Synthesis of Local Search Algorithms by Algebraic Means

Robert P. Graham, Jr. and Paul D. Bailor
Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433

Abstract

Algebraic techniques have been applied successfully to algorithm synthesis by the use of algorithm theories and design tactics, an approach pioneered in the KIDS system but described and implemented only partly in algebraic terms. Local search is an effective and popular algorithmic approach to solving optimization problems, so that a formal characterization of this family is desirable. This paper describes a more purely algebraic approach to algorithm design and its application to the synthesis of local search algorithms.

1 Introduction

Algebraic techniques have been applied successfully to algorithm synthesis by the use of algorithm theories and design tactics, an approach pioneered in the KIDS system [1, 2]. An algorithm theory is an algebraic specification that formally characterizes the essential components of a family of algorithms, such as global search or divide-and-conquer. A design tactic is a specialized procedure for recognizing in a problem specification the structures identified in the algorithm theory and then synthesizing a program. This paper describes ongoing work toward recasting the KIDS method of algorithm design, which is only partly algebraic, into a “purer” algebraic form, and on developing an algorithm theory and design tactic for local search algorithms.

Local search is a technique for solving optimization problems. Such problems are defined by a set of feasible solutions and a cost or objective function that assigns a numerical value to each solution. An optimization problem is solved by finding a solution whose cost is at least as good as the cost of every other solution. Local search algorithms can in some cases find solutions that are known to be optimal, or within a certain factor of an optimal solution. For many optimization problems, however, the known algorithms for solving them exactly are of exponential complexity, making them impractical for solving large instances. Local search is commonly used in these cases to generate good solutions even though no guarantees on solution quality are provided, and as a cheap way to improve solutions generated by other heuristic means.

The reader is assumed to be familiar with algebraic methods. The algebraic language used below is Slang, the language used in SPECWARE[3], but with a more elaborate syntax for logical formulas. Slang provides an external view of algebraic specifications and their morphisms in category theoretic terms, supporting both diagrams and colimits. It also provides an internal view of specifications based on higher-order logic and topos theory. We will try, in the space available, to explain the more distinctive features of the Slang language, such as definitional extensions, interpretations and interpretation morphisms, as they are introduced. For a fuller explanation the reader is referred to the SPECWARE Language Manual [4]. We will use the word “spec” from now on to refer to a Slang specification, reserving the word “specification” for more generic purposes.

The remaining sections of this paper are as follows. Section 2 provides an informal characterization of the family of local search algorithms. Section 3 presents a formal model for algebraic algorithm design in general. Section 4 presents an algorithm theory for local search over binary neighborhoods and a design tactic for matching such neighborhoods to problems. Section 5 describes related work on algebraic synthesis and local search. Finally, Section 6 presents conclusions and future work.

2 Characterizing local search algorithms

Local search algorithms search the space of feasible solutions by generating “new” solutions from existing ones, seeking improvement in the cost of the solutions generated. The word “new” is in quotes because local search is not systematic, in that there is nothing inherent in the approach that prevents a solution from being generated many times, or that guarantees that every feasible solution will eventually be generated or somehow ruled out. Local search is for the most part memoryless: no record is maintained of which solutions have been generated or what their costs were. This helps to keep the resource requirements (e.g., memory and CPU time) of local search algorithms modest and permits them to be used on problem instances of great size.

There is tremendous variety in local search algorithms. Many different approaches have been taken to address the problems of avoiding regeneration of solutions and finding the best solutions. This variety is a major reason why local search is interesting and important. We seek to formalize a model broad enough to encompass most or all of local search and then to specialize it to specific approaches. First, then, we
will describe the characteristic features of local search in an informal setting and outline some of the ways these features are realized in particular algorithms. The common features can be classified into three categories: overall strategy, finding an initial solution, and move discipline.

2.1 Overall strategy

The overall strategy adopted by a local search algorithm helps to determine the character of many of the other components and acts to coordinate their activities. One aspect of strategy is how many trials are run. Since local search does not guarantee optimality in all cases, running multiple trials from different starting points can produce a set of answers from which the best can be chosen. Trials can be run sequentially or in parallel; when parallel, solutions might “compete” with each other such that new solutions tend to be generated from the better current ones.

A second aspect of strategy concerns how the solution space is defined and searched. In many cases the space of feasible solutions is embedded in a larger space that includes solutions of the same general form that violate one or more of the conditions of the problem. It is sometimes valuable to include some or all of the infeasible solutions in the search. For example, some problems have special structure that allows the optimality of a solution to be determined by a simple test. One search technique in this case is to find an initial solution that satisfies this condition but may be infeasible, then look at which constraints are violated and incrementally change the solution to satisfy them. When all constraints are met, the optimal solution is at hand. In linear programming this approach is called a dual method. More generally, methods that search within the feasible region are called internal methods and those that search from without are called external methods. It is possible to combine both techniques, either by alternating periods of search in each region or by simply searching the larger space without considering whether any given solution is feasible or infeasible until the search is terminated.

A third aspect of strategy is whether or not to collect some sort of data during search. As stated above, local search is memoryless in that it keeps no detailed record of where it has been. It can be useful, however, to gather and use statistical data about the solutions visited. This data can be used either to intensify search in regions that seem promising, or conversely to diversify the search by forcing subsequent trials to begin at initial solutions that are unlike the solutions already searched.

2.2 Finding an initial solution

A local search algorithm must be able to generate one or more initial solutions in order to generate new ones. The choice of initial solution can have great impact on both the speed with which the search converges to a final solution and the quality of that solution. Some local search algorithms need the ability to generate a variety of different initial solutions; others need only one.

Any algorithmic technique can be applied to the initial solution task. Four general approaches can be identified. If only one initial solution is needed, an arbitrary choice can be made. This involves constructing or searching for a feasible solution by some means that always produces the same answer for any given input. When multiple initial solutions are needed, a random choice can be made. That is, there is some element of randomness to how the solution is generated, so that for any given input there is a set of possible outcomes. A heuristic choice attempts to produce an initial solution with above-average quality, in the hope that the final solution will either be of high quality, be found quickly, or both. Finally, a novel choice uses data gathered in previous trials to diversify the search deliberately rather than randomly. Both heuristic and novel choices may contain random elements as well.

2.3 Move discipline

Once an initial solution (or a set of them, as required) has been found, search proceeds by generating new solutions until some stopping criterion is met. At this point the best solution seen is returned, possibly along with other information about the search, with further actions as dictated by the overall strategy. Search proceeds in steps or moves. Each step is defined by a state consisting of the current solution(s) and possibly other information. The move discipline is a function on these states, defining how new states are generated, and which states are final. As stated above, only the current state is maintained; no record is kept of prior states. The move discipline consists of three components: neighborhood structure, move selection rules, and stopping criteria.

Neighborhood structure fundamentally defines what moves are possible by defining how new solutions are generated from old ones. In the simplest and most common case, the neighborhood structure is described by a binary relation on solutions. This relation defines for each solution a set of neighboring solutions. Typically, the relation is such that neighboring solutions are highly similar. Intuitively, such a neighborhood is generated by the action of a function that perturbs or incrementally transforms the solution. More generally, a neighborhood structure can be a higher-order relation describing how multiple solutions can be combined to produce new ones. For example, the crossover operators used in genetic algorithms define ternary relations that describe all the ways that two solutions can be crossed to produce offspring.

Properties of relations (symmetry, reflexivity, etc.) indicate structure in the neighborhood that can potentially be exploited by local search algorithms. One particularly important property of binary relations is reachability, which means that from any feasible solution it is possible to reach any other through a sequence of neighboring solutions. Reachability is not required for local search, but clearly it is a nice property to be able in principle to search the entire solution space from any initial solution. Another useful property is feasibility, which states that all neighbors of feasible solutions are feasible. If a neighborhood is both reachable and feasible, we call it a perfect neighborhood. Most “typical” examples of local search are over perfect neighborhoods.
The relationships between the neighborhood structure and the cost function are also important. If the cost of a solution is no worse than that of all of its neighbors, it is called a local optimum. If all local optima are global optima, then the neighborhood is called exact with respect to the cost function. The nature of local search is that it seeks local optima, so when exactness is not satisfied the search must somehow “escape” local optima and continue searching. Neighborhoods that are “smooth” are generally easier to search than “spiky” ones in which neighboring solutions often differ radically in cost.

The neighborhood structure defines the set of possible moves, but not all moves are created equal, so the move discipline includes a set of selection rules that are used to evaluate and select which move is accepted at each step. The selection rules together with the underlying neighborhood structure determine the quality of solution found and the probability that the search will get trapped in a loop of recurring states or otherwise fail to make progress. Some move disciplines guarantee that search states will not recur, others offer a low probability of recurrence, and still others make no promises whatsoever.

Moves can be classified first as improving, neutral or worsening, according to whether the solutions they generate are better, the same, or worse, respectively, than the current solutions. Clearly improving moves are preferred, but it is sometimes necessary to accept neutral or worsening moves in order to find better solutions that are more than one move away.

Selection rules evaluate moves in these categories and decide which to accept. The simplest and most obvious rule is to accept any improving move that is available. This is called hill climbing, and is generally considered the essence of local search. Steepest ascent is a refinement of hill climbing that always selects the move that makes the greatest improvement. This may lead to a shorter search, but also requires spending more time at each step. Other selection rules include forbidding certain moves, such as moves that undo other moves made recently, the defining characteristic of tabu search; selecting moves stochastically based on their effect on the cost function, the defining characteristic of simulated annealing; looking more than one move away from the current solution; and choosing a move arbitrarily or randomly, which is usually used only to break ties based on other criteria.

The most obvious stopping criterion is to stop when a globally optimal solution has been found. When searching an exact neighborhood, a necessary and sufficient condition for global optimality is local optimality. Even with an inexact neighborhood, however, sufficient conditions may be known and can be used to terminate search. Another common stopping rule is to stop when no improving moves are available, even if the solution might not be globally optimal. There are many ways to try to escape from a local optimum, and certainly one way is to stop searching and start over somewhere else. Rules based on resource limits, such as total number of moves made, or number made since the best solution seen was found, are other common stopping rules.

2.4 Example local search algorithms

The simplex algorithm for solving linear programs [5] is a classic local search algorithm. A linear program in standard form,

$$\min c^T x \text{ subject to } Ax = b, x \geq 0$$

is a continuous optimization problem, in that the variables $x$ are real-valued, but one can show analytically that optimal solutions always include at least one vertex of the feasible region defined by the constraints, so the number of solutions that actually needs to be considered is finite. The vertices in turn correspond to bases of the constraint matrix $A$, so the problem reduces to finding a subset of the variables, called basic variables, that yields an optimal solution. Neighbors of a solution are generated by swapping a non-basic variable into the current basis and swapping a basic variable out. This neighborhood is exact, so a single trial from an arbitrary initial solution is sufficient. The move selection rule is to choose the direction of most improvement in cost, and the stopping criterion is local optimality, which implies global optimality.

Selman et al. describe solving boolean satisfaction problems with local search [6]. The input is a boolean formula in conjunctive normal form, and the output is an assignment of variables to boolean values that minimizes the number of unsatisfied clauses. Neighbors of an assignment are generated by changing the value assigned to any one variable. This is not an exact neighborhood, so multiple runs are made from randomly generated assignments. Moves are selected by choosing randomly among the best neighbors of the current solution. Empirically this is usually either an improving or neutral move. Random selection prevents repeated cycles in the solutions generated and lets the search cross “plateaus” of equal-cost solutions to find better ones. Search terminates when a global optimum is found (zero unsatisfied clauses) or when a limit on the number of moves is reached. Empirical results show the algorithm to be very good at finding satisfying assignments when they exist, but failure to find one does not guarantee the formula is unsatisfiable.

Kernighan and Lin describe a very successful heuristic for the graph partitioning problem that is a variant of local search [7]. The problem is to partition the nodes of a graph into equal-size subsets so as to minimize the weighted cost of the cut edges. The neighborhood is generated by choosing one node in each part of the current partition and swapping their positions. This neighborhood is not exact, so multiple runs from randomly generated partitions are performed. The move strategy is to identify the swap yielding the best solution, then the best swap among the nodes not yet swapped, and so on until all nodes have been swapped. The chain of moves yielding the best solution in the sequence generated is then selected as the new current solution. This chain may contain worsening or neutral moves, as long as the net result is an improved solution. Search terminates when no improved solutions are in the sequence.
3 Algebraic algorithm design

Before defining the algorithm theory and design tactic for local search, we need some preliminary discussion to outline the general approach. First is a canonical representation for the problem for which we wish to design an algorithm. This is provided by

spec Problem is
  sorts D, R
  op I : D → Boolean
  op O : D, R → Boolean
end-spec

where $D$ is the domain or input sort, $R$ is the range or output sort, $I$ is a predicate defining legal inputs, and $O$ is a predicate characterizing the desired solutions. A particular problem is specified by giving an interpretation from the Problem spec to a domain theory spec.

Problem classes can also be presented in this way. Figure 1 is a specification for the class of global optimization problems. WFSS is a domain theory for a weighted feasible solution space, which consists of a feasibility problem (conveniently described by importing Problem) and an associated cost function. The spec TotalOrder, not shown, defines a sort $E$ and a total binary relation $\leq$ that is reflexive, anti-symmetric and transitive. Global optimization can be presented as the diagram

$\text{Problem} \rightarrow \text{Global-Optimum} \rightarrow d \rightarrow \text{WFSS}$

A diagram of this form, two morphisms with a common codomain, is called an interpretation. The middle spec, called the mediator, shows how the components of the source spec on the left can be defined in terms of the components of the target spec on the right. That is, GlobalOptimum identifies or defines the components that play the roles of $D$, $R$, $I$ and $O$ for global optimization problems, in terms of the elements provided by WFSS. The 'd' in the diagram indicates that GlobalOptimum is a definitional extension of WFSS, meaning it extends WFSS with new sorts and operations and provides definitions that uniquely characterize their behavior. Thus the mediator serves only to give names to terms that in a sense already exist in the target. A shorthand notation for this interpretation is $\text{Problem} \Rightarrow \text{WFSS}$. Any global optimization problem can be specified by providing a second interpretation $\text{WFSS} \Rightarrow A$ into some problem domain spec $A$ and then composing that interpretation with the one above to yield $\text{Problem} \Rightarrow A$ (composition of interpretations is explained in [4]).

Next we need a canonical form for solutions, thus:

spec Program is
  sorts D, R
  op I : D → Boolean
  op O : D, R → Boolean
  op F : D → R
axiom correctness is
  $\forall (x : D) \; I(x) \Rightarrow O(x, F(x))$
end-spec

spec CostOrder is
translate TotalOrder by \{E → R\}

spec WFSS is
import Problem, CostOrder
op Cost : D, R → R
end-spec

spec GlobalOptimum is
import WFSS

op Optimal : D, R → Boolean
definition of Optimal is
axiom $\forall (x : D, z : R) \; (\text{Optimal}(x, z) \Rightarrow$
  $\forall (z' : R) \; (O(x, z') \Rightarrow Cost(x, z) \leq Cost(x, z'))$
end-definition

op GO : D, R → Boolean
definition of GO is
axiom $\forall (x : D, z : R) \; (\text{GO}(x, z) \Rightarrow$
  $O(x, z) \land \text{Optimal}(x, z))$
end-definition
end-spec

interpretation GlobalOptimization :
  Problem ⇒ WFSS is
  mediator GlobalOptimum
  domain-to-mediator \{O → GO\}
codomain-to-mediator import-morphism

Figure 1: Canonical Form for Global Optimization Problems

This spec extends Problem by adding a function $F$ that computes outputs from inputs, and an axiom formalizing what correctness means: valid inputs are mapped to valid outputs. The behavior of $F$ on invalid inputs is unspecified. As for problems, a particular program is specified by giving an interpretation from Program. A solution is related to the problem it solves by an interpretation morphism as shown in Figure 2. An interpretation morphism is a set of three morphisms from one interpretation to another such that the resulting diagram commutes. It shows how one interpretation is embedded in or refined by another. In this case, the fact that the left square in particular commutes (that is, that $D$, $R$, $I$ and $O$ are mapped to the same symbols by the composition of Program → MyProblem with MyProblem → MyProgram) as by the composition of Program → Program with Program → MyProgram) guarantees that the solution specified by the bottom interpretation solves the problem specified by the top one. The domain theory $A$ is shown extended to $A'$ in the solution to represent the possible addition of new sorts and operators to support computation.

Algorithm theories and design tactics provide one means for completing the interpretation morphism of Figure 2. Design consists of two steps. First a prob-
lem is classified as belonging to a particular problem class possessing a certain structure for which one or more solution methods are known. Solution methods are represented by interpretation morphisms from the problem class specification to a program specification. In the second step one of these methods is selected and instantiated for the classified problem.

For a global optimization problem, the most natural way to specify the problem is to make use of WFSS and the interpretation Problem $\Rightarrow$ WFSS given above that defines this class; thus the user does the classification. For the global search tactic in KIDS, the user provides an arbitrary problem specification and provides guidance to the tactic for defining the additional structures needed by global search algorithms; the tactic automates many steps of the process. Design of local search algorithms will combine both of these styles: the user will typically provide a specification for an optimization problem, and the tactic will automatically construct a suitable neighborhood structure for it. Classification is a creative process that can be supported but not fully automated with current technology.

Figure 3 shows a generic classification step for when the problem specification is not already in a desired form. The domain theory of a problem class, called AT to emphasize its role in algorithm design, is used to "factor" a problem specification into a generic part representing a known class of problems and the details of a particular instance of the class. As before, it may be necessary to extend the domain theory A to support the classification. In the figure, $Problem \Rightarrow A$ has been factored into the composition of $Problem \Rightarrow AT'$ with $AT \Rightarrow A'$. Interpretation morphisms relate the original specification to the factored form, insuring that it specifies the same problem. $ATProblem \Rightarrow A'$ contains all the structure needed to solve an AT-problem, and the top interpretation morphism insures that this problem is the one specified. $AT \Rightarrow A'$ has a simpler source spec and serves to simplify the mediator similarly; the bottom interpretation morphism insures that the simplified form is consistent with the middle interpretation and so with the original problem specification.

Figure 4 shows a generic program scheme. The morphism $AT \rightarrow AT'$ represents problem-independent extensions needed by the program scheme, such as integers for iteration control, or deferred design decisions that are problem-dependent but unique to a particular solution method or considered secondary to the elements of the algorithm theory. We will see an example of the latter below. Figure 5 shows how a program scheme is instantiated. Instantiation is a purely syntactic operation that can be fully automated once the user has selected a program scheme. An interpretation morphism from $AT \Rightarrow A'$ is formed by taking the pushout of $AT \rightarrow MyAT$ and $AT \rightarrow A'$ and naming it $MyAT'$. This spec becomes the mediator of the target interpretation. A target spec, $A''$, is formed by removing from $MyAT'$ the definitions that were added to $A$ by $MyProblem$. $Program \Rightarrow AT'$ and $AT' \Rightarrow A''$ can then be composed to produce a program specification that solves the problem.

4 Application to local search

Figure 6 shows a domain theory for local search. It specifies a binary neighborhood structure defined over the same feasible solution space as the cost function. The class of local optimization problems can be defined in direct analogy to Figure 1. This formalization excludes genetic algorithms but still applies to all other common forms of local search.

Figure 7 is the spec for a binary neighborhood structure. It imports $Problem$ and introduces a new sort, $C$, from which will be taken local transform values that will be used in transforming a solution into one of its neighbors. The relation $N.info$ defines

Figure 5: Instantiating a Program Scheme
transforms the first into the second. The reflexive
reactions when there exists a legal transform value that
whether a given local transform value is compatible
and transitive closure of
feasibility, perfection and symmetry are shown. A hi-
fined. Axioms formalizing the notions of reachability,
search is done over the subsets of a set $S$ that
are of size $k$. A particular neighborhood is repre-
sented algebraically as an interpretation from the spec
Neighborhood to some domain theory. Doing so in
Slang would be unnecessarily detailed and volumi-
example characteristic of a number of problems where
search is done over $k$-subsets is an integer $k$ and a set $S$ (the
elements of which belong to some unspecified type $\alpha$),
such that the size of $S$ is at least $k$ and $k$ is non-
negative. The output sort is a set of elements of type
$\alpha$ and such a set is a feasible solution if it is a subset of
$S \setminus A$ and $j$ is in $A$. A new solution is generated from
$A$ by adding $i$ and removing $j$. This neighborhood is
both perfect and symmetric.

In developing a design tactic for local search, it
seems reasonable to assume that the problem spec-
ification is already expressed in terms of the global
optimization theory given above, so the Problem and
WFSS components of LocalSearch are already inter-
preted in the domain theory $A$. What remains, then,
is to define a neighborhood structure over the feasibility
problem defined by $\text{Problem} \Rightarrow A$. The result is
a specification for global optimization in the presence of
a neighborhood structure. If the neighborhood is
not exact, then local search cannot solve this problem
and it is up to the user to reformulate the problem as
local optimization.

spec LocalSearch is
  colimit of diagram
  nodes Problem, WFSS, Neighborhood
  arcs Problem -> WFSS : {},
     Problem -> Neighborhood :
     import-morphism
end-diagram

Figure 6: Local Search Domain Theory
The general problem of devising good neighborhoods for arbitrary problems ex nihilo has been closely examined for a long time and remains unsolved \cite{5,8}. My goal for the design tactic is to re-use known, proven neighborhoods by adapting them to new problems. One technique for accomplishing this adaptation is connections between specifications \cite{9}; by analyzing the axioms of Neighborhood, we find conditions under which an interpretation $\text{Neighborhood} \Rightarrow B$ can be used to extend an interpretation $\text{Problem} \Rightarrow A$ to an interpretation $\text{Neighborhood} \Rightarrow A'$. $A'$ represents a possible conservative extension of $A$ needed to complete the construction. The steps of a tactic for using connections to adapt a neighborhood are as follows:

1. Have the user select from a library of perfect neighborhood specifications a particular one, $\text{Neighborhood} \Rightarrow B$.

2. Combine the target specs $A$ and $B$ by forming the colimit of them and any user-identified common structures. The selected library neighborhood will usually have the same or a similar range sort $R$, for example, and if the library neighborhood is parameterized, as $k$-subset-1-exchange is on $\alpha$, this parameter will typically be instantiated with some component in the problem domain at this time. Incidental sharing of common specs such as natural numbers may also occur. Combine the mediator specs as well, over the same common structures, and form the morphism from the combined target to the combined mediator.

3. Add to the combined mediator signatures for two conversion functions,

\begin{align*}
\text{op } h_D &: D_A \to D_B \\
\text{op } h_R &: R_A \to R_B
\end{align*}

where $h_R$ is required to be an isomorphism. Also add two connection conditions,

- **axiom I-Condition** is

  \[ \forall (x : D_A) \ (I_A(x) \Rightarrow I_B(h_D(x))) \]

- **axiom O-Condition** is

  \[ \forall (x : D_A, z : R_A) \ (I_A(x) \Rightarrow (O_A(x, z) \Leftrightarrow O_B(h_D(z), h_R(z)))) \]

and definitions

\begin{align*}
\text{sort } C_A \quad &\text{sort-axiom } C_A = C_B \\
\text{op } N_{\text{info}} A &: D_A, R_A, C_A \to \text{Boolean} \\
\text{definition of } N_{\text{info}} A &\text{ is} \\
\text{axiom } \forall (x : D_A, z : R_A, c : C_A) (N_{\text{info}} A(x, x, c) \Rightarrow N_{\text{info}} B(h_D(z), h_R(z), c)) \\
\text{end-definition} \\
\text{op } N_{\text{is}} A &: D_A, R_A, C_A, R_A \to \text{Boolean} \\
\text{definition of } N_{\text{is}} A &\text{ is} \\
\text{axiom } \forall (x : D_A, z : R_A, c : C_A, z' : R_A) (N_{\text{is}} A(x, z, c, z') \Rightarrow N_{\text{is}} B(h_D(z), h_R(z), c, h_R(z'))) \\
\text{end-definition}
\end{align*}

and copy definitions for $N_A$ and $N_A'$ from Neighborhood.

4. Derive, using a theorem prover, definitions for $h_D$ and $h_R$ such that the connection conditions are theorems. Unskolemization \cite{9} is one technique for doing this.

It is straightforward to prove that under these circumstances the axioms of Neighborhood are theorems in the new mediator spec, and that this spec is a definitional extension of the combined target formed in step 2. This interpretation can be combined with $\text{Problem} \Rightarrow A$ and $\text{WFSS} \Rightarrow A$, since all three agree on the interpretation of Problem, yielding an interpretation from LocalSearch to the combined target.

This application of connection theory differs from \cite{9}. Conversion functions and connection conditions are derived only for symbols in Problem, not all of Neighborhood, with the strongest condition, equivalence, assumed for the rest and used to define them. The condition relating $O_A$ and $O_B$ is slightly weaker than polarity analysis gives, but still strong enough. The tactic for establishing the connection and using it to build an interpretation is more detailed and expressed more algebraically; space restraints prohibit a full explanation of our adaptation of the connection mechanism to the SPECWARE environment.

There’s a fly in the ointment, however, from a practical perspective: the perfection axiom (and in particular the feasibility aspect of it) requires the condition on $O$ to be an equivalence. This means that for every problem we wish to solve, we must have in our library a neighborhood for exactly that problem.

We can modify the tactic to avoid this problem. First we relax the equivalence to implication (from $O_A$ to $O_B$). This is enough to guarantee reachability of the neighborhood with respect to $A$, but not feasibility. It also allows our library to contain weak, general purpose neighborhoods (such as $k$-subset-1-exchange), which can be adapted to a variety of problems by restricting them. We could next attempt to show implication in the opposite direction, but most often it will not hold. Instead we use the feasibility axiom itself to derive a term we can use to restrict $N_{\text{info}} A$ to maintain feasibility. We use a theorem prover to derive a feasibility constraint, $FC$, with the same signature as $N_{\text{info}}$, that satisfies

\[ \forall (x : D_A, z : R_A, c : C_A, z' : R_A) \ (I_A(x) \land O_A(x, z) \land N_{\text{info}} B(h_D(z), h_R(z), c) \land N_{\text{is}} B(h_D(z), h_R(z), c, h_R(z')) \Rightarrow (FC(x, z, c) \Rightarrow O_A(x, z')) \]

and then change the definition of $N_{\text{info}} A$:

\[ \text{definition of } N_{\text{info}} A \text{ is} \]

\[ \text{axiom } \forall (x : D_A, z : R_A, c : C_A) (N_{\text{info}} A(x, z, c) \Rightarrow N_{\text{info}} B(h_D(z), h_R(z), c) \land FC(x, z, c)) \]

end-definition
If $FC$ is true, then $B$ is already a perfect match and we have a perfect neighborhood for $A$. If $FC$ is false, then no neighbors of feasible solutions are feasible (with respect to $A$) and this neighborhood is probably not a good choice for this problem. If $FC$ is neither, then it serves to "filter" $N_{info}$ to include only solutions feasible with respect to $A$. It is possible that including $FC$ in $N_{info}$ will violate reachability, so this axiom should be reverified.

Both the graph partitioning and linear programming problems above can be solved by local search using the $k$-subset-1-exchange neighborhood. For graph partitioning, the input conversion function $h_D$ extracts the node set from the input parameter, which for example might be in the form of a node-adjacency matrix containing edge costs, to be the set $S$, and defines $k$ to be half the size of this set. The output conversion function $h_B$ maps the first element, say, of a node partition to the output of $k$-subset-1-exchange and implicitly represents the second element as everything else; this mapping is isomorphic. With these conversions, the input connection condition is easily established, as is the implication form of the output condition. When $FC$ is derived, it is true, indicating $k$-subset-1-exchange is a perfect neighborhood for this problem.

For linear programming the input conversion function uses the number of rows and columns of the matrix $A$ to determine the number of basic variables and the variable set, respectively, needed by $k$-subset-1-exchange. The output conversion sets all non-basic variables to zero and solves the resulting system of equations for the basic variables. Again the input and output connection conditions are easily shown. In this case, however, the resulting neighborhood is not yet feasible: some variable swaps lead to solutions where some basic variables are negative. The term $FC$ derived for this case corresponds to the ratio test of the simplex, which insures feasibility and maintains reachability.

The elements of local search that are not formalized in LocalSearch—overall strategy, finding an initial solution, move selection rules and stopping criteria—are addressed in the program scheme or deferred to later in design. Design decisions are deferred by extending the domain theory for local search with sorts and operations that are not fully defined. For example, finding an initial solution is largely independent of other design decisions and is a significant subtask in its own right. To support the top-down design of an algorithmic solution to this problem, all local search program schemes extend the domain theory with some variant of an operation and an axiom

\[
\text{op } r_0 : D \rightarrow R \\
\text{axiom } \forall(x : D) (I(x) \Rightarrow O(x, r_0(x)))
\]

specifying the task of finding a feasible solution, with no further detail.

With these extensions and any others needed to support particular computations, a program scheme can be written. Figures 8 and 9 show a program scheme for strict hill climbing. The HillClimb domain theory extends LocalSearch with the operation $r_0$ and another operation, BetterNeighbor, which selects for the current solution a neighbor of strictly better cost. Since the exact choice of neighbor is not yet specified, this operation belongs in the domain theory as a deferred design choice. The mediator, HCProgram, provides the hill climbing algorithm itself. Given a feasible neighborhood, this algorithm will accept a legal input and produce a feasible, locally optimal solution. The interpretation from Program and interpretation morphism from LocalOptimization are also shown.

Once the program scheme is instantiated with the local search specification generated by the tactic above, the resulting specification of a specialized local search program can then be further refined, optimized and eventually translated to code. One refinement might specify steepest ascent, which is the selection at each step of the search of a neighbor of best cost, perhaps with ties broken randomly. Finding a best neighbor is an optimization problem in its own right, providing another target for top-down algorithm design.

5 Related work

The KIDS work on algorithm theories and design tactics includes theories for global search [1], divide and conquer [10], program reduction [11] and a number of program optimizations implemented as transformations. The global search theory has been applied successfully to the transportation scheduling domain [12]. Integration of algorithm design tactics into SPECWARE is an ongoing research effort at the Kestrel Institute [13].

Lowry proposed an algorithm theory and design tactic for local search [14]. His only example is a derivation of the simplex algorithm, including the ratio test by essentially the mechanism described above. The work presented here refines and extends his work with a more complete analysis of local search, a more purely algebraic approach to algorithm design, identification of additional neighborhood properties, and greater care in the tactic to insure these properties are established and maintained. The future work outlined below will further extend the new theory and tactic beyond Lowry's.

The literature on local search, primarily from the Operations Research community, is vast but largely informal. Much of it takes the form of descriptions of particular algorithms and the results of computational experiments. A good overview of the technique as a whole is in [5].

6 Conclusions and future work

KIDS uses a language called Regroup for program specification and implementation. The algebra used by the tactics is hidden and encoded procedurally. SPECWARE uses algebra for program specification and refinement, and the research presented here shows how to do algorithm design with the algebra explicit and declarative.

Algorithm design remains a difficult and challenging problem. The algebraic approach described here seems to capture all the necessary steps in a natural way and carefully maintains an "audit trail" from
The original problem specification through to the final solution. The algebraic approach lends itself very nicely to small, incremental steps and the organization of multiple alternatives into refinement hierarchies. Extensive, systematic reuse of software engineering knowledge, in the form of problem classes, algorithm theories and program schemes, is integral to the technique.

The connection mechanism provides a powerful and well-founded technique for reusing algorithm knowledge, but we saw for local search that it had to be substantially modified to be useful in practice. The proposed solution seems reasonable but not ideal, and largely specific to local search. Algebraic methods in general depend strongly on the ability of automated theorem provers to solve the inference tasks identified. This potential weakness can be mitigated by the decomposition of construction tasks into a series of very specific theorem proving tasks that individually do not require deep reasoning or extensive search to solve, and the connection mechanism supports this goal.

The specific work on local search presented here covers only the minimal aspects of the technique, focusing on neighborhood structure, the sine qua non of local search. Future work will shift the focus to move selection strategies. Techniques such as tabu search [15] and the Kernighan-Lin heuristic have wide applicability but involve additional problem-dependent structure beyond LocalSearch. We are investigating techniques for formalizing these structures algebraically and automating their discovery. Methods
for organizing the multiplicity of overall strategies, initial solutions, move strategies, and stopping criteria in meaningful combinations will also be explored.

References


