하여금 개발 기간을 단축시킬 수 있는 재사용 가능한 통합된 소프트웨어 프레임워크(framework)로 체계화 하려는 노력은 거의 없었다.

본 논문에서는 메타휴리스틱 중에서도 SA 알고리즘을 효과적으로 개발하는 일반적인 툴로서 사용할 수 있는 객체지향 프레임워크를 소개한다. 프레임워크의 인터페이스들은 알고리즘들을 모듈화하고 있기 때문에 사용자들은 프레임워크를 해당 조건에 맞게 적절하게 구현하여 특정한 문제에 있어서 조합적 최적화 문제를 풀 수 있다. 프레임워크의 핵심은 사용자에게 유연하게 알고리즘의 변형을 생성할 수 있는 ACP(Algorithm Configuration Pattern)를 제공한다.

본 논문의 2절에서는 기존연구를 살펴보고, 3절에서는 SA 알고리즘들을 일반화하고 ACP 프레임워크를 제시한다. 4절에서는 ACP의 알고리즘 제어 및 조합패턴을 다루고 있으며, 5절에서는 ACP 프레임워크는 사용된 디자인 패턴(design pattern)들을 소개한다. 6절에서는 실제 적용 사례 연구를 살펴본다.

2. 기존 연구

기존의 연구는 크게 국부탐색 알고리즘들의 일반 프레임워크에 대한 연구와 문제에 대한 객체지향적 설계에 관한 연구로 나누어볼 수 있다. 전자의 경우 대부분 전산과학분야의 연구로서 국부탐색 알고리즘의 일반적 구조, 즉 neighborhood 구성과 해 이동 선택 등을 프레임워크화 하는 연구이다. 주로 C++언어의 template 기법을 이용하여 알고리즘의 재사용을 추구한다. 반면에 후자의 경우는 국부탐색 알고리즘을 이용하여 OR 문제들을 해결하기 위하여 문제의 객체모델을 제안하는 연구이다. 국부탐색 알고리즘을 일반화한 프레임워크에 관한 연구는 각각의 국부탐색 알고리즘을 재사용 단위로 보기 때문에 알고리즘 자체의 개선을 통한 성능 향상보다는 알고리즘의 표현에 초점을 맞추고 있거나([1],

[2], [3]) 단위 국부탐색 알고리즘의 조합을 통한 하이브리드(hybrid) 알고리즘 실행에 초점을 맞추고 있다[4].

또한, OR 문제를 해결하기 위한 접근들은 알고리즘과 문제를 구별하지 않고 일반적인 객체지향 기법을 적용하거나, 문제의 일반화에만 중심에 둔 연구가 진행되었다[5]. 그러나 산업공학 분야에서의 국부탐색 알고리즘을 이용한 문제 해결은 기존 알고리즘을 단순 적용하거나 기존 알고리즘에 맞도록 문제를 바꿔서 적용하지 않는 경향이 있다. 즉, 특정 문제에 적합한 알고리즘 모델상수 조합을 찾거나 기존 알고리즘의 흐름을 조정하여 보다 향상된 해 탐색 능력을 추구한다. 본 연구의 대상인 SA 알고리즘의 경우, 기본적인 알고리즘인 SAO(Simulated Annealing Original)를 SE(Stochastic Evolution)와 ASA(Accelerated Simulated Annealing)등으로 개선하여 문제에 적용하고 있는 것이다. 그러므로, 알고리즘의 개선을 보장하는 알고리즘 프레임워크를 제공한다면 보다 효율적인 연구를 가능하게 할 것이다.

본 연구는 여러 형태의 국부탐색 알고리즘을 일반화하는 형태가 아닌 연구이며, 여러 국부탐색 알고리즘의 한 분류인 SA 알고리즘들을 프레임워크화하여 여러 SA 변형 알고리즘을 표현할 수 있도록 제안하였다. 알고리즘의 재사용 가능한 부분인 SA 알고리즘 기본 절차와 흐름을 제어하는 단위를 분석, 정의하였다. 또한 이들을 조합하여 원하는 SA 알고리즘을 만들 수 있도록 ACP를 제안한다.

3. SA 분석

3.1 SA 알고리즘의 일반화

Step \ Algorithm		ACP Simulated Annealing Original	ACP Stochastic evolution
Algorithm begin		Initialize(X, T, L); X_best = X;	Initialize(X, T, L) X_best = X; Counter = 0;
Outer Loop	Outer begin	repeat Costold = C(X); Costbest = C(X_best);	repeat Costold = C(X); Costbest = C(X_best);
	Inner Loop: Begin	for i = 0 to L do	for i = 0 to L do
	Inner Loop: PERTURB	Y = PERTURB(X);	Y = PERTURB(X);

	Inner Loop: ACCEPT Criterion	if(C(X) > C(Y) or
		(exp((C(X) – C(Y))/T) > random (0,1))
		then X = Y;{accept the movement}
		endif;
			GAIN (m) = C(X) – C(Y);
			if(GAIN (m) > RANMIN(-T, 0))
			then X = Y;
			endif;

	Inner Loop: X_Best Update	if(C(X) < C(x_Best) then x_Best = x;	if(C(X) < C(x_Best) then x_Best = x;
	Inner Loop: Counter2: N		
	Inner Loop: End	endfor;	endfor;
	After Inner Loop End	Costnew = C(X);	Costnew = C(X);
	Update	UPDATE(T, Costold, Costnew);	UPDATE(T, Costold, Costnew);

	Counter2 : R or Counter1 : M		if(C(X) < Costbest) then
			Costbest = C(X);
			Counter = Counter – R;
			else
			Counter = Counter + 1;
			endif;

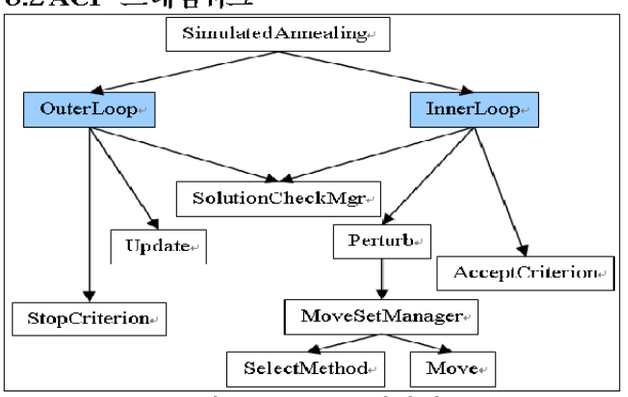
	Outer Loop StopCriterion	until (Stop-criterion) end	until (Counter > R); end

[표 1] ASO, SE 알고리즘 수정

SA은 크게 반복적 개선법을 기본으로 하되 비용증가의 이동을 확률적으로 허용하는 SAO와 큰 규모의 상승이동은 최적화 과정에 도움이 되지 않으므로 작은 규모의 상승이동만을 받아들이다가 적절하게 국부을 빠져나가는 SE, 그리고 SAO의 장점을 유지하고 해의 개선여부와 해의 변화여부를 판단하여 수렴속도를 개선하는 ASA가 있다.

SA들의 알고리즘을 분석한 결과, 모든 SA는 초기화부분과 외부루프로 구성될 수 있다. 외부루프는 전체흐름을 제어하고 내부루프와 종료조건을 가지며 내부루프는 새로운 해를 선택하여 그 해를 수락하는 기준 및 전체흐름을 제어하는 부분으로 구성된다. 이러한 알고리즘 단계를 일반화시킨 것이 [표 1]의 왼쪽 스텝이고, 이를 바탕으로 SAO와 SE를 수정한 것이 [표 1]이다. 이와 같은 일반화된 알고리즘 흐름은 위에서 제시한 SA알고리즘들뿐만 아니라 다른 형태의 SA 알고리즘을 표현할 수 있으며 이를 구현한 것이 본 논문의 ACP 프레임워크이다. (그림 1 참조)

3.2 ACP 프레임워크

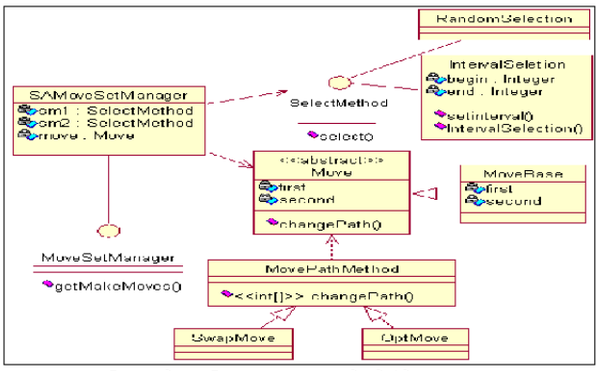


[그림 1] ACP 프레임워크

[그림 1]의 ACP 프레임워크에서 OuterLoop와 InnerLoop는 SA 알고리즘의 흐름을 정적으로 관리한다. InnerLoop는 새로운 해를 Perturb를 이용해서 생성한 후, 해의 목적함수 값을 구한다. 이러한 새로운 해를 채택하는 조건은 AcceptCriterion에서 수행한다. 또한, StopCriterion을 위해서 해의 개선이나 해의 변경여부를 판단하는 SolutionCheckMgr를 사용한다. OuterLoop는 Update에서 T(온도상수)를 수정한 후, InnerLoop와 마찬가지로 SolutionCheckMgr을 이용하여 StopCriterion을 살펴본다.

[그림 2]는 Perturb에서 새로운 Solution를 생성하기 위한 Move 프레임워크을 보여준다. 이러한 Move 프레임워크는 재사용 가능한 부분이기 때문에 SA의 Pertrub 뿐만 아니라 향후 연구 방향인 Genetic Algorithms과 Tabu Search와의 확장에 있어서도 적용할 수 있다.

실제 사용 가능한 여러 SA 알고리즘을 위해서는 이러한 프레임워크뿐만 아니라 여러형태의 알고리즘



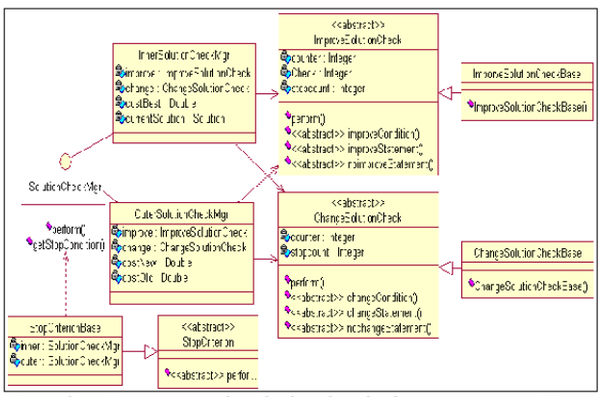
[그림 2] Move 프레임워크

제어와 조합이 가능해야 한다. 이를 위하여 ACP로서 알고리즘 제어 패턴과 알고리즘 조합 패턴을 제시한다.

4. ACP

4.1 알고리즘 제어 패턴

SE, ASA에서는 OuterLoop와 InnerLoop의 해 개선 및 변경 여부를 판별하고 이것을 OuterLoop의 종료조건으로 사용한다. 이것이 전체 알고리즘의 흐름제어 단위이며, 각각 ImproveSolutionCheck와 ChangeSolutionCheck로 나누었다. OuterLoop와 InnerLoop에서 각각 흐름제어 단위를 조합해서 사용가능 하도록 [그림 3]과 같이 InnerSolutionCheckMgr과 OuterSolutionCheckMgr로 나누었다.



[그림 3] Solution의 개선 및 변경, OuterLoop 종료

4.2 알고리즘 조합 패턴

SA 알고리즘의 핵심은 T(온도상수)를 이용하여 국부해탈출과 종료를 결정하는 것이다. 대표적인 방법으로 시간에 따라 온도상수를 낮춤으로서 해를 수렴하게하는 CoolingSchedule과 SE에서의 상승이동을 받아들이는 확률을 관장하여 국부해를 탈출하는 HeatingSchedule이 있다. 이와 같은 방법은 사용자의 요구에 따라 선택되기도 하고 조합해서 사용할 수도 있다.

본 논문에서는 각각의 스케쥴을 위해서 CoolingCriterion과 CoolingUpdate, HeatingCriterion과 HeatingUpdate 클래스를 제공한다. 또한, 두 가지 스케쥴을 논리조합(AND, OR)으로 구성한 Hybrid Criterion과 UpateManager 클래스를 제공한다.

5. ACP 프레임워크의 구현

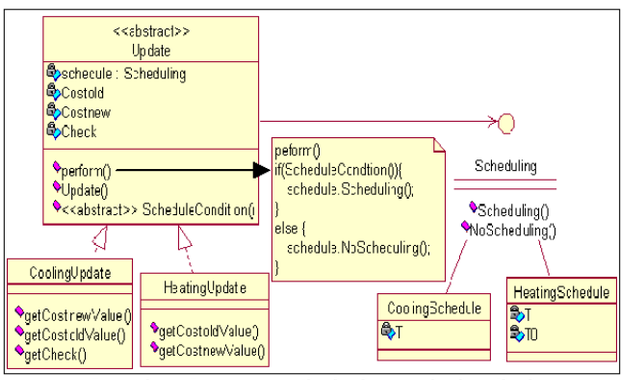
ACP 프레임워크를 구성하는데 있어서 본 논문의 [10]에서 설명하는 디자인 패턴들중 아래의 3가지 패턴을 적용하였다.

5.1 템플릿 메소드 패턴 (Template Method Pattern)

ACP 프레임워크의 흐름을 제어하는 InnerLoop, OuterLoop, StopCriterion, Perturb, AcceptCriterion, Update등의 핵심 객체들을 모두 템프릿 메소드 패턴으로 정의하였다. ACP 프레임워크는 템플릿 메소드인 perform을 통하여 알고리즘의 순서를 정의 하였다. 사용자는 각각의 primitiveOperation인 알고리즘 메소드를 재정의하여 알고리즘을 수정할 수 있다.

5.2 브릿지 패턴 (Bridge Pattern)

ACP 프레임워크에서 Scheduling의 CoolingSchedule과 HeatingSchedule은 각각 온도와 확률을 관장하는 컨트롤 파라미터를 관리하는 객체이다. Update는 스케쥴에 상응하는 Update의 조건을 판단하여 각각의 적절한 형태의 스케쥴을 호출하는 템플릿 메소드 패턴을 이용한 객체이다. [그림 4]의 브릿지 패턴을 이용하여 구현자(스케쥴)과 추상화(Update)를 분리하였기 때문에 사용자로부터 Update조건에 따르는 온도변화와 확률변화를 갱신하는 부분을 숨김으로써 확장성이 커지고 재사용성이 뛰어나게 되었다.

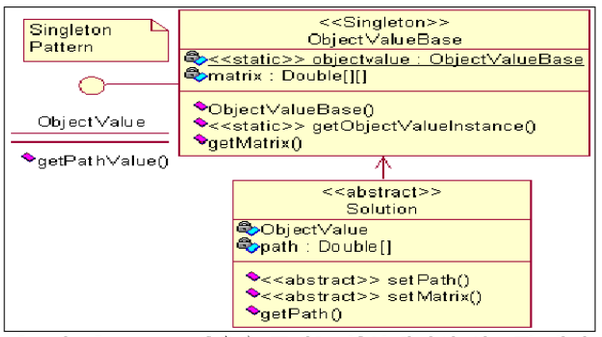


[그림 4] Update에서의 브릿지 패턴

5.3 싱글톤 패턴 (Singleton Pattern)

주어진 문제의 각 노드간의 비용을 저장하는 매트릭스(matrix)는 변하지 않는 부분이므로 Solution객체의 매트릭스를 저장하는 과정에서 ObjectValue의 인스턴스를 생성한다. ObjectValue객체는 getPathValue메소드로 현재의 Solution, BestSolution, PerturbSolution등의 패스(path)에 대한 각각의 Solution의 비용을 구

할 수 있게끔 해준다. 이러한 Solution의 비용을 구하는 부분은 변동되지 않고 하나의 인스턴스만을 필요로 하므로 이 부분에서 [그림 5]와 같이 싱글톤 패턴을 적용하였다.



[그림 5] Solution 비용을 구하는 부분에서의 싱글톤 패턴

6. 사례 연구

```
class Main {
    Solution initialSolution = new MySolution( );
    SelectMethod sm = new RandomSelection();
    MovePathMethod move = new SwapMove();
    MoveSetManger moveset = new SAMoveSetManager(sm, move);
    Perturb perturb = new PerturbBase(moveset);
    AcceptCriterion heating_criterion = new HeatingCriterionBase();
    AcceptCriterion hc = new HeatingCriterion(heating_criterion);

    InnerLoop innerloop = new InnerLoopBase(perturb, hc, loopcount);

    Update heating = new HeatingUpdate(new HeatingSchedule( ));
    UpdateManager update = new   UpdateManager(heating);
    ImproveSolutionCheck is = new ImproveSolutionCheckBase(StopCount);

    OuterSolutionCheckMgr out = new OuterSolutionCheckMgr(is);
    StopCriterion stop = new StopCriterion(out);
    OuterLoop outer = new OuterLoop( innerloop, update, stop );
    SimulatedAnneling sa = new SimulatedAnneling(initialSolution, outer, true);
    sa.startSolving();
}
```

[그림 6] ACP 프레임워크을 이용한 SE 알고리즘

[그림 6]은 ACP 프레임워크를 이용하여 SE 알고리즘을 구현한 예제이다. 먼저 주어진 문제에 대하여 적절한 초기해를 생성한다. Move방식은 랜덤으로 하여, 생성한 두 노드를 교체하는 구조로 Move를 만들고 이를 Perturb로 정의한다. 이러한 Perturb정보와 국부해를 빠져나가는 기준은 ACP의 조합 패턴인 HeatingCriterion을 사용하여 InnerLoop의 반복횟수인 loopCount만큼 실행 하도록 InnerLoop를 설정한다. Update에서는 T를 설정하는 Update를 생성한다. ACP의 제어 패턴에서 언급한 OuterSolutionCheckMgr에 ImproveSolution Check를 사용하여 해의 개선여부에 따른 종료조건을 판단한다.

7. 결론

본 연구에서는 여러 형태의 국부탐색 알고리즘을 일반화하는 형태가 아닌 연구이며, 여러 국부탐색 알고리즘의 한 분류인 SA 알고리즘들을 프레임워크화하여 여러 SA 변형 알고리즘을 표현할 수 있도록 제안했다. 이를 위해 알고리즘의 재사용 가능한 부분인 SA 알고리즘의 기본 절차와 흐름을 제어하는 단위를 분석 및 정의하였고, 또한 이들을 조합하여 원하는 SA 알고리즘을 만들 수 있도록 ACP를 제시하였다.

본 연구에서 제시한 ACP 프레임워크는 재사용 가능한 모듈들을 이용하여 기본적인 SA 알고리즘 뿐만 아니라 사용자가 원하는 진보된 SA 변형 알고리즘을 빠른시간안에 개발할 수 있도록 해준다. 앞으로의 연구는 TabuSearh와 Genetic Algorithm의 프레임워크를 통합시키는 것이다.

References

[1] Laurent Michel and Pascal Van Hentenryck, “Localizer++: An Open Library for Local Search”, *Constraints* 5(1/2): 43-84 (2000).

[2] François Laburthe and Yves Caseau, “SALSA : A Language for Search Algorithms”, *Constraints : an international journal*, v.7 no.3/4, 2002, pp.255-288.

[3] Andreas Fink and Stefan Voß, “HotFrame : A Heuristic Optimization Framework “. In: S. Voß, D.L. Woodruff (Eds.), *Optimization Software Class Libraries*, Kluwer, Boston (2002), 81-154.

[4] Luca Di Gaspero and Andrea Schaerf, “EasyLocal++: An object-oriented framework for flexible design of local search algorithms”, *Technical Report UDMI/13/2000/RR, Dipartmento di Matematica e Informatica, Universita di Udine,* 2000.

[5] Liu Shixin and Wang Mengguang, “An object-oriented methodology for solving the RCPSPs with heuristics and metaheuristics”, *Production planning & control*, v.11 no.5, 2000, pp.434-442.

[6] Andreas Fink, et al, “Building Reusable Software Components for Heuristic Search”, 1998. *Extended abstract of the talk given at OR98*, Zurich,.

[7] Alexandre A. Andreatta, Sergio E.R. Carvalho and Celso C. Ribeiro, “An Object-Oriented Framework for Local Search Heuristics”, *Technology of Object-Oriented Languages*, 1998. TOOLS 26. Proceedings, pp.33-45.

[8] 김여근, 윤복식, 이상복 (1997), “메타 휴리스틱”, 영지문화사, 185-258.

[9] Zbigniew Michalewicz and David B. Fogel, “How to Solve It: Modern Heuristics”, Springer, 2000.

[10] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, “Design Patterns – Elements of Reusable Object-Oriented Software”, Addison Wesley, Reading (Mass.), 1994.