Formal Specification for Evolution of Algorithm and Its Properties*

LUAN Shang-min[1], LI Wei[2]

[1](Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China);

[2](Department of Computer Science and Technology, Beijing University of Aeronautics and Astronautics, Beijing 100083, China)

E-mail: lsm@mail.imd.cims.edu.cn

Received March 29, 1999; accepted April 24, 2000

Abstract: The inductive process is used to specify the evolution of algorithm in this paper. The relationship between a set of sentences of first-order language and an algorithm is established and inductive rules for heuristics are presented. A probabilistic approach to algorithm analysis is also presented. This approach provides a tool for design of efficient algorithm and automatic algorithm design.

Key words: evolution of algorithm; inductive inference; limit

Algorithms play important role in computer science and artificial intelligence. Many problems are NP-complete, i.e. polynomial algorithms for these problems are not available till now. Researchers have designed many heuristics for them. The more efficient they are, the better they are. If a heuristic for a problem does not meet our needs, we will modify it or design a new heuristic to meet our needs. The above process iterates until the heuristic satisfies the given conditions. This can be described by a sequence of versions of algorithms as follows:

$$A_1, A_2, \ldots, A_n, \ldots$$

where, version A_{i+1} is obtained from A_i. This sequence is called the sequence of evolution of algorithms.

In order to describe and implement the above process, two problems need to be considered. First, we should consider how to describe the process of the evolution of algorithms. Second, we should present an operational approach to implement the automatic modification of algorithm. In this paper, we focus on describing the process of evolution of algorithms.

Smith[1,2] showed that the functional specification of an algorithm could be given by algebraic specification, i.e. the function of an algorithm and domain knowledge can be described by the sentences of first-order language, so that an algorithm is equivalent to a set of sentences of first-order language. For a set of sentences of first-order language, we can design an algorithm for it. And for an algorithm, we can give a set of sentences of first-order language for the algorithm. In this paper, we use a set of sentences of first-order language instead of an algorithm, called abstract algorithm. The sequence

* This project is supported by the National Natural Science Foundation of China under Grant No. 19992895 (国家自然科学基金). **LUAN Shang-min** was born in 1968. He received his Ph. D. degree in computer science from Beijing University of Aeronautics and Astronautics in 1999. Now he works as a postdoctor at the Institute of Software, the Chinese Academy of Sciences. His research interests include automatic algorithm design, belief revision, formal specification and human-computer interaction. **LI Wei** was born in 1943. He is a professor of the Department of Computer Science at Beijing University of Aeronautics and Astronautics. And he is an academician of the Chinese Academy of Sciences. His research interests include concurrent programming languages, operational sematics, type theories, and logical foundation of artificial intelligence.

$$\Gamma_1,\Gamma_2,\ldots,\Gamma_n,\ldots$$

is called a sequence of abstract algorithms. The inductive process[5] can be used to describe the process of evolution of algorithms if we give the inductive rules for algorithms, so that we give the inductive rules for heuristics in Section 1.

The heuristics are applied to test cases after they are designed. The comparison between performances of heuristics depends on the experimental data. And the theoretical approach to heuristic analysis has not been discussed until now. In order to solve this problem, we will present a probabilistic approach to heuristic analysis in Section 2. This approach will be illustrated by vertex-cover problem.

1 Inductive Rules for Heuristics

The inductive rules for heuristics will be given in this section.

Definition 1 (problem space). The set of all instances of a problem is called problem space, denoted by T. A nonempty subset of T is called subspace. $d,d_1,d_2,\ldots,d_n,\ldots$ are used to denote subspaces.

Definition 2 (partition). Let T be a problem space and $d_1,d_2,\ldots,d_n,\ldots$ be subspaces. If $T=\bigcup_{i=1}^{\infty}d_i$, $d_1,d_2,\ldots,d_n,\ldots$ is called a partition of problem space T. D is used to denote the partition of problem space, i.e. $D=\{d_1,d_2,\ldots,d_n,\ldots\}$.

A problem space is partitioned into subspaces in order to analyze the heuristics. This will be discussed in Section 2.

F denotes the set of heuristics, and $f,f_1,f_2,\ldots,f_n,\ldots$ are used to denote heuristics. Our first aim is to find the most efficient heuristic among all the heuristics. We need to define the following predicates in order to give inductive rules.

1. $Better(f_1,f_2)$: it means that the performance of heuristic f_1 is better than the performance of heuristic f_2.
2. $Best(f)$: it means that the performance of heuristic f is the best among all the current heuristics.
3. $Success(f,d)$: it means that the heuristic f is acceptable when it is applied to test cases in subspace d.
4. $Win(f,t)$: it means that heuristic f is acceptable when it is applied to test cases t.

The following inductive rules for heuristics are given to meet our first aim.

$$\frac{Better(f_1,f_2)\notin \mathrm{Th}(\Gamma)\ f_1,f_2\in F}{\Gamma\Rightarrow Better(f_1,f_2)\ \Gamma}$$

$$\frac{\neg Better(f_1,f_2)\notin \mathrm{Th}(\Gamma)\ f_1,f_2\in F}{\Gamma\Rightarrow\neg Better(f_1,f_2)\ \Gamma}$$

$$\frac{Best(f_1)\notin \mathrm{Th}(\Gamma)\ f_1\in F}{\Gamma\Rightarrow Best(f_1)\wedge(\forall f(\neg(f=f_1)\rightarrow\neg Better(f,f_1)))\ \Gamma}$$

$$\frac{\neg Best(f)\notin \mathrm{Th}(\Gamma)\ f\in F}{\Gamma\Rightarrow\neg Best(f)\ \Gamma}$$

$$\frac{Success(f,d)\notin \mathrm{TH}(\Gamma)\ f\in F,\ d\in D}{\Gamma\Rightarrow Success(f,d)\ \Gamma}$$

$$\frac{\neg Success(f,d)\notin \mathrm{Th}(\Gamma)\ f\in F\ d\in D}{\Gamma\Rightarrow\neg Success(f,d)\ \Gamma}$$

$$\frac{Win(f,t)\notin \mathrm{Th}(\Gamma)\ f\in F\ t\in T}{\Gamma\Rightarrow Win(f,t)\ \Gamma}$$

$$\frac{\neg Win(f,t)\notin \mathrm{Th}(\Gamma)\ f\in F\ t\in T}{\Gamma\Rightarrow\neg Win(f,t)\ \Gamma}$$

$$\frac{Best(f_1)\in \mathrm{Th}(\Gamma)\ f_1,f_2\in F}{Better(f_2,f_1)\ \Gamma\Rightarrow Better(f_2,f_1)\ Best(f_2)\ \neg Best(f_1)\ \Gamma-\{Best(f_1)\}}$$

In order to obtain a more efficient heuristic, we usually combine the current heuristic with another heuristic to form a new heuristic strategy called combined strategy. In general, the above process is described below. We first design a heuristic for an NP-complete problem. If the heuristic does not work efficiently, we will combine another heuristic with the current heuristic, and obtain a new heuristic. If the new heuristic is more efficient than the previous one, we will use the new heuristic, or we still use the previous one. This process iterates until a heuristic which meets our needs is obtained. Our second aim is to identify which strategy can raise the efficiency of the algorithm for a problem. We first give several predicates which will be used in the remainder of this section, before we give the inductive rules to meet the aim.

1. $add(f)$: it means that the current heuristic is combined with another heuristic and a new heuristic is obtained.

2. $accelerate(f)$: it means that the heuristic obtained by combining the current heuristic with heuristic f is more efficient than the current one.

3. $speedup(f,t)$: it means that the heuristic obtained by combining the current heuristic with heuristic f is more efficient than previous one when they are used to find the solution of instance t.

4. $quicken(f,d)$: it means that if the current heuristic is combined with heuristic f and a new heuristic is obtained, the efficiency of the new heuristic is better than that of the current heuristic when it is applied to test cases in subspace d.

The following inductive rules are given to meet our second aim.

$$\frac{f\in F\ \mathrm{add}(f)\notin \mathrm{Th}(\Gamma)}{\Gamma\Rightarrow \mathrm{add}(f)\ accelerate(f)\ \Gamma}$$

$$\frac{f\in F\ accelerate(f)\notin \mathrm{Th}(\Gamma)}{\Gamma\Rightarrow accelerate(f)\ \Gamma}$$

$$\frac{f\in F\ \neg accelerate(f)\notin \mathrm{Th}(\Gamma)}{\Gamma\Rightarrow \neg accelerate(f)\ \Gamma}$$

$$\frac{f\in F\ t\in T\ speedup(f)\notin \mathrm{Th}(\Gamma)}{\Gamma\Rightarrow speedup(f)\ \Gamma}$$

$$\frac{f\in F\ t\in T\ \neg speedup(f)\notin \mathrm{Th}(\Gamma)}{\Gamma\Rightarrow \neg speedup(f)\ \Gamma}$$

$$\frac{f\in F\ d\in D\ quicken(f,d)\notin \mathrm{Th}(\Gamma)}{\Gamma\Rightarrow quicken(f,d)\ \Gamma}$$

$$\frac{f\in F\ d\in D\ \neg quicken(f,d)\notin \mathrm{Th}(\Gamma)}{\Gamma\Rightarrow \neg quicken(f,d)\ \Gamma}$$

We will discuss how to give a complete instance sequence for the second case. Suppose the heuristic f_i is applied to test cases in subspaces $d_{j_1},d_{j_2},\ldots,d_{j_k}$, and suppose n_m instances in subspace d_{j_m} are tested. Then the complete instance sequence is given as follows.

$add(f_i)$,

$speedup(f_i,t_{j_{11}})$ [or $\neg speedup(f_i,t_{j_{11}})$],

$speedup(f_i,t_{j_{12}})$ [or $\neg speedup(f_i,t_{j_{12}})$],

$\ldots$,

$speedup(f_i,t_{j_{1n_1}})$ [or $\neg speedup(f_i,t_{j_{1n_m}})$], $quicken(f_i,d_{j_2})$ [or $\neg quicken(f_i,d_{j_1})$],

$speedup(f_i,t_{j_{21}})$ [or $\neg speedup(f_i,t_{j_{21}})$], $speedup(f_i,t_{j_{22}})$ [or $\neg speedup(f_i,t_{j_{22}})$],$\ldots$,

$speedup(f_i,t_{j_{2n_2}})$ [or $\neg speedup(f_i,t_{j_{2n_2}})$], $quicken(f_i,d_{j_3})$ [or $\neg quicken(f_i,d_{j_2})$],$\ldots$,

$speedup(f_i,t_{j_{k1}})$ [or $\neg speedup(f_i,t_{j_{k1}})$], $speedup(f_i,t_{j_{k2}})$ [or $\neg speedup(f_i,t_{j_{k2}})$],…,

$speedup(f_i,t_{j_{kn_k}})$ [or $\neg speedup(f_i,t_{j_{2n_k}})$], $quicken(f_i,d_{j_k})$ [or $\neg quicken(f_i,d_{j_1})$],

$accelerate(f_i)$ [or $\neg accelerate(f_i)$].

For heuristics, they are put in the following order: $f_1,f_2,\ldots,f_n,\ldots$. At last, the heuristics which raise the efficiency of the algorithm for a problem remain, all the other heuristics are deleted. Similarly, we can give the complete instance sequence for the first case.

2 A Probabilistic Approach to Algorithm Analysis

In this section, we illustrate how to partition a problem space into subspaces by vertex-cover problem, give a probabilistic approach to algorithm analysis, and discuss how to compare the performances of heuristics within subspace and space.

Vertex-cover problem: Given a graph $G=\langle V,E\rangle$, a vertex-cover of G is a subset V' of V such that at least one of u and v is in V' for each edge (u,v) of G. **Question**: Does G have a vertex-cover V', such that the size of V' is not more than positive integer k? This problem is NP-complete[6].

The problem space of vertex-cover problem can be partitioned as follows: graphs with the same average degree of connectivity are grouped into a subspace. This is similar to that given in Ref. [5]. The difference between our approach and that in Ref. [5] is given below. The approach given in Ref. [5] first partitions the space into subspaces. And the set T of subspaces must be a finite set. Then it partitions each subspace into subdomains. We partition the space into subspaces. A subspace is not partitioned into subdomains and the set T of subspaces is a infinite set.

If we partition the problem space as mentioned before, each subspace is an infinite set, and the partition D is also an infinite set.

$P_{win}(f_i,d_m)$, the probability of win by f_i within subspace d_m, is defined as the probability that true mean of f_i is better than the true mean of each f_j in F. If f_i is applied to test cases in subspace d_m, we have

$$P_{win}(f_i,d_m)=\frac{1}{|s|-1}\sum_{j\neq i}Pr(\mu_i^m>\mu_j^m\mid\hat{\mu}_i^m,\hat{\sigma}_i^m,n_i^m,\hat{\mu}_j^m,\hat{\sigma}_j^m,n_j^m)$$

where $\hat{\mu}_i^m,\hat{\sigma}_i^m,n_i^m,\mu_i^m$ are the sample mean, sample standard deviation, the number of tests, and true mean of f_i in subspace d_m, respectively. $|s|$ is the number of heuristics under consideration.

Suppose that performance values of f are normally distributed, true variance σ_i^2 of f_i is known, and heuristics in a subspace are independent. We also suppose that the sample mean $\hat{\mu}_i$ of heuristics f_i has $N(\mu_i,\sigma_i^2/n_i)$ distribution, then $\hat{\mu}_i-\hat{\mu}_j$ has $N(\mu_i-\mu_j,\sigma_i^{m^2}/n_i+\sigma_i^{m^2}/n_j)$ distribution. $Z=((\hat{\mu}_i-\hat{\mu}_j)-(\mu_i-\mu_j))/\sqrt{\sigma_i^{m^2}/n_i-\sigma_j^{m^2}/n_j}$ has $N(0,1)$ distribution.

By the above hypothesis,

$$Pr(\mu_i^m>\mu_j^m\mid\hat{\mu}_i^m,\hat{\sigma}_i^m,n_i^m,\hat{\mu}_j^m,\hat{\sigma}_j^m,n_j^m)=\Phi((\mu_i^m-\mu_j^m)/\sqrt{\sigma_i^{m^2}/n_i+\sigma_j^{m^2}/n_j})$$

where $\Phi(x)$ is the cumulative distribution function for the $N(0,1)$ distribution.

Because each subspace is an infinite set, so that we can not know the true mean of a heuristic in a subspace. By the law of large number, the sample standard mean $\hat{\mu}$ and sample standard variance $\hat{\sigma}^2$ can be used in place of true mean μ and true variance σ^2, respectively. We have

$$P(\mu_i^m>\mu_j^m\mid\hat{\mu}_i^m,\hat{\sigma}_i^m,n_i^m,\hat{\mu}_j^m,\hat{\sigma}_j^m,n_j^m)=\Phi((\hat{\mu}_i^m-\hat{\mu}_j^m)/\sqrt{\hat{\sigma}_i^{m^2}/n_i+\hat{\sigma}_j^{m^2}/n_j})$$

$Success(f_i,d_m)$ is true if $P_{win}(f_i,d_m)$ is more than a given value, or it is false.

We can not apply heuristic f in each subspace d in D, because the partition D is an infinite set. We can only apply heuristic f in the subspaces in a finite set Π_D which is a subset of partition D. $P_{suc}(f,D)$ is defined as follows.

$$P_{suc}(f,D)=\frac{\sum_{d\in\Pi_D}P_{win}(f,d)}{|\Pi_D|}$$

For two heuristics f_i and $f_j, i\neq j$, $Better(f_i,f_j)$ is true if $P_{suc}(h_i,D)$ is more than $P_{suc}(h_j,D)$, or $Better(f_i,f_j)$ is false.

We can also partition the problem space as follows: graphs with the same size of V are grouped into a subspace. Let d_n consist of the graphs whose size of vertex is n.

If the problem space is partitioned as mentioned before, we can give a probabilistic approach to algorithm analysis. Let x be the time that heuristic f spends in finding a solution for an instance. According to our knowledge, x has normal distribution, i.e. x has $N(\mu,\sigma^2)$ distribution, where μ is the true mean of time that heuristic f spends in finding the solutions for instances in subspace d, σ^2 is the true variance of f in subspace d.

Let $g(i)$ be a function parameterized by i, and i is the size of the problem. For vertex-cover problem, i is the size of V. Let f be a heuristic and d_i be a subspace. $P_{win}(f,d_i)$ is defined below:

$$P_{win}(f,d_i)=Pr(x<g(i))=\int_{-\infty}^{g(i)}\frac{1}{\sqrt{2\pi}\sigma}e^{\frac{-(x-\mu)^2}{2\sigma^2}}=\Phi\left(\frac{g(i)-\mu}{\sigma}\right)$$

Similarly, the subspace d may be an infinite set or the size of d is huge such that we can not apply a heuristic to test all the cases in d. But, by the law of large number, the sample standard mean $\hat{\mu}$ and sample standard variance $\hat{\sigma}^2$ can be used in place of true mean μ and true variance σ^2, respectively. We have

$$P_{win}(f,d_i)=Pr(x<g(i))=\Phi\left(\frac{g(i)-\hat{\mu}}{\hat{\sigma}}\right)$$

Let Π_p be a finite nonempty subset of D. We apply heuristic f to test cases in each subspace in Π_p. $P_{suc}(f,\Pi_p)$ is defined below.

$$P_{suc}(f,\Pi_p)=\frac{\sum_{d\in\Pi_p}P_{win}(f,d)}{|\Pi_p|}$$

Heuristic f is also acceptable if $P_{suc}(f,\Pi_p)$ of f is close to the maximal P_{suc}.

3 Conclusion

We use inductive process[3] to specify the evolution of algorithms in this paper. By the relationship between a set of sentences of first-order language and an algorithm, an algorithm corresponds to a set of sentences of first-order language, and a set of sentences of first-order language corresponds to an algorithm. Inductive rules for heuristics and a probabilistic approach to algorithm analysis are presented. All these are illustrated by vertex-cover problem. This approach provides a tool for design of efficient algorithm and automatic algorithm design. We will give an operational approach to the modification of algorithm in a later paper.

References:

[1] Smith, D.R. Toward a Classification Approach to Design. LNCS 1101, Springer-Verlag, 1996. 62~84.

[2] Peter, P., Smith, D.R. A high-level derivation of global search algorithms (with constraint propagation). Science of Computer Programming, 1997,28:247~271.

[3] Li, W. Inductive process: a logical framework for inductive inference. Science in China, Series A, 1995,38(Supp.):12~27.

[4] Christos, H. Papadimitriou. Computational Complexity. Addison Wesley, San Diego, 1994.

[5] Wah, B. W., Arthur, I. W., *et al*. Genetic-based learning of new heuristics: rational scheduling of experiment. IEEE Transactions on Knowledge and Data Engineering, 1995,7(5):763~785.

算法演化的形式归约及其性质

栾尚敏[1], 李 未[2]

[1](中国科学院 软件研究所,北京 100080);

[2](北京航空航天大学 计算机科学与工程系,北京 100083)

摘要: 使用归纳过程说明算法的演化,建立了一阶语言语句集和算法的关系,并提出了启发式的归纳规则.还提出了算法分析的概率式研究方法.这种方法为有效算法的设计和自动算法设计提供了工具.

关键词: 算法演化;归纳推理;极限

中图法分类号: TP301 **文献标识码**: A