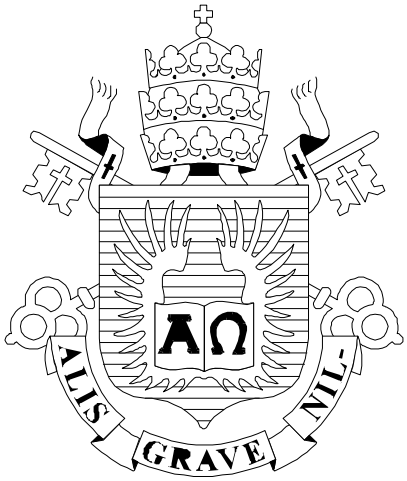

PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 06/05

A Logic-Based Formal Model for (Meta)Heuristics

Fernando Náufel do Amaral
Edward Hermann Haeusler

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 06/05 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena March, 2005

A Logic-Based Formal Model for (Meta)Heuristics *

Fernando Náufel do Amaral
Edward Hermann Haeusler
{fnaufel,hermann}@inf.puc-rio.br

Abstract. We present a structural model for (meta)heuristic search strategies for solving computational problems. The model is defined through the use of topos-theoretical tools and techniques which provide an appropriate internal logic (with the language of Local Set Theory) where objects of interest can be represented.

Keywords: Heuristics, Metaheuristics, Topos Theory, Internal Logic, Local Set Theory.

Resumo. Apresentamos um modelo estrutural para estratégias (meta)heurísticas para resolver problemas computacionais. O modelo é definido através do uso de ferramentas e técnicas da teoria de *topos* que fornecem uma lógica interna apropriada (com a linguagem da Teoria Local dos Conjuntos) onde os objetos de interesse podem ser representados.

Palavras-chave: Heurísticas, Meta-heurísticas, Teoria de *Topos*, Lógica Interna, Teoria Local dos Conjuntos.

---

* This work has been sponsored by the Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

1 Introduction

The main goal of the research reported in this paper is the development of a logic-based formal model for (meta)heuristics general enough to encompass the many (sometimes informal) definitions found in the literature. Among such definitions, we may quote, for example: "Heuristics are criteria, methods, or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal." [1]

According to [2], a metaheuristic is "an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions."

In a more abstract, unified view, both heuristics and metaheuristics are techniques for solving a given problem by first defining some kind of "space" inhabited by candidate answers (which are related by some structure imposed by the definition of such a space) and then defining a strategy for moving in the defined space in search of an appropriate answer. These two aspects — the search space and the search strategy — are the basis for our formal definition of (meta)heuristics.

A general enough formal model for problems and (meta)heuristics can be of great use in the process of choosing, comparing and combining different strategies for solving computational problems of any kind. A *logic-based* model, such as the one presented here, could allow us to represent (meta)heuristics at any desired level of abstraction. The model could provide means for transforming higher-level descriptions of (meta)heuristics into more concrete, detailed representations, much in the same way a software framework is refined and instantiated to generate executable code. Moreover, our logic-based model could allow us to develop a deductive calculus to reason, again at any desired level of abstraction, about (meta)heuristics and their characteristics, as well as the relationships among them.

The development of a formal model of such generality is no light undertaking. In the first place, the theories and tools employed in the definition of the model must be high-level and expressive enough to guarantee completeness, in the sense that our model must be able to represent all (meta)heuristics used in practice. In the second place, the same theories and tools must allow representations of (meta)heuristics to be accurately refined to a more concrete level so as to distinguish between similar techniques, or different variations of a single technique.

It is in this attempt to reconcile abstraction and concreteness that the tools and techniques of Category Theory show their usefulness. More precisely, by using Topos Theory, we will define a universe — i.e., a topos — that comes equipped with a logical language and theory of its own. It is in this logic, by using Local Set Theory, that our logic-based formal model for (meta)heuristics will be developed.

Category Theory [3] was a milestone in the development of a mathematical language for dealing with the universals of a research area. The seminal work relating mathematical results from topology and algebra (culminating in the cohomology concept), the establishment of a foundational alternative for the mathematical discourse, and its widespread use in the various aspects of computational models, helping to elucidate basic questions in concurrency theory, typing discipline, software specification and other areas, are good examples of applications of Category Theory.

Topoi [4] are, in a strong sense, the formal counterparts of sheaves, a geometrical concept that also arises in logical form, e.g., by means of local set theories [5]. Thus, in a single theoretical approach we are able to put together two of the main aspects of (meta)heuristics: the geometry of search spaces and the logic of search strategies. Implementing a (meta)heuristic will then be a matter of coding search spaces as functors (data structures) and search strategies as natural transformations (algorithms).

The main advantages of using Local Set Theory to construct a formal model of (meta) heuristics can be summarized as follows: (1) Category Theory emphasizes the morphisms instead of the objects. This makes for an abstract point of view, where the internal structure of the objects is glimpsed only through the relationships among them. Furthermore, these relationships give rise to categorical constructions where more complex objects are defined in terms of simpler ones; as a result, compositionality is naturally emphasized. (2) Once we define an appropriate topos relating problems and search spaces, we will immediately have at our disposal a logical theory whose model is the defined topos. This eliminates questions of expressiveness, consistency and completeness which would fall upon us if we were to construct our logical theory otherwise. In other words, one of the greatest benefits of Local Set Theory is that the model (i.e. the defined topos) comes equipped with a logical language and theory of its own.

1.1 Related Work

Models, classifications and taxonomies of heuristic techniques have appeared in the literature since the 70's. An example of a well-founded formal approach is the work presented in [6], which seeks to define a grand unifying model for heuristic search, branch-and-bound algorithms and dynamic programming.

The work in [7] features a unified model for tabu search, simulated annealing and genetic algorithms; the presentation, although instructive, is informal and imprecise. Taxonomies like [8] and classifications like [9] tend to be even more imprecise or application-oriented, and cannot be considered real formal models.

More recently, there has been great interest in formal models for search strategies motivated by the need to define and implement software frameworks and object-oriented algorithms for optimization problems, as discussed in [10] and [11]. The models defined in this kind of work are usually in the form of class or component libraries in object-oriented languages or in the form of new modeling languages (or search-oriented extensions of existing languages) like that in [12], possibly coupled with automatic problem solvers. Possible connections between that line of research and the present work are pointed out in our concluding remarks, in Sec. 7.

To our knowledge, this paper represents an entirely original application of Category and Topos Theory to the construction of a formal model for problems, reductions, search spaces and (meta)heuristics.

2 Search Spaces: a Category-Theoretical View

We assume the reader is familiar with basic categorical and topos-theoretical notions such as object, morphism, (contravariant) functor, natural transformation, sub-object classifier and others. [4] is a comprehensive reference on the subject.

2.1 Problems

We define a category with computational problems as objects and problem reductions as morphisms. We base our definitions on the general theory of problems presented in [13].

Definition 2.1 (Problems). *A problem is a triple $P = \langle D, R, p\rangle$, with D and R countable, nonempty sets, and $p \subseteq D \times R$ a relation. Elements $d \in D$ are called* data *or* instances*; elements $r \in R$ are called* results *or* answers*; the relation p is called the* problem condition*; $(d, r) \in p$ means that r is a correct answer for instance d.*

Problems are related to each other through the notion of *reduction*:

Definition 2.2 (Reductions). *Given $P = \langle D, R, p\rangle$ and $P' = \langle D', R', p'\rangle$, a reduction $P \xrightarrow{(\tau,\sigma)} P'$ consists of a pair (τ, σ) of functions computable in polynomial time, with $\tau : D \to D'$ and $\sigma : R' \to R$ such that correct answers are preserved; more precisely, for every $d \in D$ and every $r' \in R'$, we have that $(\tau(d), r') \in p' \;\Rightarrow\; (d, \sigma(r')) \in p$.*

Proposition 2.1 (The category Prob of problems). *Problems as objects and reductions as morphisms form a category* $\mathbf{Prob}$.

An alternative definition of reduction yields a different category. A *binary reduction* is one where σ is an answer-preserving function of the form $\sigma : D \times R' \to R$; i.e., when transforming answers of P' into answers of P, we can use information about the original instance d of P. We have shown elsewhere [14] that $\mathbf{Prob}$ as we have defined it is a reflective subcategory of this alternative category of problems and binary reductions.

It should be noted that $\mathbf{Prob}$ is a *small* category (one whose collection of objects is a set — see [3]). This will prove important at a later point.

2.2 Search Forest Assignments

Part of the definition of a (meta)heuristic strategy to solve a given problem is the construction of the search space. In state space search, this means defining states as well as actions corresponding to arcs (or transitions) between the states; in local search, this means defining neighborhoods; in population-based search, this means defining populations and operators to transform them.

We define a category of functors that assign search spaces to computational problems in a systematic way. The defined category is shown to be a topos, setting the ground for the definition of (meta)heuristics in Local Set Theory, in the internal logic of the topos.

For technical reasons to be justified below, it is not feasible to assign search spaces to *all* problems in $\mathbf{Prob}$ at once in an adequate way. We want to limit the number of reductions between problems; therefore, from now on we will assume that a subcategory $\mathbf{Prob}_0$ of $\mathbf{Prob}$ is defined such that between any two objects P and P' of $\mathbf{Prob}_0$ there is at most one morphism. In technical terms, this amounts to $\mathbf{Prob}_0$ being a thin, skeletal subcategory of $\mathbf{Prob}$. In other words, $\mathbf{Prob}_0$ is a partially ordered set seen as a category.

We will consider a search space for a problem P to be a forest whose nodes are associated with certain information about P. Forests form a category where the morphisms are forest homomorphisms. A forest homomorphism h from a forest F to a forest F' is a

mapping of the nodes of F to the nodes of F' such that parenthood is preserved: if node n_1 is the parent of node n_2 in F, then node $h(n_1)$ is the parent of node $h(n_2)$ in F'.

Given a thin, skeletal category $\mathbf{Prob}_0$ of problems as described above, we assign forests to these problems by means of a contravariant functor from $\mathbf{Prob}_0$ to the category **Forest** of forests:

Definition 2.3 (Search Forest Assignment). *A Search Forest Assignment (SFA) is a functor* $G : \mathbf{Prob}_0^{op} \to \mathbf{Forest}$.

The fact that G is a contravariant functor is indicated by the superscript "^{op}" in $\mathbf{Prob}_0^{op}$. This means that a reduction $P \xrightarrow{(\tau,\sigma)} P'$ is mapped to a forest homomorphism $GP' \xrightarrow{G(\tau,\sigma)} GP$, in the opposite direction. This corresponds to the intuition that, as P reduces to P' (and answers to P' can be transformed into answers to P), a traversal of the search space of P' can be transformed into a traversal of the search space of P.

As each SFA is a functor, the collection of all SFA's can be structured as a functor category:

Definition 2.4 (The category SFA of Search Forest Assignments). SFA *is the functor category* $\mathbf{Forest}^{\mathbf{Prob}_0^{op}}$

An object G of **SFA** assigns a search forest GP to each problem P in $\mathbf{Prob}_0$. The assigned forest GP has unlabeled nodes and edges. Ultimately, we will want to label the nodes with information concerning the answers of P. For now, however, we may postpone considerations about labelings to appreciate the fact that we already have a topos:

Theorem 2.1. *The category* **SFA** *is a topos.*

Proof. The category **SFA** was defined as $\mathbf{Forest}^{\mathbf{Prob}_0^{op}}$. However, the category **Forest** of forests is isomorphic to the functor category $\mathbf{Set}^{\omega^{op}}$, where ω^{op} is the infinite total order $0 \leftarrow 1 \leftarrow 2 \leftarrow \cdots$ seen as a category. So **SFA** is isomorphic to the functor category $(\mathbf{Set}^{\omega^{op}})^{\mathbf{Prob}_0^{op}}$, which, in turn, is isomorphic to the functor category $\mathbf{Set}^{\omega^{op} \times \mathbf{Prob}_0^{op}}$. Now, any functor category of the form $\mathbf{Set}^{\mathbf{C}}$ — with $\mathbf{C}$ a small category — is a topos (see, e.g., [4]). As $\omega^{op} \times \mathbf{Prob}_0^{op}$ is small (as noted in Sec. 2.1), this yields the desired result. □

Given a problem $P = \langle D, R, p \rangle$ and an SFA G, we want the forest GP to be labeled with information concerning the answers of P. More precisely, we want to label each node n of GP with a set of answers $\lambda(n) \subseteq R$. This is justified by the following:

- In population-based metaheuristics, each node in the search space corresponds to a population, i.e., a set of answers;
- In local search metaheuristics and transformation heuristics, each node corresponds to a single answer r, which can be seen as the singleton $\{r\}$;
- In constructive heuristics, each node corresponds to a "partial" answer, and moving from one node to another corresponds to adding "elements" to the partial answer in order to build a complete answer. One example would be to try to solve the Traveling Salesman Problem (TSP – see [1], e.g.) by picking one initial city and subsequently adding one city at a time, constructing a complete tour in an incremental

fashion. In [6], e.g., it is shown that a partial answer of this kind can be seen as (or rather represented by) a set of answers, a point of view which provides an interesting connection between generate-and-test methods of AI and split-and-prune methods of Operations Research. In the example of the TSP, a partial tour t can be represented by the set of all complete tours having t as a prefix.

So we want SFAs to work as follows: to each problem $P = \langle D, R, p\rangle$ in $\mathbf{Prob}_0$, the SFA G will assign a forest GP whose nodes are labeled by sets of answers in R. Furthermore, given a reduction $P \xrightarrow{(\tau,\sigma)} P'$, the SFA G will assign a labeled forest homomorphism $GP' \xrightarrow{G(\tau,\sigma)} GP$ with the added constraint that a node n' of GP' labeled by a set A' of answers must be mapped to a node $G(\tau,\sigma)(n')$ of GP labeled by the set $\sigma(A') = \{\sigma(r') \mid r' \in A'\}$.

This arrangement corresponds to the intuition that finding the answer r' in the search space for P' implies finding the answer $\sigma(r')$ in the search space for P, which reduces to P'. Formally, this can be achieved by defining an SFA to be a special kind of functor from $\mathbf{Prob}_0$ to the category **LForest** of labeled forests (whose morphisms preserve the node labels as desired). However, a more elegant alternative presents itself: we can label the forests assigned to problems by means of a categorical construction involving a specific object of the category **SFA**.

Definition 2.5 (The L functor). $L : \mathbf{Prob}_0 \to \mathbf{Forest}$ *is the object of* **SFA** *that maps each problem $P = \langle D, R, p\rangle$ to the forest LP with infinitely many levels, whose set of nodes at level 0 is $\wp(R)$ (the powerset of R), and whose set of nodes at level i, for $i > 0$, is $\wp(R)^{i+1}$, the set of all $(i+1)$-tuples whose components are sets of answers in R.*

As for morphisms, L maps a reduction $P \xrightarrow{(\tau,\sigma)} P'$ to the forest homomorphism $LP' \xrightarrow{L(\tau,\sigma)} LP$ such that a node $(A'_1, \ldots, A'_i)$ at the i-th level of LP' is mapped to the node $(\sigma(A'_1), \ldots, \sigma(A'_i))$ at the i-th level of LP.

Proposition 2.2. *Given a forest F and a problem $P = \langle D, R, p\rangle$, a forest homomorphism $F \xrightarrow{\lambda} LP$ will label the nodes of F with sets of answers in R.*

Proof. A node n at the i-th level of F with $\lambda(n) = (A_1, \ldots, A_i)$ is considered to be labeled by the set A_i. The other components $A_1, \ldots, A_{i-i}$ store the labels of the ancestor nodes of n from the root down to the parent of n. □

Furthermore, given an SFA G, if the labeling must occur consistently for all search forests assigned by G, then a natural transformation $\lambda : G \xrightarrow{\cdot} L$ will do. The main consequence of these considerations is the fact that, if we want our SFAs to assign *labeled* forests to problems, it suffices to take pairs $\langle G, \lambda\rangle$, with G an SFA and λ a natural transformation from G to L, as shown in Fig. 1.

It turns out that pairs of the form $\langle G, \lambda\rangle$ as above form a category themselves, called the *slice category* $\mathbf{SFA} \downarrow L$. What's more, by the Fundamental Theorem of Topoi [4], a slice category of any topos is again a topos. So we find ourselves with two topoi at our disposal: (1) **SFA**, the topos whose objects are SFAs (functors assigning unlabeled forests to problems); and (2) $\mathbf{SFA} \downarrow L$, the topos whose objects are pairs of the form $\langle G, \lambda\rangle$, with G an object of **SFA** and λ a collection of forest homomorphisms giving a labeling of the forests assigned by G.

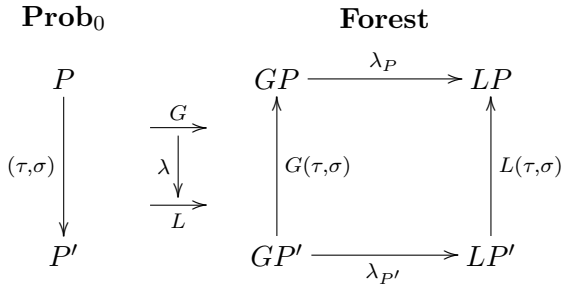

Figure 1: Labeling search forests via a natural transformation $\lambda : G \stackrel{\cdot}{\rightarrow} L$

Both topoi have internal logics which are suitable for specifying (meta)heuristics. For the development of our model, we choose to work with the logic of **SFA** for the simple reason that objects and morphisms of the slice topos $\mathbf{SFA} \downarrow L$ can be easily referenced in the language of **SFA**, as always happens with slice categories of topoi (see [5, pp. 131ff]).

3 Local Set Theory and the Internal Logic of SFA

Any topos can be seen as a model of some local set theory (LST). In LST, the notion of set is replaced by that of *type*. In the language of LST each term (including those representing sets) has an associated type. The "local" in LST means that some common set-theoretical operations, such as union, intersection etc., are only defined for terms of the same type (i.e., locally). Apart from that, the language is very similar to that of set theory, with primitive symbols $=$, $\in$ and $\{\,|\,\}$. According to [5, p. 68ff], the language of LST is defined as follows:

Definition 3.1 (Local language). *A local language $\mathcal{L}$ is determined by the following components:*

- *Symbols: the unit symbol* $\mathbf{1}$*, the truth-value type symbol* $\mathbf{\Omega}$*, ground type symbols* $\mathbf{A}, \mathbf{B}, \mathbf{C}, \ldots$*, and function symbols* $\mathbf{f}, \mathbf{g}, \mathbf{h}, \ldots$*;*
- *Types: the set of types of $\mathcal{L}$ is the least set $\mathcal{T}$ containing $\mathbf{1}, \mathbf{\Omega}$, all ground type symbols $\mathbf{A}, \mathbf{B}, \mathbf{C}, \ldots$ and closed under the following operations:*
  - *For $\mathbf{A} \in \mathcal{T}$, the power type $\mathbf{PA}$ is also in $\mathcal{T}$;*
  - *For $\mathbf{A_1}, \ldots, \mathbf{A_n} \in \mathcal{T}$, the product type $\mathbf{A_1} \times \cdots \times \mathbf{A_n}$ is also in $\mathcal{T}$ (for $n = 0$, the product type is $\mathbf{1}$).*
- *Signatures: Each function symbol $\mathbf{f}$ is associated to a signature $\mathbf{A} \rightarrow \mathbf{B}$, where $\mathbf{A}$ and $\mathbf{B}$ are types. This is denoted by $\mathbf{f} : \mathbf{A} \rightarrow \mathbf{B}$;*
- *Variables: For each type $\mathbf{A}$ there is a countable set of variables $V_{\mathbf{A}}$;*
- *Terms: For each type $\mathbf{A}$, there is a set $T_{\mathbf{A}}$ of terms of type $\mathbf{A}$, defined as follows:*
  - $\star \in T_{\mathbf{1}}$;
  - $V_{\mathbf{A}} \subseteq T_{\mathbf{A}}$;

- *For* $\mathbf{f} : \mathbf{A} \to \mathbf{B}$ *and* $\tau \in T_{\mathbf{A}}$*, we have that* $\mathbf{f}(\tau) \in T_{\mathbf{B}}$*;*
- *For* $\tau_i \in T_{\mathbf{A}_i} (i = 1, \ldots, n)$*, we have that* $(\tau_1, \ldots, \tau_n) \in T_{\mathbf{A_1} \times \cdots \times \mathbf{A_n}}$*. For* $n = 0$*, this term is* $\star$*;*
- *For* $\tau \in T_{\mathbf{A_1} \times \cdots \times \mathbf{A_n}}$*, we have that* $\pi_i(\tau) \in T_{\mathbf{A}_i}$ *(with* $i = 1, \ldots, n$*);*
- *For* $\varphi \in T_{\mathbf{\Omega}}$ *e* $x \in V_{\mathbf{A}}$*, we have that* $\{x \mid \varphi\} \in T_{\mathbf{PA}}$*;*
- *For terms* σ *and* τ *of type* $\mathbf{A}$*, we have that* $\sigma = \tau$ *is a term in* $T_{\mathbf{\Omega}}$*;*
- *For terms* σ *and* τ *of types* $\mathbf{A}$ *and* $\mathbf{PA}$*, respectively, we have that* $\sigma \in \tau$ *is a term in* $T_{\mathbf{\Omega}}$*.*

Terms of type $\mathbf{\Omega}$ are called formulae. Free and bound occurrences of variables are defined in the usual fashion. Logical operators are defined as abbreviations, as shown in [5, p. 70]. For example, $\top$ is defined as $\star = \star$; given formulae φ, ψ, we have that $\varphi \wedge \psi$ is defined as $(\varphi, \psi) = (\top, \top)$, and $\varphi \Rightarrow \psi$ is defined as $(\varphi \wedge \psi) = \varphi$. For an example involving a quantifier: given a variable x of the appropriate type, $\forall x : \varphi$ is defined as an abbreviation of $\{x \mid \varphi\} = \{x \mid \top\}$.

Some terms in a local language will represent "sets" (in a topos, each object can be seen as a set). Thus, the usual set-theoretical operations and entities such as union, intersection, disjoint union, powersets, sets of "functions" etc. are also defined as abbreviations. The details can be found in [5, pp. 83ff]. There, it is also shown how a local language can be interpreted in an arbitrary topos.

As explained in Sec. 2.2, our model presupposes a thin, skeletal subcategory $\mathbf{Prob}_0$ of $\mathbf{Prob}$. Due to space limitations, we will simplify our exposition here by considering $\mathbf{Prob}_0 = \bullet$; i.e., the domain of all our SFAs (functors $G : \mathbf{Prob}_0^{op} \to \mathbf{Forest}$) is a category formed by one single problem. A more thorough treatment, with $\mathbf{Prob}_0$ an arbitrary thin, skeletal subcategory of $\mathbf{Prob}$, is found in [14].

An interesting consequence of this simplification is that $\mathbf{SFA}$ $(= \mathbf{Forest}^{\bullet^{op}})$ becomes the category $\mathbf{Forest}$ itself. So, in what follows, we will be using the internal logic of the topos of forests to specify (meta)heuristics for one single problem.

Like all topoi of the form $\mathbf{Set}^{\mathbf{C}}$, the topos $\mathbf{SFA}$ has a natural number object (NNO), which allows for the definition, in the theory of $\mathbf{SFA}$, of many sets and structures that are found in everyday mathematics. For example, there is an object Q representing the rational numbers; elements of this object will be used below to express discrete probabilities in the definition of stochastic search strategies.

4 Stochastic Search Strategies in Local Set Theory

Given a forest G (i.e., an object of $\mathbf{SFA}$), we will represent the behavior of a stochastic search strategy by a sequence $\{S_i\}_{i \in \mathbb{N}}$ where each S_i is a set of triples of the form $\langle T, v, q \rangle$, with T a subforest[1] of G corresponding to the forest formed by all nodes visited up to step i, with v the node last visited, and with q a probability. The sequence $\{S_i\}_{i \in \mathbb{N}}$ is determined by the set S_0 and by a function $step$ from the set of all such sets of triples to itself such that $step(S_i) = S_{i+1}$ for all $i > 0$. The sequence $\{S_i\}_{i \in \mathbb{N}}$ must be such that the

[1]Here and in what follows, by a "subforest of G" we mean a subobject of G, i.e., a forest that can be embedded in the forest G by a forest homomorphism. The roots of the subforest must be at the same level as the roots of the forest.

set S_0 contains only triples where T is a single-node subforest of G and where v is that single node. This means that S_0 is actually a collection of possible initial nodes of the search, each node accompanied by a probability. Moreover, for each triple $\langle T', v', q' \rangle$ in S_{i+1} there must exist some triple $\langle T, v, q \rangle$ in S_i such that either $T = T'$, meaning that no new node is visited at step $i + 1$, or T' is an extension of T by exactly the one node v'. Finally, obvious conditions on the values of the probabilities must also hold.

Recall from Sec. 2.2, however, that we intend to describe objects of the slice topos $\mathbf{SFA} \downarrow L$ in the language of **SFA**: the forests of **SFA** must have their nodes labeled by sets of answers of the problem in question (something that is achieved by morphisms $G \xrightarrow{\lambda} L$), so we must consider pairs of the form $\langle G, \lambda \rangle$ and $\langle T, \theta \rangle$ instead of forests G and subforests T. Furthermore, given $\langle G, \lambda \rangle$ and $\langle G', \lambda' \rangle$, we have that a morphism $\langle G, \lambda \rangle \xrightarrow{\alpha} \langle G', \lambda' \rangle$ can only be considered if α preserves the labels of G. This is expressed by the predicate[2]

$$\mathbf{isLabelPreserving}(\alpha, G, \lambda, G', \lambda') \iff \forall x \in G : \lambda(x) = \lambda'(\alpha(x))$$

Then, $\langle H, \kappa \rangle$ is a subobject of $\langle G, \lambda \rangle$ iff the following predicate is satisfied (where $\mathbf{isMonic}(\alpha)$ is satisfied iff α is a monomorphism — see [5]):

$$\begin{array}{l} \mathbf{isSubObject}(H, \kappa, G, \lambda) \iff \\ \quad \exists \alpha \in G^H : (\mathbf{isMonic}(\alpha) \wedge \mathbf{isLabelPreserving}(\alpha, H, \kappa, G, \lambda)) \end{array}$$

From now on, in order to make the formulae more readable, the labeling morphisms (e.g., λ, κ, etc.) and the condition that a morphism α must be label-preserving will be omitted, except where they must be explicitly mentioned. So, labeled forests and subforests will simply be denoted by G, H, T, etc.

The set of all subforests of a given labeled forest G will be denoted PG.

A node H of a labeled forest G is identified with the finite path from the root to the node. In other words, a node H is a nonempty, finite, linear subobject of the forest. Nonemptiness corresponds to H being different from the initial object $\varnothing$; finiteness corresponds to the unique morphism 1_H from H to 1 not being epic; linearity corresponds to 1_H being monic:

$$\begin{array}{l} \mathbf{isNode}(H, \kappa, G, \lambda) \iff \\ \quad \mathbf{isSubObject}(H, \kappa, G, \lambda) \wedge H \neq \varnothing_G \wedge \mathbf{isMonic}(1_H) \wedge \neg \mathbf{isEpic}(1_H) \end{array}$$

Recall that L is the codomain of the morphisms responsible for the labeling of nodes. Then the set of all nodes of a labeled forest $\langle G, \lambda \rangle$ is represented by the following set-term, which we will abbreviate by $nodes(G, \lambda)$:

$$nodes(G, \lambda) = \{ \langle H, \kappa \rangle \in \coprod_{H \in PG} L^H \mid \mathbf{isNode}(H, \kappa, G, \lambda) \}$$

Again, we will omit labeling morphisms and write $nodes(G)$. Clearly, we have that $nodes(G) \subseteq PG$.

A root of a labeled forest G is an element of $nodes(G)$ that is minimal with respect to the partial order "is a subobject of":

$$\mathbf{isRoot}(H, G) \iff H \in nodes(G) \wedge \neg \exists H' \in nodes(G) : \mathbf{isSubObject}(H', H)$$

[2] As seen in Sec. 3, the language of LST is typed; here, in order to unclutter the notation, we omit typing information about about all terms whose type can be aprehended from the context.

The fact that a labeled forest G extends another forest G' by exactly one node is represented by the following predicate:

$$\begin{aligned}&\mathbf{extendsByOne}(G, G') \iff \\ &\quad \mathbf{isSubObject}(G', G) \ \wedge \ \exists! H \in nodes(G) : H \notin nodes(G')\end{aligned}$$

And the following predicate states that the node given by H is the one that was added to G' to yield G:

$$\begin{aligned}&\mathbf{wasAdded}(H, G, G') \iff \\ &\quad \mathbf{extendsByOne}(G, G') \ \wedge \ H \in nodes(G) \ \wedge \ H \notin nodes(G')\end{aligned}$$

In our definition of stochastic search strategies, we want to consider only finite, nonempty sets of triples of the form $\langle T, H, q\rangle$ such that the sum of all probabilities q equals 1. To this end, let C be the set

$$C = \left(\coprod_{T \in PG} nodes(T) \ \times \ \{q \in Q \mid 0 < q \leq 1\} \right)$$

where Q is the object of rational numbers[3]. The elements of C are triples of the form $\langle T, H, q\rangle$ with q a probability. Now define S to be the set[4]

$$S = \left\{ X \in PC \;\middle|\; X \neq \varnothing, \ X \text{ finite}, \ \sum_{(x,y,q) \in X} q \ = 1 \right\}$$

The functor $R : \mathbf{Prob}_0^{op} \to \mathbf{Forest}$ maps each problem $P = \langle D, R, p\rangle$ to a specific forest representing the set R of answers of P. In specifying a search strategy, we must define how answers are to be returned; to this end, define the set A as

$$A = \left\{ X \in P(\ R \times \{q \in Q \mid 0 < q \leq 1\}\ ) \ \middle| \ \sum_{(x,q) \in X} q \ = 1 \right\}$$

Then a morphism $answer : N \to A$ from the natural number object N determines the answers returned by the strategy if the search terminates at the nth iteration. Note that each answer r is accompanied by a probability value q.

Now we may give a definition of a stochastic search strategy in LST:

Definition 4.1 (Stochastic search strategy). *A stochastic search strategy over a forest G is represented by a term $\langle init, step, answer\rangle$ with $init$ a term of type $\mathbf{S}$, $step$ a term of type $\mathbf{S}^{\mathbf{S}}$ and $answer$ a term of type $\mathbf{A}^{\mathbf{N}}$ satisfying the following predicate:*

$$\begin{aligned}&\mathbf{isStochasticSearchStrategy}(init, step, answer, G) \iff \\ &\quad \forall \langle T, H, q\rangle \in init : (T = H \ \wedge \ \mathbf{isRoot}(H, G)) \wedge \\ &\quad \forall X, Y \in S : (step(X) = Y \ \Rightarrow \\ &\qquad \forall \langle T, H, q\rangle \in Y : \exists \langle T', H', q'\rangle \in X : \\ &\qquad\quad (T = T' \ \vee \ (\mathbf{extendsByOne}(T, T') \ \wedge \ \mathbf{wasAdded}(H, T, T'))) \\ &\quad)\end{aligned}$$

[3] Q is defined in any nontrivial topos.

[4] For space limitations, we do not include the definitions of the "finiteness" predicate or of the summation term. See [14].

5 Examples

To illustrate the capabilities of the model, we will specify some search strategies in the language of LST. First, however, we define two additional useful terms.

Many heuristics are search strategies where the nodes of the search space are evaluated according to some heuristic function. In our language, a heuristic function to evaluate the nodes of a given labeled forest G can be defined as a term h of type $\mathbf{N}^{nodes(G)}$, where N is the natural number object. In other words, the "grade" a node receives upon evaluation is a natural number. We assume that the lower the grade, the better the evaluation.

For every stochastic search strategy $\langle init, step, answer\rangle$ we may define by simple recursion (see [5]) a term $stage$ of type $\mathbf{S}^{\mathbf{N}}$, where N is the natural number object. The idea is that, given any natural number n, the term $stage(n)$ will be equivalent to the term $step(step(\cdots step(init)\cdots))$, with n applications of $step$.

5.1 Greedy Search

In a simple greedy search strategy, the node visited at each stage after the first is the best child of the node visited in the previous stage, as long as the best child has a better (or equal) evaluation than the node visited in the previous stage. Nothing is assumed about the initial node.

As simple greedy search is deterministic rather than stochastic, for each n the set $stage(n)$ is a singleton. To simplify the formulae, for all n we consider $stage(n)$ to be a pair of the form $\langle T, H\rangle$, with T representing the labeled forest (a tree, actually) formed by the nodes visited so far and H the last node visited.

A search strategy $\langle init, step, answer\rangle$ is considered greedy if it satisfies the following predicate for some interpretation of h (the heuristic function):

$$
\begin{array}{ll}
 & isGreedy(init, step, answer, G, h) \iff \\
1 & \mathbf{isStochasticSearchStrategy}(init, step, answer, G) \wedge \\
2 & \forall T, T', H, H' : (\\
3 & \quad step(T, H) = (T', H') \iff (\\
4 & \quad\quad H' = H \ \wedge\ \forall K \in children(H, G) : h(K) > h(H) \\
5 & \quad\quad \vee \\
6 & \quad\quad h(H') \leq h(H) \ \wedge\ \forall H'' \in children(H, G) : h(H') \leq h(H'') \\
7 & \quad) \\
8 &)
\end{array}
$$

Here, $children(H)$ is a term denoting the set of children[5] of node H in forest G. Line 4 specifies that the search terminates (i.e., the current node is revisited indefinitely) when the current node H has no better children. Line 6 says that otherwise the next node to be visited is the best child of the current node.

[5] A node H' is a child of a node H iff $\mathbf{extendsByOne}(H', H)$ is satisfied.

1. $current \leftarrow initial$
2. $n \leftarrow 0$
3. $T \leftarrow sched(n)$
4. If $T = 0$ then return $current$
5. $next \leftarrow$ child of $current$ chosen randomly
6. $\Delta E \leftarrow h(next) - h(current)$
7. If $\Delta E \leq 0$ then $current \leftarrow next$
   else $current \leftarrow next$ with probability $\exp(-\Delta E/T)$
8. $n \leftarrow n + 1$
9. Go to 3

Figure 2: Simulated Annealing Algorithm

5.2 Simulated Annealing

The algorithm in Fig. 2 describes *simulated annealing* [15], a metaheuristic frequently used to solve combinatory optimization problems:

At each stage, a child of the current node is chosen at random; the chosen child may be visited with a probability that depends both on the difference between the grade of the current node and the grade of the chosen child (ΔE) and on the value of a parameter T, which, by analogy with a physical process, is called "temperature". At each stage n, the value of T is given by a function $sched(n)$. The idea is to define this function in such a way that it will occasionally happen that a child that is worse than its parent is visited, so as to escape local optima. In the algorithm, $current$, $initial$ and $next$ are nodes of the search space.

A stochastic search strategy $\langle init, step, answer \rangle$ is an example of simulated annealing over a forest G, using heuristic function h and cooling schedule $sched$, precisely when it satisfies the following predicate:

$$
\begin{array}{ll}
 & isSimAnn(init, step, answer, G, h, sched) \iff \\
1 & \mathbf{isStochasticSearchStrategy}(init, step, answer, G) \wedge \\
2 & \forall n : sched(sn) \leq sched(n) \wedge \\
3 & \forall X, Y, n : (\\
4 & \quad X = stage(n) \wedge Y = stage(sn) \Rightarrow (\\
5 & \quad\quad (sched(n) = 0 \iff X = Y) \wedge \\
6 & \quad\quad (sched(n) > 0 \iff Y = gatherTriples(expandSet(X))) \\
7 & \quad) \\
8 &)
\end{array}
$$

Line 2 states that $sched$ is a nonincreasing function (sn represents the successor of n); line 5 establishes the termination criterion: when $sched(n)$ becomes 0, the strategy makes

no more progress; line 6 describes one iteration, using terms that are defined as follows:[6]

$$expandSet(X) = \{\ expandTriple(\langle T, H, q\rangle) \mid \langle T, H, q\rangle \in X\ \}$$

$$expandTriple(\langle T, H, q\rangle) = \{\ \langle T, H, q \cdot pNoAdvance(H, sched(n))\rangle\ \} \cup \bigcup_{H' \in children(H)} \{\ \langle T', H', q \cdot pVisit(H', H, sched(n))\rangle \mid \mathbf{wasAdded}(H', T, T')\ \}$$

$$gatherTriples(Z) = \{\ \langle T, H, s\rangle \mid \exists z \in Z : \langle T, H, q\rangle \in z\ \wedge\ s = \sum_{\substack{z \in Z \\ \langle T,H,q\rangle \in Z}} q\ \}$$

$$pNoAdvance(H, t) = \frac{|worseChildren(H)| - \sum_{H' \in worseChildren(H)} e^{(h(H)-h(H'))/t}}{|children(H)|}$$

$$worseChildren(H) = \{\ H' \in children(H) \mid h(H') > h(H)\ \}$$

$$pVisit(H', H, t) = \begin{cases} e^{(h(H)-h(H'))/t}\ /\ children(H) & \text{if } h(H') > h(H) \\ 1\ /\ children(H) & \text{if } h(H') \leq h(H) \end{cases}$$

In $pNoAdvance$ (a term denoting the probability that no child of the current node is visited) and $pVisit$ (a term denoting the probability that a given child of the current node is visited), e represents a rational approximation of the real constant e.

6 Verifying Properties of Search Strategies

By using the sound and complete sequent calculus for Local Set Theory defined in [5], one can prove properties of the search strategies specified in our model. We offer below some brief comments on examples of provable formulae involving greedy search and simulated annealing. The properties are quite simple and intuitive, and are included here only for illustrative purposes. The proofs themselves are not presented, and although a bit lengthy, can be easily constructed.

- The provable formula that says that simple greedy search is deterministic (i.e., not stochastic) asserts that each set $stage(n)$ is a singleton:

$$\forall init, step, answer, G, h : \mathbf{isGreedy}(init, step, answer, G, h)\ \Rightarrow \forall n : \exists! \langle T, H\rangle : \langle T, H, 1\rangle \in stage(n)$$

- Furthermore, simple greedy search never backtracks. Equivalently, at each iteration, the subforest consisting of all nodes visited by the search is actually a linear tree. This is expressed by the provable formula (recall that a node is identified with a path from the root to the node; i.e., a finite linear tree)

$$\forall init, step, answer, G, h : \mathbf{isGreedy}(init, step, answer, G, h)\ \Rightarrow \forall n : \forall \langle T, H, q\rangle \in stage(n) : \mathbf{isNode}(T)$$

- Actually, an analogous formula for simulated annealing is also provable.

[6] Due to space limitations, not all set-theoretical terms and operations used here (e.g. definition by cases) have had their definitions included in this paper.

- Although not shown in the paper, one can specify in the logic properties of the instance d of the problem $P = \langle D, R, p \rangle$ being solved and of the search forest G assigned to it. Once this is done, one can compare the performances of simple greedy search and simulated annealing in solving the problem in question. For example, the following formula states that (for the given problem P and in the given search forest G, using a single heuristics function h) the simple greedy search strategy $\langle init, step, answer \rangle$ (with associated function $stage$) fails in finding a correct answer, whereas the simulated annealing strategy $\langle init', step', answer' \rangle$ (with associated function $stage'$) has a better than 50% chance of succeeding:

$$\begin{array}{l} \forall n : ((stage(sn) = stage(n) \;\Rightarrow\; \forall r : (\langle r, 1\rangle \in answer(n) \Rightarrow \neg correct(r))) \\ \quad \wedge\; (stage'(sn) = stage'(n) \;\Rightarrow \\ \qquad\quad \exists r \exists q : (\langle r, q\rangle \in answer'(n) \wedge correct'(r) \wedge q \geq 1/2)) \\ ) \end{array}$$

  Alternatively, the model of the problem P and the search forest G could be defined outside our logical language, and verification of the above formula be conducted through a model-checking procedure.

The implementation of theorem-proving and model-checking techniques for Local Set Theory is work in progress. As soon as this is completed, we will be able to present examples of the verification of more complex, real-life properties of search spaces and search strategies.

7 Conclusion

We believe that the most important achievements of the present work so far have been the successful use of topos-theoretical tools to define our structural model and the generation of examples to provide evidence to the effect that our model is indeed comprehensive enough to represent the techniques used by practitioners. In fact, [14] discusses further examples of specifications of (meta)heuristics, including the well-known paradigm of Genetic Algorithms.

As far as we know, (meta)heuristics are built from search spaces and search strategies. The former are functors, and the latter are natural transformations, in a strong sense. Taking this into account, one cannot escape our topos-theoretical view of (meta)heuristics, and we may say that this view provides a complete model. Besides, this model is made formal by means of adequate manipulations on the respective local set theories, yielding means of proving properties, besides correctness, of each of the (meta)heuristics that one can conceive. This goes beyond modeling languages like the one presented in [12] (which usually consist of formalisms whose main purpose is the expression of algorithms) in the sense that our formal model also supports property-checking.

Apart from the construction of a "library" of specifications of (meta)heuristics and the definition of a formal taxonomy of strategies, we envisage other applications of the present work: (1) Logical specifications can be used to check the correctness of more concrete representations of (meta)heuristics, possibly even in the form of code in a programming language or in the form of code in a modeling language like that presented in [12]. To this end, the study and implementation of theorem-proving and/or model-checking techniques for Local Set Theory would certainly lead to interesting and useful

results; (2) the language of our model can be used to define a high-level software framework, which would be refined through the use of techniques of code transformation to generate modules in some programming language; these generated modules would then be combined with a fixed library of supporting modules, written in the programming language, to generate complete implementations of (meta)heuristics, profiting from the code reuse advantages offered by software frameworks. An interesting point is that this setting would be flexible enough to allow the instantiation of the framework to take place either early in the generation (i.e., at the logical level) or in the final stages (i.e., at the programming language level), or at various points in the process.

References

[1] PEARL, J.. **Heuristics: Intelligent Search Strategies for Computer Problem Solving**. The Addison-Wesley Series in Artificial Intelligence. Addison-Wesley, 1985.

[2] VOSS, S.. **Meta-heuristics: The state of the art**. In: PROCEEDINGS OF THE WORKSHOP ON LOCAL SEARCH FOR PLANNING AND SCHEDULING-REVISED PAPERS, p. 1–23. Springer-Verlag, 2001.

[3] MAC LANE, S.. **Categories for the Working Mathematician**. Springer-Verlag, 1971.

[4] GOLDBLATT, R.. **Topoi – the Categorial Analysis of Logic**. North Holland, 1979.

[5] BELL, J. L.. **Toposes and Local Set Theories, an Introduction**. Oxford University Press, 1988.

[6] KUMAR, V.. **A general heuristic bottom-up procedure for searching AND/OR graphs**. Information Sciences, 56(1–3):39–57, 1991.

[7] RAYWARD-SMITH, V. J.. **A unified approach to tabu search, simulated annealing and genetic algorithms**. In: Rayward-Smith, V. J., editor, APPLICATIONS OF MODERN HEURISTIC METHODS, p. 17–38. Alfred Waller Limited, Henley-on-Thames, UK, 1995.

[8] TALBI, E.-G.. **A taxonomy of hybrid metaheuristics**. Journal of Heuristics, 8:541–564, 2002.

[9] ZANAKIS, S. H.; EVANS, J. R. ; VAZACOPOULOS, A. A.. **Heuristic methods and applications: A categorized survey**. European Journal of Operational Research, 43:88–110, 1989.

[10] FINK, A.; VOSS, S. ; WOODRUFF, D.. **Metaheuristic class libraries**. In Glover and Kochenberger [15].

[11] Voss, S.; Woodruff, D. L., editors. **Optimization Software Class Libraries**. Kluwer, 2002.

[12] VAN HENTENRYCK, P.; MICHEL, L.. **The modeling language OPL: A short overview**. In Voss and Woodruff [11].

[13] VELOSO, P. A. S.; VELOSO, S. R. M.. **Problem decomposition and reduction: Applicability, soundness, completeness**. In: Trappl, R., editor, PROGRESS IN CYBERNETICS AND SYSTEMS RESEARCH, volume 8, p. 199–203. Hemisphere Publ. Co., 1981.

[14] AMARAL, F. N.. **Teoria de Modelos para Heurísticas Baseada em *Topoi***. PhD Thesis, Depto. de Informática – PUC-Rio – Brazil, 2004.

[15] Glover, F.; Kochenberger, G., editors. **Handbook of Metaheuristics**. Kluwer, 2002.