The Application of Automated Reasoning to Formal Models of Combinatorial Optimization

Paul Helman and Robert Veroff
University of New Mexico, Albuquerque, NM, U.S.A.
{helman,veroff}@cs.unm.edu

Abstract

Many formalisms have been proposed over the years to capture combinatorial optimization algorithms such as dynamic programming, branch and bound, and greedy. In 1989 Helman presented a common formalism that captures dynamic programming and branch and bound type algorithms. The formalism was later extended to include greedy algorithms. In this paper, we describe the application of automated reasoning techniques to the domain of our model, in particular considering some representational issues and demonstrating that proofs about the model can be obtained by an automated reasoning program. The long-term objective of this research is to develop a methodology for using automated reasoning to establish new results within the theory, including the derivation of new lower bounds and the discovery (and verification) of new combinatorial search strategies.

1 Introduction

Many formalisms have been proposed over the years to capture combinatorial optimization algorithms such as dynamic programming, branch and bound, and greedy (see, for example, [6, 13, 16, 17, 18, 19]). These models tend be relatively narrow in that each captures one specific class of solution, and hence provides neither conceptual unity nor a common framework in which the techniques can be studied. In 1989 Helman [9] presented a common formalism that captures dynamic programming and branch and bound type algorithms. The formalism was later extended to include greedy algorithms [10, 11].

Several of the classical models, most notably [13, 14, 15, 16, 17], are finite automata-based and have been used successfully to derive the functional equations associated with dynamic programming and to study the decidability of the general solvability of these problems and questions of their minimal representations. Other models of branch and bound [16, 19] view the solution process in terms of subproblem decomposition, where the exact nature of a subproblem is left unspecified. By abstracting out the nature of the subproblems, the model becomes more flexible, but it also becomes less computationally precise.

Our model of combinatorial optimization is algebraic and emphasizes computational aspects of the search of a finite solution space. The primary goals of our model are to provide: (i) a restricted model of computation in which problem lower bounds can be derived; (ii) a computationally precise model flexible enough to capture, analyze, and compare a broad-range of combinatorial optimization paradigms; and (iii) an algorithm design framework in which problems can be represented and their solutions formally verified.

The major features of our model are a free algebra over a finite set of "atoms", a congruence relation on the free algebra for capturing the equivalence of objects within a given application, and a dominance relation that both specifies the optimization criteria and supports the implicit enumeration strategies of many classical optimization algorithms.

While new lower bounds have been obtained from the model for several well-known problems [9], the task of deriving lower bounds often is quite difficult. The model is built upon a relatively complex axiom set, and lower bound proofs are often quite intricate. We have begun to investigate the applicability of automated reasoning techniques to the domain of our model. Our short-term objective is to use an automated reasoning program to prove some of the basic properties of the model. Our longer-term objective, which may be quite ambitious, is to develop a methodology for using automated reasoning to establish new results within the theory, including the derivation of new lower bounds and the discovery (and verification) of new combinatorial search strategies.

This paper reviews our results to date and indicates the directions our research will take. Specifically, Section 2 reviews our model of combinatorial optimization, restricting attention to those model components considered later in the paper. Section 3 reviews the foundations of automated reasoning, again emphasizing those aspects most relevant to the results reported in this paper, and discusses the representation of the axiom system used by the automated reasoning program OTTER [21], including a look at some of the difficult issues arising in attempting to apply automated reasoning to this non-traditional problem domain. Section 4 then presents some results of basic properties of the model obtained using the automated reasoning program. Finally, Section 5 indicates the types of open problems (e.g., sophisticated model properties, lower bounds) which we believe can be successfully addressed by automated reasoning techniques.

2 Summary of the Model

This section summarizes only the formalism of the model required for the results presented in the current paper. For a complete description of the model, see [9].

2.1 Problem Structures

A problem structure is a triple $P =< ATOMS, \cong, Q >$, where

> $ATOMS$ is a finite set of *atoms*. Every atom is a *conjunct*, and if X and Y are conjuncts so too is $(X \odot Y)$. The set of all conjuncts is denoted $CONJ$.
>
> $\cong$ is an equivalence relation on $CONJ$. $\cong$ captures that distinct conjuncts may represent the same application object. $\cong$ is assumed to be a congruence relation. That is, for all conjuncts W, X, Y, Z,
>
> $$(W \cong X) \wedge (Y \cong Z) \Longrightarrow (W \odot Y) \cong (X \odot Z).$$
>
> Q is a subset of $CONJ$ consisting of a finite number of full $\cong$-classes. Q is extension free in the sense that if q and its extension q' are in Q, then it must be that $q \cong q'$. Q plays the role of the solution space.

(We say conjunct c' *extends* conjunct c if c' can be obtained from c via application of zero or more $\odot$ operations. For example, if a and b are atoms, conjunct $((a \odot b) \odot c)$ extends conjunct c.)

Conjuncts are like strings over a finite alphabet, but, unlike strings, need not associate; different associations may represent different application objects (see Example 2.2 below). Conjuncts give us full flexibility. When we do wish to have distinct conjuncts model the same application object, we capture this via the congruence relation; we set $c \cong c'$ when we want these distinct conjuncts to represent the same application object, e.g., for one application we might define $\cong$ to make $\odot$ associative, for another to make it commutative, and for another, we might define $\cong$ to be the identity, so that all distinct conjuncts represent distinct application objects.

Example 2.1 *Traveling Salesperson Problem (TSP).* Given a graph, find a least cost tour through the graph. (Without loss of generality, we can assume the graph is complete.) The graph vertices correspond to the set of atoms. $\cong$ specifies that different associations of the same sequence of atoms are congruent; hence conjuncts behave as strings and represent paths through the graph. Q is the set of all conjuncts representing tours through the graph.

Example 2.2 *Matrix Multiplication.* Given a matrix product $A_1 \times A_2 \times ... \times A_n$ to compute, find the association of the product chain that results in the fewest number of scalar multiplications. The atoms are the matrices to be multiplied. $\cong$ is the identity relation; hence different associations of the same chain of matrices are distinct conjuncts. Q is the set of all conjuncts representing associations of the matrix product $A_1 \times A_2 \times ... \times A_n$.

2.2 Dominance Relations

The optimization criterion of a problem instance is captured by a dominance relation denoted α. α is a strict partial order (i.e., α is irreflexive, antisymmetric, and transitive) on Q which fails to order a pair of members of Q exactly when these conjuncts are congruent. Consequently, α totally orders the $\cong$ classes of Q. The optimization problem is to find an α-min element of Q.[1]

For example, in the TSP problem, α orders members of Q in accordance with the sums of the represented paths' edge costs; for matrix multiplication, α orders the represented matrix products in accordance with the number of scalar multiplications required.

A dominance relation also orders conjuncts outside of Q is accordance with the following condition: $c\alpha c'$ iff c has an extension q in Q such that $q\alpha q'$ for all q' in Q extending c'. (See the claused-based representation of this property in Example 3.2.)

Notice that once relation α is specified on Q, it has a unique extension to a relation on *CONJ* satisfying the above condition.

2.3 Computations

The model postulates that α between any pair of members of Q can be determined in one computational step. However, α cannot be determined feasibly, in general, on conjuncts outside of Q. Observe, for example, that it does not suffice in TSP to compare the lengths of an arbitrary pair of paths outside Q (i.e., of subtours); in general, all extensions of the paths must be considered to determine their relationship under α, and this is not computationally feasible.

Hence, with nothing but α on which to base our computations, our only method for solving the optimization problem is explicit enumeration: perform a sequence of α comparisons involving a representative of every $\cong$ class of Q until a minimal

[1]The full model also introduces the notion of a problem type which, informally, is a family of problem structures and associated dominance relations. In the remainder of this paper, we restrict attention to a single, fixed problem instance. We refer the reader to [9] for the development of problem types.

member of Q can be deduced. For many interesting problems, this is not computationally feasible, since Q will contain a huge number of $\cong$ classes.[2]

On the other hand, if α could somehow be determined between arbitrary pairs of conjuncts, highly efficient implicit enumeration strategies could be developed. From a single compare, $c\alpha c'$, a potentially huge number of elements of Q (i.e., all extensions of c') could be identified as being suboptimal.

We model classic implicit enumeration algorithms as performing a sequence of comparisons between selected pairs of conjuncts and inferring the suboptimality of extensions of the losers of the comparisons. Hence, the design of an implicit enumeration algorithm consists of two components:

i. identifying a relation $\ll$ on $CONJ$ such that the $\ll$ order of a pair of conjuncts c and c' can be determined in one computational step (or determined that this pair is not ordered under $\ll$) and such that it can be determined from $c \ll c'$ that c' has no optimal extensions in Q, and

ii. specifying a sequence of comparisons, i.e., a computation, that is guaranteed to eliminate all suboptimal members of Q.

The following two subsections consider these components of implicit enumeration algorithms in turn.

2.3.1 Computationally Feasible Optimality Relations

A computationally feasible optimality relation ($CFOR$) is a relation satisfying (for a fixed problem structure and α relation):

i. $\ll$ is a strict partial order on $CONJ$.

ii. $c_1 \ll c_2 \Rightarrow c_1$ and c_2 are not congruent under $\cong$.

iii. $c_1 \ll c_2 \Rightarrow (\exists q \in Q, \forall q' \in Q \; extending \; c_2 \; (q\alpha q'))$.

Property (iii) is expressed within the automated reasoning formalism in Example 3.4.

Dynamic programming algorithms typically are based on a special type of $CFOR$ which we have identified in [9] as comparability relations (CRs). A CR $\ll$ agrees with α on Q and is invariant under extension in the sense that

$$c_1 \ll c_2 \Rightarrow (c1 \odot c') \ll (c2 \odot c') \text{ and } (c' \odot c1) \ll (c' \odot c2)$$

for all conjuncts c' such that $(c1 \odot c')$ and $(c' \odot c1)$ have extensions into Q. It can be shown that $\ll$ orders a subset of those conjuncts ordered by α (i.e., that α refines

[2]For the TSP problem over N atoms, the number of classes is $(N-1)!$.

$\ll$), and that any *CR* is also a *CFOR* (but not conversely). In [7, 8] we discuss how the *CR* property is an abstraction of Bellman's principle of optimality [1].

As an example of a *CR*, consider the TSP and define $\ll$ as: $c \ll c'$ iff c and c' are simple paths over the same set of vertices having identical terminal vertices, and the sum of the edge costs of c is smaller than that of c'.

It may be interesting to note that while comparability relations appear "stronger" than *CFOR*s, they possess no additional computational power over *CFOR*s with respect to their ability to infer from a sequence of comparisons an optimal member of Q. In particular, the termination condition presented in Section 2.3.2 for *CFOR* computations remains a necessary condition, even if the comparisons are based on relations known to be *CR*s [9].

2.3.2 Computations

A computation based on a *CFOR* $\ll$ is a sequence of $\ll$ comparisons between pairs of conjuncts. The inferences that can be drawn from such a sequence of comparisons are exactly captured by the following definitions and theorem.

Definition. q is in the *QCLOSURE* of a set S of conjuncts if and only if there is a conjunct c' congruent to some conjunct in S and such that c' has an extension in Q which is congruent to q.

QCLOSURE is expressed within the automated reasoning formalism in Example 3.3.

Definition. If $c \ll c'$, we say c' is the *loser* of this comparison.

Theorem. Every member of the *QCLOSURE* of the set of losers of a sequence of *CFOR* comparisons is suboptimal.

It follows that a sequence of *CFOR* comparisons whose set of losers has a *QCLOSURE* which contains all but one $\cong$ class of Q can identify that class as being optimal. We call such a *QCLOSURE complete*. The above theorem establishes that computing a complete *QCLOSURE* is a sufficient termination condition for computations performing comparisons based on *CFOR*s. In [9] we show that a sequence of comparisons computing a complete *QCLOSURE* also is a necessary termination condition. As is seen in Section 4, the automated reasoning system has been able to obtain a proof that a complete *QCLOSURE* is a sufficient termination condition.

3 Automated Reasoning

The primary objective of automated reasoning is to develop computer programs that will help people solve problems requiring logical reasoning. Such programs already have been used successfully in several application areas, including, for example, solving open questions in mathematics and formal logic (see, for example, [22]) and solving a variety of verification and design problems (see, for example, [4]).

There are several different approaches to the automation of reasoning, depending on the underlying logical foundations and the general approach taken for manipulating and reasoning about logical constructs. In this paper, we consider an application of first-order refutation systems—specifically, resolution-style systems using the language of clauses [5]. A proof (by contradiction) in these systems consists of a sequence of clauses, starting with an initial set of axioms and hypotheses and ending with the derivation of the *empty clause*, indicating that a logical contradiction has been found. The derived clauses in a proof are the result of applying inference rules to clauses appearing earlier in the sequence.

Resolution-style reasoning programs provide a wide variety of problem solving capabilities and have several features that are under user control. Successful application of automated reasoning to a particular problem area requires careful consideration of the approach taken, including choices such as the following.

- Problem representation.
- Choice of inference rules.
- Strategies used for restricting and directing the application of inference rules.
- Strategies used for the management of information.

Representation. The clause language includes predicate symbols representing relations on some domain of elements and function symbols representing mappings on the domain. A set of clauses represents a first-order formula in conjunctive normal form. For example, the formula $(\forall x(P(x) \wedge Q(x)) \to (R(x) \wedge S(x)))$ is represented with the two clauses

```
-P(x) | -Q(x) | R(x).
-P(x) | -Q(x) | S(x).
```

where the symbol '|' represents logical "or", and the symbol '-' represents negation. All variables are (implicitly) universally quantified. The scope of a variable is the variable's containing clause; that is, each clause is considered to have its own quantifiers. [5] gives a procedure for transforming an arbitrary formula into the language of clauses.

Due to the generality of first-order logic and the richness of the clause language, there are many ways to represent a given fact or problem. This is significant because the choice of representation can effect dramatically the nature of the search for a proof or solution. For example, the simple fact that multiplication is associative, $(\forall x, y, z\ ((xy)z = x(yz)))$, can be represented in at least two distinct ways. One representation emphasizes equality relations and uses a special function symbol to represent multiplication.

```
EQUAL(prod(prod(x,y),z),prod(x,prod(y,z))).
```

A second representation uses a predicate `P(x,y,z)` to denote the product relation $xy = z$ and requires two clauses to capture associativity.

```
-P(x,y,xy) | -P(y,z,yz) | -P(xy,z,w) | P(x,yz,w).
-P(x,y,xy) | -P(y,z,yz) | -P(x,yz,w) | P(xy,z,w).
```

The first example formulation might be more effective if the strategies and inference rules used place an emphasis on functions. The second might be more effective if the emphasis is on predicates. Although experiments have illustrated dramatic differences in the behavior of problems represented with function and predicate orientations, there is no known reliable rule for deciding when one is better than the other.

Inference rules. The inference rules employed are based on resolution [25] and paramodulation [23]. More advanced rules, such as hyperresolution [24], UR-resolution [20] and linked inference rules [26, 31], are designed to take larger deductive steps than ordinary binary resolution and binary paramodulation, while avoiding some of the redundancy caused by the explicit generation of the intermediate steps.

Search strategies. Arbitrary or exhaustive application of inference rules generally is not successful. The growth of the set of clauses typically overwhelms reasonable memory and time constraints, even for simple problems running on the fastest and largest-memory machines available. It is necessary, in general, to employ strategies to restrict and direct the application of inference rules. Some examples of search strategies that have proven to be effective are *set of support* [29], *weighting* [20], *hints* [27], and *target* [26].

Management of information. Even with good choices for strategy, there typically is a proliferation of clauses. A number of features have proven to be effective at limiting this proliferation. One of the most common is subsumption [25], a deletion strategy in which less general clauses are discarded in the presence of more general clauses. Some other examples are the use of rewrite rules to simplify clauses [30], and the use of weighting as part of a deletion strategy.

OTTER [21] is an automated reasoning program that incorporates the described features and many more. It and its predecessors have been used successfully to solve a number of open questions in abstract algebra and formal logic, including the renowned result regarding the Robbins problem [22].

3.1 A Representation Strategy for the Current Application

In addition to the basic elements (for example, conjuncts, extensions, congruence, and α relations), we need to represent and reason about computations, cost models, and complexity issues associated with the execution of algorithms. Further, the nature of the theory is such that we need a representation that is general enough to support set-theoretic arguments, possibly even to the extent described in [3].

The following three examples illustrate our current representation for selected parts of the theory.[3] The examples make use of the following predicate symbols.

$ATOM(x)$	– x is an atom.
$CONJ(x)$	– x is a conjunct.
$ALPHA(x)$	– x is an α relation.
$REL(x,y,z)$	– the ordered pair (y,z) is in relation x.
$CONGR(x,y)$	– x is congruent to y.
$EXT(x,y)$	– x is an extension of y.
$EL(x,yset)$	– x is an element of set yset.
$Q(x)$	– x is in the distinguished set Q.
$QCLOSURE(xset,y)$	– y is an element of the $QCLOSURE$ of set xset.
$CFOR(z,za)$	– z is a $CFOR$ with respect to α relation za.

In addition, the term $dot(x,y)$ is used to denote the concatenation operation $x \odot y$.

Example 3.1 The first example is one of the elementary properties, defining conjuncts in terms of atoms and the concatenation operator ($\odot$). This property can be expressed with the formula

$$
\begin{aligned}
&(\forall \text{ x (CONJ(x)} \\
&\qquad \leftrightarrow \text{(ATOM(x)} \vee \\
&\qquad\qquad (\exists \text{ y, z (CONJ(y)} \wedge \text{CONJ(z)} \wedge \text{x = dot(y,z)))))),}
\end{aligned}
$$

which is represented with the following five clauses.

```
-ATOM(x) | CONJ(x).
-CONJ(x) | -CONJ(y) | z != dot(x,y) | CONJ(z).

-CONJ(x) | ATOM(x) | CONJ(conj1(x)).
-CONJ(x) | ATOM(x) | CONJ(conj2(x)).
-CONJ(x) | ATOM(x) | dot(conj1(x),conj2(x)) = x.
```

[3]In our current implementation, we are assuming a specific problem structure $P = < ATOMS, \cong, Q >$.

The function symbols `conj1` and `conj2` are Skolem functions introduced by the mapping of the formula to the clause language [5].

Example 3.2 The second example illustrates the representation strategy for α relations. Following the presentation of Section 2 and [9], we define α relations and their extension to all of *CONJ* in terms of three basic properties:

$$ALPHA(x) \leftrightarrow (PROPi(x) \wedge PROPii(x) \wedge PROPiii(x)).$$

The third property, for example, concerns the extension of the α relation to a partial order on *all* of *CONJ* and can be characterized as follows.

> $c\alpha c'$ if and only if there exists an extension of c to Q that is α every extension of c' to Q.

This property is expressed with the formula

```
(∀ xa (  PROPiii(xa)
         ↔
         (∀ x, y (  (CONJ(x) ∧ CONJ(y))
                    →
                    (  REL(xa,x,y)
                       ↔
                       (∃ xq1 (  Q(xq1)
                                 ∧ EXT(xq1,x)
                                 ∧ (∀ xq2 (  (Q(xq2) ∧ EXT(xq2,y))
                                             →
                                             REL(xa,xq1,xq2)))
                    )))
         ))
)),
```

which in turn is represented with the following fourteen clauses.

```
-PROPiii(xa) | -CONJ(x) | -CONJ(y) | -REL(xa,x,y)
   | Q(p3_1(xa,x,y)).
-PROPiii(xa) | -CONJ(x) | -CONJ(y) | -REL(xa,x,y)
   | EXT(p3_1(xa,x,y),x).
-PROPiii(xa) | -CONJ(x) | -CONJ(y) | -REL(xa,x,y)
   | -Q(xq2) | -EXT(xq2,y) | REL(xa,p3_1(xa,x,y),xq2).
```

```
-PROPiii(xa) | -CONJ(x) | -CONJ(y) | REL(xa,x,y)
   | -Q(xq1) | -EXT(xq1,x) | Q(p3_2(xa,x,y,xq1)).
-PROPiii(xa) | -CONJ(x) | -CONJ(y) | REL(xa,x,y)
   | -Q(xq1) | -EXT(xq1,x) | EXT(p3_2(xa,x,y,xq1),y).
-PROPiii(xa) | -CONJ(x) | -CONJ(y) | REL(xa,x,y)
   | -Q(xq1) | -EXT(xq1,x) | -REL(xa,xq1,p3_2(xa,x,y,xq1)).

PROPiii(xa) | CONJ(p3_6(xa)).
PROPiii(xa) | CONJ(p3_5(xa)).
PROPiii(xa) | REL(xa,p3_6(xa),p3_5(xa)) | Q(p3_3(xa)).
PROPiii(xa) | REL(xa,p3_6(xa),p3_5(xa))
   | EXT(p3_3(xa),p3_6(xa)).
PROPiii(xa) | REL(xa,p3_6(xa),p3_5(xa))
   | -Q(x1) | -EXT(x1,p3_5(xa)) | REL(xa,p3_3(xa),x1).
PROPiii(xa) | -REL(xa,p3_6(xa),p3_5(xa))
   | -Q(x2) | -EXT(x2,p3_6(xa)) | Q(p3_4(xa,x2)).
PROPiii(xa) | -REL(xa,p3_6(xa),p3_5(xa))
   | -Q(x2) | -EXT(x2,p3_6(xa)) | EXT(p3_4(xa,x2),p3_5(xa)).
PROPiii(xa) | -REL(xa,p3_6(xa),p3_5(xa))
   | -Q(x2) | -EXT(x2,p3_6(xa)) | -REL(xa,x2,p3_4(xa,x2)).
```

The first six clauses give the implication of a relation known to satisfy the third property of α relations. The last eight clauses can be used to deduce that a relation satisfies the third property. The function symbols `p3_1` through `p3_6` are Skolem functions introduced by the mapping of the formula to the clause language.

Example 3.3 The following formula defines the *QCLOSURE* of a set.

```
(∀ xset, xel
        (QCLOSURE(xset,xel) ↔
                (∃ xel1, xel2, xel3
                (
                  EL(xel1,xset)
                  ∧ CONGR(xel2,xel1)
                  ∧ EXT(xel3,xel2)
                  ∧ Q(xel3)
                  ∧ CONGR(xel,xel3)
                ))
        )
).
```

This formula is represented with the following six clauses.

```
-EL(xel1,xset)
   | -CONGR(xel2,xel1)
   | -EXT(xel3,xel2)
   | -Q(xel3)
   | -CONGR(xel,xel3)
   | QCLOSURE(xset,xel).

-QCLOSURE(xset,xel) | CONGR(xel,fqcl3(xset,xel)).
-QCLOSURE(xset,xel) | Q(fqcl3(xset,xel)).
-QCLOSURE(xset,xel) | EXT(fqcl3(xset,xel),fqcl2(xset,xel)).
-QCLOSURE(xset,xel) | CONGR(fqcl2(xset,xel),fqcl1(xset,xel)).
-QCLOSURE(xset,xel) | EL(fqcl1(xset,xel),xset).
```

The function symbols `fqcl1`, `fqcl2`, and `fqcl3` are Skolem functions introduced by mapping to the clause language.

Example 3.4 Finally, the following formula defines the key *CFOR* property. We note that the predicate `CFOR` takes as its second argument the α relation upon which the *CFOR* is based.

$$
\begin{array}{l}
(\forall \text{ z, za (CFOR(z,za)} \rightarrow \text{(ALPHA(za)} \wedge (\forall \text{ x, y ((REL(z,x,y)} \rightarrow \\
\qquad (\exists \text{ xq1 (Q(xq1)} \wedge (\forall \text{ xq2 ((Q(xq2)} \wedge \text{EXT(xq2,y))} \\
\qquad\qquad \rightarrow \text{REL(za,xq1,xq2)))))))))))).}
\end{array}
$$

This formula is represented with the following three clauses.

```
-CFOR(z,za) | ALPHA(za).
-CFOR(z,za) | -REL(z,x,y) | Q(cfor(z,za,x,y)).
-CFOR(z,za) | -REL(z,x,y) | -Q(xq2) | -EXT(xq2,y)
   | REL(za,cfor(z,za,x,y),xq2).
```

The function symbol `cfor` is a Skolem function introduced by mapping to the clause language.

3.2 Reasoning Effectively

The axiom set we are developing already is getting to be of substantial size, both in the number of clauses as well as in the number of distinct predicate and function symbols. Reasoning effectively about this application area will require the development of new and sophisticated strategies. For example, part of the theory consists of a natural hierarchy of defined terms. Many proofs will contain distinct subproofs at the various levels of abstraction defined by this hierarchy. Automated reasoning programs typically have not been good at pursuing multiple lines of reasoning simultaneously; mixing arguments at different levels of abstraction—similar to mixing cases of a case analysis—is not conducive to effective reasoning. [28] considers this aspect of effective reasoning and describes some possible solutions. One strategy, a type of "relevance strategy", has been particularly effective for the problems in the current application.

In addition to the issues associated with a large and complex axiom system, some types of problems will present special challenges. For example, several theorems are best proved by induction. Although some automated reasoning systems are particularly well suited for doing proofs by induction (see [2], for example), OTTER is not. When induction appears to be called for, as in Example 4.3, we set up the inductions ourselves; proving the base case and inductive steps as separate theorems. Two other notable automated reasoning challenges we face are reasoning about finite sets and proving theorems that require existence arguments (for example, the existence of a relation or computation with a certain property).

4 Example Theorems

To date, the main result we have obtained with the automated reasoning system is that the $QCLOSURE$ of a set of $CFOR$ losers cannot contain an optimal element of Q. Since the result implies that a computation yielding a complete $QCLOSURE$ is a sufficient termination condition, we have in fact been able to use the automated reasoning system to derive a key result of the theory. In this section, we present three theorems—including this key result—whose proofs have been obtained with the automated reasoning system OTTER. We present here only one of the three proofs; the interested reader is referred to [12] for additional proofs.

Example 4.1 The first theorem is a basic property of α relations.

Theorem. Let α be an α relation. Then for all x, y, u, and v,

$$((x \cong u) \wedge (y \cong v) \wedge (x\ \alpha\ y)) \Longrightarrow (u\ \alpha\ v).$$

The following clauses are used to represent this theorem. Since we are doing proofs by contradiction, the conclusion of the theorem is negated.

```
% alpha is an alpha relation
```

```
ALPHA(alpha).

% assumed congruences
CONGR(cx,cu).
CONGR(cy,cv).

% cx and cy are related by alpha
REL(alpha,cx,cy).

% negation: the respective congruent
% elements are not related
-REL(alpha,cu,cv).
```

Using the relevance strategy described in [28], OTTER finds the following proof.

```
6 [] ALPHA(alpha).
7 [] CONGR(cx,cu).
8 [] CONGR(cy,cv).
9 [] REL(alpha,cx,cy).
10 [] -REL(alpha,cu,cv).
22 [] -CONGR(x,y) | CONGR(y,x).
25 [] -CONGR(x,y) | CONJ(x).
26 [] -CONGR(x,y) | CONJ(y).
35 [] -CONGR(x,y) | -EXT(xe,x) | EXT(extc(x,y,xe),y).
36 [] -CONGR(x,y) | -EXT(xe,x) | CONGR(extc(x,y,xe),xe).
55 [] -Q(x) | -CONGR(x,y) | Q(y).
58 [] -ALPHA(xa) | PROPii(xa).
59 [] -ALPHA(xa) | PROPiii(xa).
75 [] -PROPii(xa) | -Q(x) | -Q(y)
        | -REL(xa,x,y) | -CONGR(x,z) | REL(xa,z,y).
76 [] -PROPii(xa) | -Q(x) | -Q(y)
        | -REL(xa,x,y) | -CONGR(y,z) | REL(xa,x,z).
89 [] -PROPiii(xa) | -CONJ(x) | -CONJ(y)
        | -REL(xa,x,y) | Q(p3_1(xa,x,y)).
90 [] -PROPiii(xa) | -CONJ(x) | -CONJ(y)
        | -REL(xa,x,y) | EXT(p3_1(xa,x,y),x).
91 [] -PROPiii(xa) | -CONJ(x) | -CONJ(y) | -REL(xa,x,y)
        | -Q(xq2) | -EXT(xq2,y) | REL(xa,p3_1(xa,x,y),xq2).
92 [] -PROPiii(xa) | -CONJ(x) | -CONJ(y) | REL(xa,x,y)
        | -Q(xq1) | -EXT(xq1,x) | Q(p3_2(xa,x,y,xq1)).
93 [] -PROPiii(xa) | -CONJ(x) | -CONJ(y) | REL(xa,x,y)
        | -Q(xq1) | -EXT(xq1,x) | EXT(p3_2(xa,x,y,xq1),y).
94 [] -PROPiii(xa) | -CONJ(x) | -CONJ(y) | REL(xa,x,y)
        | -Q(xq1) | -EXT(xq1,x)
```

```
          | -REL(xa,xq1,p3_2(xa,x,y,xq1)).
103 [hyper,6,59] PROPiii(alpha).
104 [hyper,6,58] PROPii(alpha).
106 [hyper,7,26] CONJ(cu).
107 [hyper,7,25] CONJ(cx).
109 [hyper,8,26] CONJ(cv).
110 [hyper,8,25] CONJ(cy).
113 [hyper,8,22] CONGR(cv,cy).
170 [hyper,110,90,103,107,9] EXT(p3_1(alpha,cx,cy),cx).
171 [hyper,110,89,103,107,9] Q(p3_1(alpha,cx,cy)).
317 [hyper,170,36,7]
      CONGR(extc(cx,cu,p3_1(alpha,cx,cy)),p3_1(alpha,cx,cy)).
318 [hyper,170,35,7] EXT(extc(cx,cu,p3_1(alpha,cx,cy)),cu).
338 [hyper,317,22]
      CONGR(p3_1(alpha,cx,cy),extc(cx,cu,p3_1(alpha,cx,cy))).
415 [hyper,338,55,171] Q(extc(cx,cu,p3_1(alpha,cx,cy))).
429 [hyper,415,93,103,106,109,318,unit_del,10]
      EXT(p3_2(alpha,cu,cv,extc(cx,cu,p3_1(alpha,cx,cy))),cv).
453 [hyper,415,92,103,106,109,318,unit_del,10]
      Q(p3_2(alpha,cu,cv,extc(cx,cu,p3_1(alpha,cx,cy)))).
480 [ur,415,94,103,106,109,10,318]
      -REL(alpha,extc(cx,cu,p3_1(alpha,cx,cy)),
           p3_2(alpha,cu,cv,extc(cx,cu,p3_1(alpha,cx,cy)))).
759 [hyper,429,36,113] CONGR(extc(cv,cy,
      p3_2(alpha,cu,cv,extc(cx,cu,p3_1(alpha,cx,cy)))),
      p3_2(alpha,cu,cv,extc(cx,cu,p3_1(alpha,cx,cy)))).
760 [hyper,429,35,113]
      EXT(extc(cv,cy,p3_2(alpha,cu,cv,
             extc(cx,cu,p3_1(alpha,cx,cy)))),cy).
1478 [ur,480,75,104,171,453,338]
       -REL(alpha,p3_1(alpha,cx,cy),
          p3_2(alpha,cu,cv,extc(cx,cu,p3_1(alpha,cx,cy)))).
5932 [hyper,759,22] CONGR(p3_2(alpha,cu,cv,
      extc(cx,cu,p3_1(alpha,cx,cy))),
      extc(cv,cy,p3_2(alpha,cu,cv,
      extc(cx,cu,p3_1(alpha,cx,cy))))).
8010 [hyper,5932,55,453] Q(extc(cv,cy,
      p3_2(alpha,cu,cv,extc(cx,cu,p3_1(alpha,cx,cy))))).
8365 [hyper,8010,91,103,107,110,9,760]
      REL(alpha,p3_1(alpha,cx,cy),
          extc(cv,cy,p3_2(alpha,cu,cv,
          extc(cx,cu,p3_1(alpha,cx,cy))))).
8395 [ur,8010,76,104,171,759,1478]
       -REL(alpha,p3_1(alpha,cx,cy),
          extc(cv,cy,p3_2(alpha,cu,cv,
```

```
            extc(cx,cu,p3_1(alpha,cx,cy))))).
8396 [binary,8395.1,8365.1] $F.
```

Each line of the proof consists of a clause number for reference, derivation information in square brackets, and a clause. The absence of derivation information indicates an input clause, either from the axioms or from the statement of the theorem. Clause `8396` is the empty clause, indicating that a proof by contradiction has been found. The numbers are not consecutive because the automated reasoning program deduces many clauses that do not participate in the proof.

Example 4.2 This theorem states that a conjunct that is congruent to a $CFOR$ loser cannot have an optimal extension in Q. The following clauses are used to represent this theorem.

```
% alpha and cfor relations
ALPHA(alpha).
CFOR(cfor1,alpha)

% cy is a CFOR loser
REL(cfor1,cx,cy).

% cy2 is congruent to cy
CONGR(cy,cy2).

% cy2ext is an extension of cy2 in Q
EXT(cy2ext,cy2).
Q(cy2ext).

% negation: cy2ext is optimal
MinOfSet(alpha,qset,cy2ext).
```

The proof of this theorem relies on a lemma, also proved by OTTER, that says that if conjuncts *cx* and *cy* are congruent, and *cxext* is an extension of *cx* in Q, then *cy* has an extension in Q that is congruent to *cxext*.

Example 4.3 This theorem states that the $QCLOSURE$ of a set of $CFOR$ losers cannot contain an optimal element of Q. This is the key step in establishing that $QCLOSURE$ being complete is a sufficient termination condition. We prove this theorem by induction on the set of losers. The inductive step is represented with the following clauses.

```
% alpha and cfor relations
ALPHA(alpha).
CFOR(cfor1,alpha)

% cbest is an optimal element of Q
MinOfSet(alpha,qset,cbest).

% cy is a CFOR loser
REL(cfor1,cx,cy).

% cbest is not in the QCLOSURE of cyset
-QCLOSURE(cyset,cbest).

% negation: cbest is in the QCLOSURE
% after adding cy to cyset
QCLOSURE(AddSet(cyset,cy),cbest).
```

5 Conclusions

In this paper we have demonstrated how a formal model of combinatorial optimization can be expressed in the clause-based language of an automated reasoning system and that an automated reasoning system can be used to derive results within the model.

To date, the proofs obtained with the automated reasoning system have been previously known results pertaining to properties of the model. Nevertheless, several of these results are nontrivial and give us hope that the automated reasoning system can be utilized to attack open problems of a variety of natures. Establishing lower bounds for problems is one type of result we would like to tackle with the automated reasoning system. Our model defines a restricted class of computation, and we have previously been able to obtain interesting lower bounds within this restricted class. Several advances will be necessary before we can succeed, including identifying suitable representations of a computation's cost and sophisticated search strategies.

Another type of result of interest pertains to the inference axioms used by different classes of algorithms. While the inferences made by a large number of dynamic programming and branch and bound algorithms are captured by the current formalism of the model, there are similar algorithms which exploit domain specific information. Optimization algorithms from computational geometry are important examples. We would like to axiomatize the additional information that these algorithms exploit and believe the automated reasoning system can be a valuable tool in helping us to formulate these extensions to the model.

We believe also that, in the future, the automated reasoning system can be used to verify, and even generate, algorithms stated within the model. This would be an important contribution to the applications community.

References

[1] Bellman, R., and Dreyfus, S., *Applied Dynamic Programming*, Princeton: Princeton University Press, 1962.

[2] Boyer, R., and Moore, J, *A Computational Logic* New York: Academic Press, 1979.

[3] Boyer, R., Lusk, E., McCune, W., Overbeek, R, Stickel, M., and Wos, L., "Set theory in first-order logic: clauses for Gödel's axioms", *J. Automated Reasoning* **2**(3), 1986, 287–327.

[4] Boyer, R., and Moore, J, "Mechanized formal reasoning about programs and computing machines", R. Veroff (ed.), *Automated Reasoning and Its Applications*, MIT Press, Cambridge, 1997, 147–176.

[5] Chang, C., and Lee, R., *Symbolic Logic and Mechanical Theorem Proving*, New York: Academic Press, 1973.

[6] Edmonds, J., "Matroids and the greedy algorithm", *Math. Programming* **1**, 1971, 127–136.

[7] Helman, P., "The principle of optimality in the design of efficient algorithms", *J. of Mathematical Analysis and Applications* **119**, 1986, 97–127.

[8] Helman, P., "An algebra for search problems and their solutions", L. Kanal and V. Kumar (eds.), *Search in Artificial Intelligence*, Springer Series on Symbolic Computation, Springer-Verlag, New York, 1988, 28–90.

[9] Helman, P., "A common schema for dynamic programming and branch and bound algorithms", *J. ACM* **36**(1), 1989, 97–128.

[10] Helman, P., A Theory of Greedy Structures Based on K-ary Dominance Relations, Technical Report CS89-11, University of New Mexico, Department of Computer Science, Albuquerque, New Mexico, 1989.

[11] Helman, P., Moret, B., and Shapiro, H., "An exact characterization of greedy structures", *SIAM J. on Discrete Math* **6**(2), 1993, 274–283.

[12] Helman, P., and Veroff, R., Automated Proofs of Some Results in Combinatorial Optimization, Technical Report, University of New Mexico, Department of Computer Science, Albuquerque, New Mexico, 1999, in preparation.

[13] Ibaraki, T., "Solvable classes of discrete dynamic programming", *J. of Mathematical Analysis and Applications* **43**, 1973, 642–693.

[14] Ibaraki, T., "On the optimality of algorithms for finite state sequential decision processes", *J. of Mathematical Analysis and Applications* **53**, 1976, 618–643.

[15] Ibaraki, T., "The power of dominance relations in branch and bound algorithms", *J. ACM* **24**, 1977, 264–279.

[16] Ibaraki, T., "Branch-and-bound procedure and state-space representation of combinatorial optimization problems", *Inform. Control* **36**, 1978, 1–27.

[17] Karp, R. M. and Held, M., "Finite-state processes and dynamic programming", *SIAM J. Appl. Math* **15**, 1967, 693–718.

[18] Korte, B., and Lovasz, L., "Greedoids and linear objective functions", *SIAM J. Alg. Discrete Meth.* **5**, 1984, 229–238.

[19] Kumar, V. and Kanal, L., "A general branch and bound formulation for understanding and synthesizing and/or tree search procedures", *Artificial Intelligence* **21**, 1983, 179–198.

[20] McCharen, J., Overbeek, R., and Wos, L., "Complexity and related enhancements for automated theorem-proving programs", *Computers and Mathematics with Applications* **2**, 1976, 1–16.

[21] McCune, W., OTTER 3.0 Reference Manual and Guide, Technical Report ANL-94/6, Argonne National Laboratory, Argonne, Illinois, 1994.

[22] McCune, W., "Solution of the Robbins problem", *J. Automated Reasoning* **19**(3), 1997, 263–276.

[23] Robinson, G., and Wos, L., "Paramodulation and theorem-proving in first-order theories with equality", B. Meltzer and D. Michie (eds.), *Machine Intelligence* **4**, Edinburgh University Press, 1969, 135–150.

[24] Robinson, J., “Automatic deduction with hyper-resolution”, *International Journal of Computer Mathematics* **1**, 1965, 227–234.

[25] Robinson, J., “A machine-oriented logic based on the resolution principle”, *J. ACM* **12**(1), 1965, 23–41.

[26] Veroff, R., and Wos, L., “The linked inference principle, I: the formal treatment”, *J. Automated Reasoning* **8**(2), 1992, 213–274.

[27] Veroff, R., “Using hints to increase the effectiveness of an automated reasoning program: case studies”, *J. Automated Reasoning* **16**(3), 1996, 223–239.

[28] Veroff, R., “Reasoning at multiple levels of abstraction”, in preparation.

[29] Wos, L., Robinson, G., and Carson, D., “Efficiency and completeness of the set of support strategy in theorem proving”, *J. ACM* **12**(4), 1965, 536–541.

[30] Wos, L., Robinson, G., Carson, D., and Shalla, L., “The concept of demodulation in theorem proving”, *J. ACM,* **14,**(4), 1967, 698–709.

[31] Wos, L., Veroff, R., Smith, B., and McCune, W., R. Shostak (ed.), “The linked inference principle, II: the user’s viewpoint”. *Proceedings of the 7th International Conference on Automated Deduction*, Lecture Notes in Computer Science, Vol. 170, Springer-Verlag, 1984, 316–332.