Solving Generalized Optimization Problems Subject to SMT Constraints

Feifei Ma[1], Jun Yan[1], and Jian Zhang[1,2]

[1] Institute of Software, Chinese Academy of Sciences
[2] State Key Laboratory of Computer Science
maff@ios.ac.cn, {junyan,jian_zhang}@acm.org

Abstract. In a classical constrained optimization problem, the logical relationship among the constraints is normally the logical conjunction. However, in many real applications, the relationship among the constraints might be more complex. This paper investigates a generalized class of optimization problems whose constraints are connected by various kinds of logical operators in addition to conjunction. Such optimization problems have been rarely studied in literature in contrast to the classical ones. A framework which integrates classical optimization procedures into the DPLL(T) architecture for solving Satisfiability Modulo Theories (SMT) problems is proposed. Two novel techniques for improving the solving efficiency w.r.t. linear arithmetic theory are also presented. Experiments show that the proposed techniques are quite effective.

1 Introduction

Many real-world and theoretical problems can be classified as optimization problems, i.e., minimizing or maximizing an objective function possibly subject to a set of constraints. A constraint is typically a mathematical equation or inequality. Optimization problems have long been the interest of mathematicians as well as engineers. Many efficient algorithms have been developed for various kinds of optimization problems over the past decades, for example, the simplex method for linear programming.

Although not explicitly stipulated, in a classical constrained optimization problem, the logical relationship among the constraints is the logical **AND** ($\wedge$), which means all of the equations or inequalities must be simultaneously satisfied. For such problems, solid theoretical foundations have been laid. However, in some applications, the relationship among the constraints might be more complex. The constraints may be connected by several kinds of logical operators so as to describe, for instance, a compound environment in a robot path planning problem or certain restrictions in task scheduling. In software testing and analysis, one strategy is to find out the conditions under which the software system uses resources heavily. This can also be formulated as an optimization problem with complex constraints. For some specific applications, tailored methods were sporadically proposed. But in the literature, such optimization problems have rarely been studied in the general form.

In this paper, we assume that the constraints are expressed as Satisfiability Modulo Theories (SMT) formulae. SMT, as an extension to the satisfiability (SAT) problem, has received more and more attention in recent years [1,4,12,10]. Instead of Boolean formulae, SMT checks the satisfiability of logical formulae with respect to combinations of background theories. Examples of theories include real numbers, integers, bit vectors and even non-linear constraints. An SMT solver typically integrates a powerful SAT solver as the Boolean search engine and several theory solvers for deciding the consistency of theory fragments.

In the sequel, we show how to solve an optimization problem which is constrained by SMT formulae via modifying the SMT solving procedure. Two pruning techniques are proposed so as to accelerate the searching process. Although the basic ideas of our approach are applicable to multiple theories, we are especially interested in problems with a linear objective function and Boolean combination of linear arithmetic constraints, or, SMT(LAC) constraints.

2 Background

This section describes some basic concepts and notations. We also give a brief overview of some existing techniques that will be used later.

The main object of study in this paper can be regarded as an optimization task whose constraints involve variables of various types (including integers, reals and Booleans). There can be logical operators (like AND, OR) and arithmetic operators.

We use b_i $(i > 0)$ to denote Boolean variables, x_j $(j > 0)$, $y, \ldots$, to denote numeric variables. A *literal* is a Boolean variable or its negation. A *clause* is a disjunction of literals. A Boolean formula in conjunctive normal form is a conjunction of clauses. A linear arithmetic constraint (LAC) is a comparison between two linear arithmetic expressions. A simple example is $x_1 + 2x_2 < 3$.

In general, a constraint ϕ can be represented as a Boolean formula $PS_\phi(b_1, \ldots, b_n)$ together with definitions in the form: $b_i \equiv expr_{i_1}\ op\ expr_{i_2}$. That means, the Boolean variable b_i stands for the LAC: $expr_{i_1}\ op\ expr_{i_2}$. Here $expr_{i_1}$ and $expr_{i_2}$ are numeric expressions, while *op* is a relational operator like '$<$', '$=$', etc. The LAC is called the *theory predicate* corresponding to b_i. The Boolean formula PS_ϕ is the *propositional skeleton* of the constraint ϕ.

For a Boolean formula, a *model* is an assignment[1] of truth values to all the Boolean variables such that the formula is evaluated to TRUE. Usually the Davis-Putnam (DPLL) procedure can be used to decide whether a Boolean formula has a model.

2.1 DPLL(T) Framework

Most of the state-of-the-art SMT solvers are built upon the DPLL(T) architecture [6]. DPLL(T) is a generalization of DPLL for solving the fragment of a decidable theory T. It combines a DPLL-based SAT solver and theory-specific solving [10] through a well-defined interface.

To solve an SMT formula, a Boolean abstraction of the formula is derived by encoding each theory predicate with a new propositional variable. The SAT solver then explores the Boolean search space of the formula, and passes the conjunctions of theory

[1] In this paper we represent an assignment as a set of literals.

predicates on to the theory solver for feasibility checking. For this integration to work well, however, the theory solver must be able to participate in propagation and conflict analysis, i.e., it must be able to infer new facts from already established facts, as well as to supply succinct explanations of infeasibility when theory conflicts arise.

2.2 Optimization Problem with Complex Constraints

In classical constrained optimization problems, the constraints are conjunctively connected. For instance, in linear programming, all linear inequalities must hold simultaneously. However, in many real applications, the relationships among constraints are very complex, and such problems cannot be handled with traditional optimization methods.

Let us see a practical example. In real-time operating systems, an important research issue is rate monotonic scheduling. Given a set of n periodic tasks $\{\tau_1 \ldots \tau_i \ldots \tau_n\}$, where each task τ_i is associated with a computation time C_i and a release period T_i, the goal is to assign the computation time to each task such that all the tasks can be scheduled and the performance of the system is optimized as well. It is proved that the optimization problem can be formulated as follows:

$$\begin{array}{ll} \text{maximize} & \Sigma_{i=1}^{n} \dfrac{C_i}{T_i} \\ \text{subject to} & \forall i \bigvee_{t \in S_i} (\Sigma_{j=1}^{i} \lceil \dfrac{t}{T_j} \rceil C_j - t \leq 0) \\ & C_i^{min} \leq C_i \leq C_i^{max} \end{array}$$

where $S_i = \{rT_j | j \leq i, r = 1, \ldots, \lfloor \frac{T_i}{T_j} \rfloor\}$. The objective function represents the CPU utilization. The constraints are integer linear inequalities, connected by disjunctions and conjunctions, forming an SMT(LAC) formula. In this paper, we study the optimization of a given function subject to SMT constraints[2].

3 Solving Optimization Problems with DPLL(T)

3.1 A Straightforward Method

We know that an SMT instance ϕ is satisfiable if there is an assignment α to the Boolean variables in PS_ϕ such that:

1. α propositionally satisfies ϕ, or formally $\alpha \models PS_\phi$;
2. The conjunction of theory predicates under the assignment α, which is denoted by $\hat{Th}(\alpha)$, is consistent w.r.t. the addressed theory.

We call an assignment satisfying the above conditions a *feasible* assignment. For example, suppose PS_ϕ is $b_1 \vee b_2 \vee b_3$, where $b_1 \equiv (x > 10), b_2 \equiv (y > 5), b_3 \equiv (x + 2y < 18)$. Then $\{b_1, b_2, \neg b_3\}$ is a feasible assignment; but $\{b_1, b_2, b_3\}$ is not.

[2] Without loss of generality, in the sequel we assume all objective functions are to be minimized.

Suppose an SMT optimization problem is composed of an objective function $f(x)$ and an SMT formula ϕ. The solution space of ϕ is the union of feasible regions of all feasible assignments of ϕ. Denote the set of all feasible assignments of ϕ by $Mod(\phi)$. The minimum value of $f(x)$ over ϕ, denoted by $Min(f(x), \phi)$, is defined as follows.

$$Min(f(x), \phi) = min\{Min(f(x), \alpha) | \alpha \in Mod(\phi)\}$$

Since α is a conjunction of theory predicates, $Min(f(x), \alpha)$ is a normal procedure for computing the optimal solution of a classical optimization problem.

The DPLL(T) architecture can be directly adapted to compute the optimal solution of a given formula by replacing the theory solver with a theory optimizer. We ask the SMT solver to enumerate all feasible assignments to formula ϕ, compute the optimal solution of each assignment and pick the smallest one.

However, the problem with the straightforward approach is that the classical optimization procedure is called as many times as the number of feasible assignments are. It is vital to the efficiency of the algorithm to reduce the number of calls to classical optimization procedures. In the rest of the section, we shall present two such techniques. The ideas behind the two techniques have one thing in common: they are both concerned with how to compute the optimal solution of an area as large as possible through a single call to the classical optimization procedure.

3.2 Optimization in Bunches

In [11], when studying the volume computation problem of SMT formulae, Ma et al. proposed a strategy called "volume computation in bunches". We find that the same idea is applicable to the optimization problem.

Given an assignment α to the Boolean variables in formula ϕ, concerning the two conditions for feasible assignments, we can distinguish four cases:

(i) $\alpha \models PS_\phi$, and $\hat{Th}(\alpha)$ is consistent in the specific theory. (Here α is a feasible assignment.)
(ii) $\alpha \models PS_\phi$, while $\hat{Th}(\alpha)$ is inconsistent in the specific theory.
(iii) α falsifies ϕ propositionally, while $\hat{Th}(\alpha)$ is consistent in the specific theory.
(iv) α falsifies ϕ propositionally, and $\hat{Th}(\alpha)$ is inconsistent in the specific theory.

The situation that α is a feasible assignment is just one of the cases when both the conditions are `true`. The optimal solution of formula ϕ is the smallest one amongst those of all assignments in case (i). When $\hat{Th}(\alpha)$ is inconsistent, as in case (ii) or case (iv), its feasible region can be viewed as empty. So when searching through the combined feasible regions of feasible assignments, it will be safe to count in some theory-inconsistent assignments since they would not affect the result. The theory-inconsistent assignments, when properly selected, can be combined with the feasible assignments to form fewer assignments, reducing the number of classical optimizations.

Definition 1. *A set of full assignments $\mathcal{S}$ is called a* bunch *if there exists a partial assignment α_c such that for any full assignment α, $\alpha \in \mathcal{S} \leftrightarrow \alpha_c \subseteq \alpha$. α_c is called the* cube *of $\mathcal{S}$.*

In other words, the assignments in a bunch $\mathcal{S}$ share a partial assignment α_c, and for the Boolean variables which are not assigned by the cube α_c, these assignments cover all possibilities of value combinations. For the assignments in a bunch, optimization can be greatly simplified, as the following theorem reveals:

Proposition 1. *Given an SMT optimization problem which is to minimize $f(x)$ over ϕ, for a bunch $\mathcal{S}$ with the cube α_c, the following equation holds:*

$$min\{Min(f(x),\alpha)|\alpha \in \mathcal{S}\} = Min(f(x),\alpha_c).$$

Example 1. Consider the optimization problem that minimizes $x - y$ subjected to ϕ, where $\phi = (((y + 3x < 3) \to (30 < y)) \vee (x \leq 60)) \wedge ((30 < y) \to \neg(x > 3) \wedge (x \leq 60))$.

We first introduce a Boolean variable for each linear inequality of ϕ and obtain its propositional skeleton $PS_\phi = ((b_1 \to b_2) \vee b_4) \wedge (b_2 \to \neg b_3 \wedge b_4)$, where $b_1 \equiv (y + 3x < 3)$, $b_2 \equiv (30 < y)$, $b_3 \equiv (x > 3)$ and $b_4 \equiv (x \leq 60)$.

There are seven feasible assignments, three of which are: $\alpha_1 = \{\neg b_1, \neg b_2, b_3, \neg b_4\}$, $\alpha_2 = \{\neg b_1, \neg b_2, \neg b_3, b_4\}$, and $\alpha_3 = \{\neg b_1, \neg b_2, b_3, b_4\}$. Their respective optimal solutions are[3]:

$Min(x - y, \alpha_1) = 30 + \delta$, where $x = 60 + \delta, y = 30$
$Min(x - y, \alpha_2) = -39$, where $x = -9, y = 30$
$Min(x - y, \alpha_3) = -27 - \delta$, where $x = 3 + \delta, y = 30$

Now let's consider another assignment: $\alpha_4 = \{\neg b_1, \neg b_2, \neg b_3, \neg b_4\}$. It is easy to check that α_4 satisfies PS_ϕ, but $\hat{Th}(\alpha_4)$ is inconsistent in linear arithmetic. Also, these four assignments form a bunch whose cube is $\{\neg b_1, \neg b_2\}$. Noticing this, we have

$$\begin{aligned}
& min\{Min(x - y, \alpha_1), Min(x - y, \alpha_2), Min(x - y, \alpha_3)\} \\
= \ & min\{Min(x - y, \alpha_1), Min(x - y, \alpha_2), Min(x - y, \alpha_3), Min(x - y, \alpha_4)\} \\
= \ & Min(x - y, \{\neg b_1, \neg b_2\}) \\
= \ & -39
\end{aligned}$$

where $x = -9, y = 30$. As a result, we need to call a linear programming routine only once rather than four times.

It would be ideal to incorporate the assignments in both case (ii) and case (iv) to form larger bunches. However, we currently neglect case (iv) because a typical DPLL(T)-style solver doesn't provide decision procedures for assignments that falsifies the propositional skeleton. For convenience, we just handle the assignments in case (ii). A key point is that when the SMT solver finds a feasible assignment, we try to obtain a smaller one which still propositionally satisfies the formula. It is formally defined as follows.

Definition 2. *Suppose α is a feasible assignment for formula ϕ. An assignment α_{mc} is called a* minimum cube *of α if i) $\alpha_{mc} \subseteq \alpha$ and $\alpha_{mc} \models PS_\phi$ and ii) $\forall \alpha'(\alpha' \models PS_\phi \to \alpha' \not\subset \alpha_{mc})$.*

[3] δ represents an arbitrarily small value since there exist strict inequlities.

In fact, the minimum cube α_{mc} of an assignment α is the cube of a bunch $\mathcal{S}$ such that for any bunch $\mathcal{S}'$, $\alpha \in \mathcal{S}' \rightarrow \mathcal{S} \not\subset \mathcal{S}'$. Any assignment in $\mathcal{S}$ also satisfies PS_ϕ because only part of it has evaluated PS_ϕ to be true. As we have explained before, it is pretty safe to count in such an assignment while solving the SMT optimization problem, regardless of its consistency in the specific theory.

Note that an assignment might have several minimum cubes. Currently we use a simple method to find only one minimum cube, as described in [11].

3.3 Feasible Region Expansion

When solving an SMT constrained optimization problem, the standard optimization procedure is called each time the SAT engine reaches a feasible assignment. If it is possible to obtain more information other than the optimal solution in the optimization subroutine, the whole searching process might benefit a lot.

For a standard optimization problem, there might be certain constraints which do not influence the optimal solution, in other words, the optimal solution is not bounded or restricted by these constraints. They are called **redundant constraints**, defined formally as follows.

Definition 3. *Suppose an optimization problem is to minimize $f(x)$ over a set of constraints $\{c_1, c_2, \ldots, c_m\}$. c_k $(1 \leq k \leq m)$ is a redundant constraint if the following equation holds:*

$$Min(f(x), \bigwedge_{1\leq i\leq m} c_i) = Min(f(x), \bigwedge_{1\leq i\leq m, i\neq k} c_i) \tag{1}$$

By definition, a standard optimization problem can be reduced to a simplified version by removing redundant constraints, while preserving the optimal solution. The feasible region of the problem expands as the constraints are removed. As a result, when we get an optimal solution for a feasible region, possibly we can conclude that it is also the optimal solution for a larger one. Inspired by this observation, we propose the second pruning strategy, namely "feasible region expansion". It can be used simultaneously with the "optimization in bunches" strategy. The basic idea is as follows: each time a feasible assignment α is obtained, find the optimal solution of its minimum cube $\alpha_m c$, then remove all redundant constraints from $\alpha_m c$, and get a smaller partial assignment α'. α' has the same optimal solution as $\alpha_m c$, while covers a larger region than $\alpha_m c$ does. So the negation of α' instead of $\alpha_m c$ is added to PS_ϕ. In this way, through one single call to the standard optimization subroutine, a piece of broader search space can be examined and excluded.

A key problem naturally arises: How to identify redundant constraints without extra calls to the optimization procedure? The problem is very general while the answer is closely related to the specific type of the constraints. We find that linear programming has a very fine property which makes the identification of some redundant constraints quite a simple task.

Proposition 2. *Given a linear programming problem with the objective $c^\top \boldsymbol{x}$ and subject to $A\boldsymbol{x} \otimes b$, if it has an optimal solution x_{opt}, then the linear constraint $A_i\boldsymbol{x} \otimes_i b_i$ is redundant if the point x_{opt} is not in the hyperplane $A_i\boldsymbol{x} = b_i$, or formally $A_i x_{opt} \neq b_i$.*

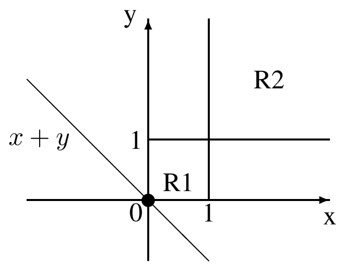
Such a constraint $A_i\mathbf{x} \otimes_i b_i$ is called a **non-binding constraint**. It does not affect the optimal solution.

Example 2. Given an SMT(LAC) optimization problem which is defined as:

$$\begin{aligned} \text{minimize} \quad & x + y \\ \text{subject to} \quad & \phi : x \geq 0 \wedge y \geq 0 \\ & \wedge (x \geq 1 \vee \neg(y \geq 1)) \qquad (2) \\ & \wedge (\neg(x \geq 1) \vee y \geq 1) \end{aligned}$$

We have $PS_\phi = b_1 \wedge b_2 \wedge (b_3 \vee \neg b_4) \wedge (\neg b_3 \vee b_4)$, where

$$\begin{cases} b_1 \equiv (x \geq 0); \\ b_2 \equiv (y \geq 0); \\ b_3 \equiv (x \geq 1); \\ b_4 \equiv (y \geq 1); \end{cases}$$



The optimization problem is illustrated in the right figure. The solution space of ϕ is the combined area of feasible regions R1 and R2. Suppose we get a feasible assignment $\alpha = \{b_1, b_2, \neg b_3, \neg b_4\}$, whose feasible region is R1. Obviously the minimum cube of α is α itself. The optimal solution of $Min(x + y, \alpha)$ is $\{x = 0, y = 0\}$, with the objective function evaluating to 0. Since the point $\{x = 0, y = 0\}$ is located on the lines $x = 0$ and $y = 0$, while away from the lines $x = 1$ and $y = 1$, the linear constraints corresponding to $\neg b_3$ and $\neg b_4$ are redundant constraints, thus can be omitted in the linear programming problem. The solution $\{x = 0, y = 0\}$ is also the optimal solution of the partial assignment $\{b_1, b_2\}$, which covers the whole solution space of ϕ. As a result, we know $\{x = 0, y = 0\}$ is the optimal solution of the original problem, without calculating the other feasible assignment $\{b_1, b_2, b_3, b_4\}$ with feasible region R2.

3.4 The Algorithms

In this subsection, we present the pseudo-codes of our approach and the pruning techniques.

Figure 1 describes the algorithm to compute the minimum cube of a given assignment α. The algorithm flips the literals in α one by one. More specifically, if α with literal l_i removed still evaluates PS to `true`, then l_i can be removed from α. Finally α becomes the minimum cube of the original assignment. Note that in an assignment, some variables are decision variables, while others get assigned by BCP. These implied literals need not be checked when finding the minimum cube of an assignment. (It has been proved in [11].)

Figure 2 illustrates the "feasible region expansion" strategy. For a given assignment α and its optimal solution, `FeaRegExpan` removes the redundant constraints from α.

The DPLL(T) framework for SMT solving is adapted to SMT optimization. The detailed algorithm is presented in Figure 3. *OptSol* stands for the optimal solution

```
MiniCube( Assignment $\alpha$, Boolean Formula $PS$ )
Assignment $\alpha'$;
for all Literal $l_i \in \alpha$ do
  if $l_i$ is a decision variable or its negation then
    $\alpha' = \alpha - \{l_i\}$;
    if $\alpha' \models PS$ then
      $\alpha = \alpha'$;
    end if
  end if
end for
return $\alpha$;
```

Fig. 1. Function: MiniCube

```
FeaRegExpan( Assignment $\alpha$, $solution$ )
for all literal $l_i \in \alpha$ do
  if $solution$ is not in the hyperplane $\hat{Th}(l_i)$ then
    $\alpha = \alpha - \{l_i\}$;
  end if
end for
return $\alpha$;
```

Fig. 2. Function: FeaRegExpan

```
1:  Boolean Formula $PS$ = $PS_\phi$;
2:  $OptSol$ = Null;
3:  while TRUE do
4:    if BCP() == CONFLICT then
5:      backtrack-level = AnalyzeConflict();
6:      if backtrack-level $< 0$ then
7:        return $OptSol$;
8:      end if
9:      backtrack to backtrack-level;
10:   else
11:     $\alpha$ = current assignment;
12:     if $\alpha \models PS$ then
13:       $\alpha$ = MiniCube($\alpha$, $PS$);
14:       $CurSol$ = LinearProgramming($\hat{Th}(\alpha)$);
15:       if $CurSol$ is unbounded then
16:         return UNBOUNDED;
17:       end if
18:       if $CurSol$ is smaller than $OptSol$ then
19:         $OptSol = CurSol$;
20:       end if
21:       $\alpha$ = FeaRegExpan($\alpha$, $CurSol$);
22:       Add $\neg\alpha$ to $PS$;
23:     else
24:       choose a Boolean variable and extend $\alpha$;
25:     end if
26:   end if
27: end while
```

Fig. 3. DPLL(T) for Optimization

of the SMT optimization problem, and $CurSol$ is the optimal solution of the current assignment during the search.

At the beginning of the algorithm, the propositional skeleton of the SMT formula ϕ is extracted and denoted as PS. After Boolean constraint propagation (BCP()), if the current assignment α already satisfies the Boolean formula PS, the subroutine `MiniCube` is called to the minimum cube of α.

When the minimum cube is obtained, the algorithm calls a linear programming package to compute the optimal solution of the cube. If the optimal solution $CurSol$ exists, the algorithm replaces $OptSol$ with $CurSol$ if the latter is smaller, and call the subroutine `FeaRegExpan` to eliminate redundant constraints from α. Otherwise, α is inconsistent and `PostCheck` is called to reduce it. Then its negation $\neg\alpha$ is added to PS so that the feasible region associated with the reduced α would not be counted more than once. It is a blocking clause, ruling out all the assignments in the bunch related to the reduced α. The algorithm terminates when there is no model for PS and returns $OptSol$, or when an unbounded solution is discovered.

Although Figure 3 describes the optimization approach under the assumption that all linear constraints are defined over real numbers, it can also handle integer linear constraints with slight modification. Firstly, in line 14, the LP relaxation of the integer linear programming is solved. Then if $CurSol$ is smaller than $OptSol$, $\hat{Th}(\alpha)$ is solved with integer linear programming, otherwise there is no need to compute the integer optimal solution, and "feasible region expansion" strategy is applied w.r.t. the real optimal solution $CurSol$.

4 Implementation and Experimental Results

The algorithm was implemented using the SAT solver `MiniSat` 2.0 [5], which serves as the search engine for the Boolean structure of the SMT(LAC) instance. The linear programming tool Cplex [8] is integrated for consistency checking and optimization w.r.t. a set of linear constraints. A laptop with Core 2 duo 2.10 GHz CPU running 32-bit linux was used for the experiments.

To study the effectiveness of the aforementioned techniques, we randomly generated a number of SMT(LAC) instances whose propositional skeletons are in CNF, and the length of clauses varies from 3 to 5. We compared both the running times and the numbers of calls to Cplex on these random instances in three settings: with no pruning strategy, with only "optimization in bunches", and with both the "bunch" and "feasible region expansion" employed. The results are listed in Table 1, where '-' represents a timeout of 30 minutes. Obviously each of the pruning techniques can reduce the running time and number of calls significantly, even by several orders of magnitude in some cases. The denotations `#LC`, `#C`, `#V` and `#calls` represent the number of linear constraints, clauses, numerical variables and calls to Cplex, respectively.

We also compared the performance of our program with Cplex on some random instances. The inputs for Cplex are mixed integer linear programming problems translated from SMT(LAC) instances by introducing "Big-M" constraints. Table 2 lists the running results (where OptSMT denotes our program), which indicate that our method outperforms MILP on instances with large number of boolean constraints.

Table 1. Comparison of Techniques

	No Pruning		Bunch		Bunch&FRE	
(#LC #C #V)	Time	#calls	Time	#calls	Time	#calls
(20 40 15)	-		124.34	15,228	0.05	52
(20 50 9)	26.66	17,595	0.92	1,175	0.15	262
(20 50 10)	-		227.73	19,453	0.06	79
(20 100 10)	89.42	35,463	8.66	5,236	0.32	383
(40 400 20)	-		219.61	23,192	18.03	4,908
(40 500 20)	-		1178.80	44,759	47.41	9,421
(100 800 60)	498.49	19,844	136.17	3,820	25.72	804
(100 700 80)	-		215.18	3,976	47.95	913
(100 1000 60)	209.57	5,160	24.75	693	5.53	139

Table 2. Comparison with Cplex

Parameters			Time	
#LC	#C	#V	OptSMT	Cplex
100	600	50	78.08	83.42
100	800	50	0.12	31.22
100	1000	50	0.05	14.52
200	1000	100	16.63	4.40
200	1000	120	0.19	4.91
200	1200	100	5.71	34.10
200	1400	100	242.35	-
200	1600	100	5.54	-
200	2000	100	0.96	-

The pruning techniques are effective for integer linear constraints as well. Here we just give an example instead of a thorough evaluation. Let us consider an instance of rate monotonic scheduling problem which has been introduced in the background section. It consists of 4 tasks, with characteristics defined as $\{T_1 = 100, T_2 = 150, T_3 = 210, T_4 = 400\}$, and $\{20 \leq C_1 \leq 60, 20 \leq C_2 \leq 75, 30 \leq C_3 \leq 100, 30 \leq C_4 \leq 150\}$. Our program finds an optimal solution $\{C_1 = 33, C_2 = 20, C_3 = 30, C_4 = 148\}$, with the optimal value 0.9762. Cplex is called 59 times, and the running time is 0.028s. If the "feasible region expansion" technique is disabled, there are 75 calls to Cplex and the running time is 0.040s. However, if both techniques are disabled, the number of calls grows to 2835, and it takes 4.123s to find the same solution.

We also did experiments on real optimization problems in software testing area. For instances with dozens of integer variables, our program can find the optimal solution within 0.1s, which is quite useful to software testing practitioners.

5 Related Works and Discussion

The problem studied in this paper is a generalization of SMT, and also a generalization of the traditional optimization (linear programming) problem.

As we mentioned at the beginning, there are quite some works on solving the SMT problem. But few of them care about optimization, except for two recent works in [3] and [13] which discuss a variant of weighted Max-SAT problem, namely weighted Max-SMT. Each clause of the SMT formula is associated with some weight or cost. The task is to find a feasible assignment such that the total weight of satisfied clauses are maximized, or a given cost function is minimized. It is possible to translate weighted Max-SMT into our optimization problem in general. However, weighted Max-SMT has its own features, and deserves independent study.

Many works on constraint solving and constraint programming (CP) can be extended to deal with optimization. In particular, Hooker and his collaborators have been advocating the tight integration of CP and classical optimization techniques (like mixed integer linear programming) [7]. One promising approach is mixed logic linear programming (MLLP) which extends MILP by introducing logic-based modeling and solutions [9]. However, it seems that the constraints they considered have simpler logical structures than those studied in this paper. We did not compare with their work for we can only find a demo version of their tool and our experimental instances cannot be translated into the demo tool's input automatically.

Cheng and Yap studied search space reduction methods for constraint optimization problems [2] where the constraints are of some special forms. The variables are all 0-1 integer variables and the constraints are basically linear equations. The methods perform quite well on the Still-Life problem. We will investigate whether they will be helpful on the more general problems.

6 Concluding Remarks

Optimization problems occur naturally in various important applications. Traditionally the constraint part in such a problem is a set of inequalities between arithmetic expressions. In this paper, we investigate a generalized class of optimization problems constrained by arbitrary Boolean combinations of linear inequalities. Based on the DPLL(T) framework for SMT, we present an exact optimization algorithm augmented with two efficient pruning techniques. The experimental results show that our approach is very promising. A significant future study is to incorporate more theories into the framework.

Acknowledgements. The work is supported by the National Natural Science Foundation of China (NSFC) under grant No. 61100064 and No. 60903049, and partially funded by the State Key Laboratory of Rail Traffic Control and Safety of Northern Jiaotong University. The authors are grateful to Tian Liu for his comments on an earlier version of the paper.

References

1. Barrett, C.W., Tinelli, C.: CVC3. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 298–302. Springer, Heidelberg (2007), `http://www.cs.nyu.edu/acsys/cvc3`
2. Cheng, K.C., Yap, R.H.C.: Search space reduction and Russian doll search. In: Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI 2007) (2007)
3. Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., Stenico, C.: Satisfiability Modulo the Theory of Costs: Foundations and Applications. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 99–113. Springer, Heidelberg (2010)
4. Dutertre, B., de Moura, L.: A Fast Linear-Arithmetic Solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006), `http://yices.csl.sri.com/`
5. Eén, N., Sorensson, N.: The MiniSat Page (2011), `http://minisat.se/`
6. Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: DPLL(*T*): Fast Decision Procedures. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 175–188. Springer, Heidelberg (2004)
7. Hooker, J.N.: Logic, optimization, and constraint programming. INFORMS Journal on Computing 14, 295–321 (2002)
8. IBM. Cplex, `http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/`
9. Hooker, J.N., Osorio, M.A.: Mixed logical-linear programming. Discrete Appl. Math. 96-97, 395–442 (October 1999)
10. Kroening, D., Strichman, O.: Decision Procedures. Springer (2008)
11. Ma, F., Liu, S., Zhang, J.: Volume Computation for Boolean Combination of Linear Arithmetic Constraints. In: Schmidt, R.A. (ed.) CADE-22. LNCS, vol. 5663, pp. 453–468. Springer, Heidelberg (2009)
12. de Moura, L., Bjørner, N.S.: Z3: An Efficient SMT Solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008), `http://research.microsoft.com/projects/z3/index.html`
13. Nieuwenhuis, R., Oliveras, A.: On SAT Modulo Theories and Optimization Problems. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 156–169. Springer, Heidelberg (2006)

A Proofs

Proof of **Proposition** 1

Proof. Firstly, we show that if the left side of the equation is infeasible (doesn't have any solution), so is the right side. Assume that the left side is infeasible, which means that $\{Min(f(x), \alpha) | \alpha \in \mathcal{S}\}$ is an empty set, while the right side has an optimal solution, namely $x^\star$. For a Boolean variable b_i which is not assigned by α_c, its truth value can be determined w.r.t $x^\star$: if $x^\star$ satisfies $\hat{Th}(b_i)$, then b_i is true. Denote the set of such literals by $\alpha_c^\star$, we now get a full assignment $\alpha^\star = \alpha_c \cup \alpha_c^\star$. Obviously $x^\star$ satisfies $\hat{Th}(\alpha^\star)$, and $x^\star$ is the optimal solution of $Min(f(x), \alpha^\star)$. Since $\mathcal{S}$ is a bunch with a cube α_c, we have $\alpha^\star \in \mathcal{S}$. Therefore, the assumption that $\{Min(f(x), \alpha) | \alpha \in \mathcal{S}\}$ is empty is contradicted.

Now assume that $x^\star$ is the optimal solution to the left side of the equation, and its corresponding assignment is $\alpha^\star$, i.e., $min\{Min(f(x), \alpha) | \alpha \in \mathcal{S}\} = Min(f(x), \alpha^\star) = f(x^\star)$. Since $\alpha_c \subseteq \alpha^\star$, $x^\star$ is a feasible solution of $\hat{Th}(\alpha_c)$, $Min(f(x), \alpha_c)$ is not infeasible and we have $f(x^\star) \leq Min(f(x), \alpha_c)$. Suppose $Min(f(x), \alpha_c)$ has an optimal solution x', or formally $Min(f(x), \alpha_c) = f(x')$. A full assignment α' can be constructed w.r.t x' in the same way as we have mentioned above. Because x' satisfies $\hat{Th}(\alpha')$, we have $f(x') \leq Min(f(x), \alpha')$. Also since $\alpha_c \subseteq \alpha'$, we have $\alpha' \in \mathcal{S}$ and thus $Min(f(x), \alpha') \leq f(x^\star)$. Therefore, $f(x') = f(x^\star)$.

The situation that the optimal value is unbounded can be viewed as a special case of the second situation. □

Proof of **Proposition** 2

Proof. A well-known fact about linear programming is that if the optimal solution exists, it must be one of vertices of the polytope defined by the linear constraints. This property is essential to our proof. We denote the polytope defined by $A\mathbf{x} \otimes b$ as $\mathcal{P}$, and the polytope without the constraint $A_i\mathbf{x} \otimes_i b_i$ as $\mathcal{P}'$.

We firstly explain that with $A_i\mathbf{x} \otimes_i b_i$ removed from the constraints, the linear programming problem still has an optimal solution. Assume without the constraint $A_i x \otimes_i b_i$, the problem is unbounded. There must exist a point x_1 such that $c^\top x_1 < c^\top x_{opt}$, and satisfies all constraints except for $A_i x \otimes_i b_i$. Since the two points x_{opt} and x_1 are on different sides of the hyperplane $A_i x = b_i$, the line connecting the two points will intersect with the hyperplane at some point, say x_2. Obviously x_2 satisfies all constraints including $A_i x \otimes_i b_i$. Moreover, because of the monotonicity of the objective function, we have $c^\top x_1 < c^\top x_2 < c^\top x_{opt}$. So x_2 is a solution and is smaller than x_{opt}, contradicting the precondition of the theorem. As a result, the linear programming problem is still bounded without the constraint $A_i x \otimes_i b_i$. Also, the possibility for the problem to be infeasible can be directly ruled out. There must be an optimal solution of the problem.

Suppose the new optimal solution is x'_{opt}. According to the property of linear programming, it is a vertex of $\mathcal{P}'$. Similarly, x_{opt} is a vertex of $\mathcal{P}$. Since $\mathcal{P}'$ is obtained via removing the hyperplane $A_i\mathbf{x} \otimes_i b_i$ from $\mathcal{P}$, each vertex of $\mathcal{P}'$ is also a vertex of $\mathcal{P}$. So we have $c^\top x_{opt} \leq c^\top x'_{opt}$. Because the point x_{opt} is out of the hyperplane $A_i\mathbf{x} = b_i$, it is also a vertex of $\mathcal{P}'$ and we have $c^\top x'_{opt} \leq c^\top x_{opt}$. As a result, x_{opt} and x'_{opt} are the same optimal solution, and the constraint $A_i x \otimes_i b_i$ is redundant. □