Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 165927, 6 pages
http://dx.doi.org/10.1155/2013/165927

Research Article

Message Passing Algorithm for Solving QBF Using More Reasoning

Su Weihua, Yin Minghao, Wang Jianan, and Zhou Junping

School of Computer Science and Information Technology, Northeast Normal University, Changchun 130117, China

Correspondence should be addressed to Wang Jianan; wangjn@nenu.edu.cn and Zhou Junping; zhoujp877@nenu.edu.cn

Received 26 June 2013; Accepted 24 July 2013

Academic Editor: William Guo

We present a novel solver for solving Quantified Boolean Formulae problem (QBF). In order to improve the performance, we introduce some reasoning rules into the message passing algorithm for solving QBF. When preprocessing the formulae, the solver incorporates the equality reduction and the hyperbinary resolution. Further, the solver employs the message passing method to obtain more information when selecting branches. By using the unit propagation, conflict driven learning, and satisfiability directed implication and learning, the solver handles the branches. The experimental results also show that the solver can solve QBF problem efficiently.

1. Introduction

Quantified Boolean Formulae can be seen as an extension of the well-known canonical NP-complete problem of Propositional Satisfiability (SAT) with existential or universal quantifiers to every variable in the propositional formulae. Given a Quantified Boolean Formula, the question of deciding the satisfiability of the formula is called a Quantified Boolean Satisfiability problem (QBF). QBF is an important issue in artificial intelligence because it is the prototypical PSPACE-complete problem. Many practical problems can be transformed into QBF, for example, conformant planning [1], verification [2], nonmonotonic reasoning [3], and reasoning about knowledge [4].

Now most researchers make great efforts on designing excellent QBF solvers in order to increase the efficiency for solving QBF problem. WalkQSAT is the first incomplete QBF solver using stochastic local search methods, and it applies a SAT solver—walksat for selecting the next branch [5]. WATCHEDCSBJ features watched data structure [6]. QuBE is an adaptation of the classic DPLL (Davis, Putnam, Logemann, and Loveland) backtracking search algorithm. Moreover, it adopts watched data structure and learning [7]. QSAT employs the maximum occurrences in minimum sized clauses (MOMS) heuristic [8]. Quaffle extends variable state independent decaying sum (VSIDS) heuristic and conflict driven learning [9].

In this paper, we propose a novel QBF solver based on a DPLL algorithm, called EHSPQBF. The system combines many advanced techniques together. In preprocessing, we adopt the equality reduction and the hyperbinary resolution to simplify the formulae. In choosing branches process, we employ a message passing method—survey propagation. By giving exact information, the approach selects branches more exactly, which can decrease the search space and improve the backtracking time. In branched treatment process, we utilize conflict reasoning, conflict driven learning, and satisfiability directed implication and learning to help pruning the search space. Owing to these outstanding techniques, the ability of solving QBF problems has greatly improved.

2. Quantified Boolean Formulae

We begin this section by presenting some notions in this paper. A literal is either a Boolean variable v or its negation $\neg v$. If a literal is l, the negation of the literal is $\neg l$. A clause is a disjunction of literals which does not contain a complementary pair v and $\neg v$ simultaneously. A propositional logic formula in Conjunction Normal Form (CNF) is a conjunction of clauses. A Quantified Boolean Formula (QBF) has the form

$$Q_1 x_1 \cdots Q_n x_n E(x_1, x_2, \ldots, x_n), \quad (1)$$

where $E(x_1, x_2, \ldots, x_n)$ is a propositional logic formula in CNF involving Boolean variables $x_1, x_2, \ldots, x_n$, and every Q_i $(1 \leq i \leq n)$ is either an existential quantifier $\exists$ or a universal quantifier $\forall$. A variable is existential if the restriction quantifier of the variable is $\exists$, and universal otherwise. Any Boolean variable v can take a value *true* or *false*. A truth assignment for a propositional logic formula is a map that assigns each variable a value. The satisfying assignment, called model, is the truth assignment that makes the propositional logic formula evaluated to *true*. The partial variable assignment is a map that assigns some variables values. We say a formula composed of zero clause is an empty formula, denoted by $E = \varnothing$, which is interpreted as *true*. An empty clause is a clause containing zero literal, which means that the clause is *false*.

The Quantified Boolean Formulae Satisfiability problem is to decide whether a Quantified Boolean Formula is satisfiable. For example, given a QBF $\forall x_1 \exists x_2((x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2))$, the QBF is satisfiable if and only if for each truth assignment to x_1 there exists a truth assignment to x_2. In the rest of the paper, we will use QBF to denote the formulae and the satisfiability problems.

For a QBF of the form (1), because $\exists x_1 \exists x_2 E(x_1, x_2) \equiv \exists x_2 \exists x_1 E(x_1, x_2)$ and $\forall x_1 \forall x_2 E(x_1, x_2) \equiv \forall x_2 \forall x_1 E(x_1, x_2)$, we can group in the same set all consecutive variables having the same quantifier. Therefore, equality (1) can be rewritten into the following form:

$$Q_1 X_1 \cdots Q_k X_k E(x_1, x_2, \ldots, x_n) \quad (1 \leq k \leq n), \qquad (2)$$

where X_i $(1 \leq i \leq k)$ is a set of propositional variables and all such sets are mutual disjoint. In such a format, each quantifier is applied to a set of variables rather than to a single Boolean variable. Moreover, the sequence of quantifiers alternates; an existential quantifier follows a universal quantifier and vice versa. With the quantifier restricted, each variable has a quantification level, which increases from the outermost quantification set to the innermost quantification set; that is, the variables belonging to the outermost quantification set have quantification level 1 and so forth.

3. The Relationship between SAT and QBF

The SAT problem is a typical NP-complete problem, which has a close relationship with QBF problem. More specifically, SAT can be regarded as an especial form of QBF with only existing existential quantifier, and QBF can be regarded as an extended form of SAT with additional prefixes. In addition, they are both prototypical problems for complexity classes. SAT is an NP-complete problem while QBF is a PSPACE-complete problem, which is inherently more difficult than SAT. In the following, we will discuss the similarities and dissimilarities between QBF and SAT.

Orders of Disposing Variables. When dealing with SAT, we can select variables in a random order. While dealing with QBF, we must select variables in a sequence of quantifier level that is, the variables whose quantifier level is minimal will be assigned first and so on. As a matter of fact, the reason of doing so is that the variables in QBF have a dependency relationship. For example, given a QBF $\forall x_1 \exists x_2((x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2))$, the value of an existential x_2 that makes the formula true may depend directly on the value of a universal x_1 which is quantified prior to x_2. Therefore, the key of dealing with QBF is to select the variables in the sequence of quantifier level.

Search Space. The search space of SAT is a tree while the search space of QBF is an AND/OR tree. They both regard the assignment of variables as weights and variables as nodes. The difference is that QBF has two types of nodes: AND node and OR node. In the AND/OR tree, the universal variables correspond to the AND nodes and the existential variables correspond to the OR nodes.

Hardness of the Problems. The QBF is more difficult than SAT. As we know, SAT is NP-complete while QBF is PSPACE-complete. For a SAT, the algorithm will stop when a satisfying assignment is found. However, for a QBF, the algorithm needs to continue searching because of the universal quantifiers.

Because of the close relationship between SAT and QBF, most of methods for QBF can be regarded as a generalization of algorithms commonly for SAT. Therefore, we can solve QBF with equality reduction and the hyperbinary resolution which is really effective in solving SAT.

4. QBF Algorithm Framework

Recently, search-based QBF solvers are based on an extension to the DPLL algorithm. Now we give the DPLL algorithm for QBF in Algorithm 1. In the algorithm, $C_\varnothing$ is an empty clause and $C_{\text{all}\forall}$ is a clause composed of only universal variables. The DPLL algorithm works on the principle of preprocessing, choosing branches, branched treatment, and then recursively solving the simplified QBF. In preprocessing process (line 1), DPLL employs several reasoning rules to simplify QBF like pure literal rule, unit literal rule, and so forth. In choosing branches (variables) process (line 4), DPLL selects variables in a fixed order; that is, the variables in the outermost quantification set are chosen first and so on. In branched treatment process (lines 5–8), DPLL searches AND/OR tree depending on the chosen variable. If the chosen variable is an existential variable, an OR node is explored; otherwise an AND node is explored. DPLL terminates when the simplified QBF is an empty formula (line 2), which means the input formula $Q \cdot E$ is satisfiable; when the simplified QBF contains an empty clause or there is a clause made of all universally quantified variables (line 3), which means the input formula $Q \cdot E$ is unsatisfiable.

In Algorithm 2, we present the basic algorithm of EHSPQBF, which is based on DPLL algorithm. At first, we explain the terms occurred in the algorithm. The term $C_\varnothing$ is an empty clause, and $C_{\text{all}\forall}$ is a clause composed of only universal variables. In preprocessing process (line 1), we simplify the input formula in CNF employing unit literal rule, the equality reduction, and the hyperbinary resolution. In choosing branches process (line 9), we use survey propagation as a heuristic. By giving exact information,

Procedure DPLL $(Q \cdot E)$
(1) preprocess $(Q \cdot E)$;
(2) **if** $E = \varnothing$ **then return** SAT;
(3) **if** $(C_{\varnothing} \in E) \vee (C_{\text{all}\forall} \in E)$ **then return** UNSAT;
(4) $v \leftarrow$ choosevariable $(Q \cdot E)$;
(5) **if** (v is existential)
(6) **then return** DPLL $(Q \cdot E|_{v=\text{true}})$ or DPLL $(Q \cdot E|_{v=\text{false}})$
(7) **if** (v is universal)
(8) **then return** DPLL $(Q \cdot E|_{v=\text{true}})$ and DPLL $(Q \cdot E|_{v=\text{false}})$

Algorithm 1: DPLL for solving QBF.

Procedure EHSPQBF $(Q \cdot E)$
(1) preprocess $(Q \cdot E)$;
(2) **if** $(E = \varnothing)$ **then return** SAT;
(3) **if** $(C_{\varnothing} \in E) \vee (C_{\text{all}\forall} \in E)$ **then return** UNSAT;
(4) result = deduce();
(5) **if** (result = conflict)
(6) **then** analyze_conflict();
(7) **if** (result = satisfaction)
(8) **then** analyze_satisfaction();
(9) $v \leftarrow$ SP_choosevariable $(Q \cdot E)$;
(10) **if** (v is existential)
(11) **then return** EHSPQBF $(Q \cdot E|_{v=\text{true}})$ or EHSPQBF $(Q \cdot E|_{v=\text{false}})$
(12) **if** (v is universal)
(13) **then return** EHSPQBF $(Q \cdot E|_{v=\text{true}})$ and EHSPQBF $(Q \cdot E|_{v=\text{false}})$

Algorithm 2: EHSPQBF algorithm.

the approach makes the choice of branch more exactly, which can decrease the search space and improve the runtime. In branched treatment process (line 4), we deduce the formula using conflict reasoning rule. The result of the deduction has three values: *undetermined*, *satisfaction,* and *conflict*. If result is *conflict* (*satisfaction*), that is, E evaluates to 0 (resp., 1) under the current partial variable assignment, the algorithm analyzes the conflict (resp., satisfaction); if the result is *undetermined*, that is, E evaluates neither 0 nor 1, the algorithm continues to choose branches. In this process, we employ the conflict driven learning and satisfiability directed learning to reduce the search space.

5. Preprocessing via Reasoning Rules

Besides the unit literal rule, the preprocess stage also uses more powerful reasoning rules, that is, the hyperbinary resolution and the equality reduction [10]. As a matter of fact, the hyperbinary resolution rule is a sequence of ordinary resolution steps. Though they are of similar function in simplifying the formulae, the hyperbinary resolution reduces the space which the sequence intermediate clauses occupy. And the equality reduction rule can further reduce the size of the input formula. It first finds the equivalences, and then it substitutes each equivalent variable, which can simplify the input formula. Now we describe them in the following.

Definition 1. The hyperbinary resolution rule of inference takes as input a $m + 1$-ary clause $(l_1 \vee l_2 \vee \cdots \vee l_m \vee l)$ and m binary clauses each of the form $(\neg l_i \vee l')$ $(1 \leq i \leq m)$, where l_i $(1 \leq i \leq m)$ is existential variable; then it deduces the clause $(l \vee l')$.

The advantage of the hyperbinary resolution is that it can save the solving time. The hyperbinary resolution can generate unit clauses or binary clauses in order to simplify the input formula. In addition, if the deduced clause contains only universal variables, the formula can be directly decided the QBF instance is *unsatisfiable*. For example, consider a formula $F = \forall x \exists x_1 x_2 x_3 x_4 x_5 x_6 (x_1 \vee x) \wedge (x_2 \vee x) \wedge (x_3 \vee x) \wedge (x_4 \vee x) \wedge (x_5 \vee x) \wedge (x_6 \vee x) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4 \vee \neg x_5 \vee \neg x_6)$. If we apply the hyperbinary resolution rule to F, the rule is able to obtain a deduced clause x which contains only one universal variable, and we can decide that the formula is *unsatisfiable*. By simplifying the formula, the hyperbinary resolution rule reduces the solving time.

Definition 2. The equality reduction of a QBF formula F by a pair of equivalent literals $l_1 \equiv l_2$ is a rule of replacing l_1 by l_2 if F satisfies

(1) F contains the pair of binary clauses $(\neg l_1 \vee l_2)$ and $(l_1 \vee \neg l_2)$;
(2) the quantification level of l_1 is greater than l_2;
(3) l_1 and l_2 are both existential.

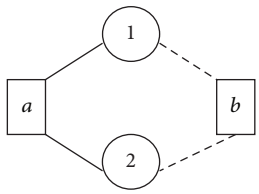

Figure 1: A simple example of a factor graph.

Note that the QBF formula F is *unsatisfiable* if there is a universal in the clause $\neg l_1 \vee l_2$. Once that equality reduction finds the equivalences, then it substitutes each variable with its definition in the formula. For example, let $F = \exists x_1 \forall x_3 x_4 \exists x_2 (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee x_4)$. Since literals x_1 and x_2 are equivalent literals, we obtain the simplified QBF. F' is $\exists x_1 \forall x_4 (x_1 \vee \neg x_4) \wedge (x_1 \vee x_4)$ by using equality reduction. Notice that when the hyperbinary resolution rule applies to F', the input formula can be further simplified.

Above all, we can find that the hyperbinary resolution rule and equality reduction have the power to reduce the size of the formula. They cannot only occupy less space but also increase the efficiency for solving QBF.

6. Choosing Branches with Survey Propagation

The survey propagation (SP) [11] method is a groundbreaking development for solving SAT. The occurrence makes the SAT problems that we can deal with scale from with ten thousand variables to with one million variables. As far as we know, SP is the only algorithm successful at solving random SAT problems with one million variables and beyond in near-linear time in the hardest region. In this section, we will discuss how to choose branches with the SP method. As a message passing algorithm, SP passes message on a factor graph which provides an easy graphical description to the message passing procedures. In the following, we will address the factor graph in detail.

6.1. Factor Graph. The factor graph is an undirected graph (see Figure 1). It has two types of nodes: one represents variables called "variable node" (circles in Figure 1) and the other represents clauses called "function node" (squares in Figure 1). A function node a is connected to a variable node i by an edge whenever the variable i (or its negation) appears in the clause a. In the graphical representation, we use a full (dashed) line between a and i whenever the variable appearing in the clause is i ($\neg i$). For every variable node i, we denote by $V(i)$ the set of function nodes to which it is connected, by $V_+(i)$ the subset of consisting of function nodes where the variable appears unnegated, and by $V_-(i)$ the complementary set of $V_+(i)$. $V(i) \setminus b$ denotes the set $V(i)$ without node b. Similarly, for each function node a, we denote by $V(a) = V_+(a) \cup V_-(a)$ the set of neighboring variable nodes. For example, given a QBF $\forall x_1 \exists x_2 (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$, we can represent the propositional logic formula with factor graph in Figure 1, where clause a represents $x_1 \vee x_2$ and clause b represents $\neg x_1 \vee \neg x_2$.

6.2. Message Passing. The survey propagation algorithm passes two types of messages: one passed from a function node to a variable node; the other passed from a variable node to a function node. The update of both messages is realized by the message update rule which is defined as follows.

(1) The message passed from a function node a to a variable node i: the message, called "survey", is a real number $\eta_{a \to i} \in [0, 1]$:

$$\eta_{a \to i} = \prod_{j \in V(a) \setminus i} \left[\frac{\Pi^u_{i \to a}}{\Pi^u_{i \to a} + \Pi^s_{i \to a} + \Pi^0_{i \to a}} \right]. \tag{3}$$

If $V(a) \setminus i$ is empty, then $\eta_{a \to i} = 1$. In equality (3), the variable j sends a particular symbol to clause a saying that the variable cannot satisfy the clause ("u"), that the variable can satisfy the clause ("s"), or that it is indifferent ("0").

(2) The message passed from a variable node i to a function node a: each variable $I \in V$ passes a triplet of real numbers $\Pi_{i \to a} = (\Pi^u_{i \to a}, \Pi^s_{i \to a}, \Pi^0_{i \to a})$ to each of its clause neighbors $a \in V(i)$:

$$\begin{aligned} \Pi^u_{i \to a} &= \left[1 - \prod_{b \in V^u_a(i)} (1 - \eta_{b \to i}) \right] \prod_{b \in V^s_a(i)} (1 - \eta_{b \to i}), \\ \Pi^s_{i \to a} &= \left[1 - \prod_{b \in V^s_a(i)} (1 - \eta_{b \to i}) \right] \prod_{b \in V^u_a(i)} (1 - \eta_{b \to i}), \\ \Pi^0_{i \to a} &= \prod_{b \in V(i) \setminus a} (1 - \eta_{b \to i}), \end{aligned} \tag{4}$$

where $v^u_a(i)$ and $v^s_a(i)$ denote neighbors which tend to make variable i dissatisfy or satisfy clause a. If a set like $v^s_a(i)$ is empty, the corresponding product takes value 1. In the above equalities, the variable i sends a particular symbol to clause a saying that the variable cannot satisfy the clause ("u"), that the variable can satisfy the clause ("s"), or that it is indifferent ("0").

When SP converges to a fixed point set of messages $\eta^*_{a \to i}$, one can use it to estimate the statistic characteristic (called bias) of every variable. There are two types of biases: positive bias and negative bias. The positive (negative) bias is the probability that variable i is restricted to 1 (0). The computing formulae are as follows:

$$\begin{aligned} W^{(+)}_i &= \frac{\widehat{\Pi}^+_i}{\widehat{\Pi}^+_i + \widehat{\Pi}^-_i + \widehat{\Pi}^0_i}, \\ W^{(-)}_i &= \frac{\widehat{\Pi}^-_i}{\widehat{\Pi}^+_i + \widehat{\Pi}^-_i + \widehat{\Pi}^0_i}, \end{aligned} \tag{5}$$

where

$$\widehat{\Pi}_i^+ = \left[1 - \prod_{a \in V_+(i)} \left(1 - \eta^*_{a \to i}\right)\right] \prod_{a \in V_-(i)} \left(1 - \eta^*_{a \to i}\right),$$

$$\widehat{\Pi}_i^- = \left[1 - \prod_{a \in V_-(i)} \left(1 - \eta^*_{a \to i}\right)\right] \prod_{a \in V_+(i)} \left(1 - \eta^*_{a \to i}\right), \quad (6)$$

$$\widehat{\Pi}_i^0 = \prod_{a \in V(i)} \left(1 - \eta^*_{a \to i}\right).$$

As a whole, SP gathers the statistical information which can guide the choosing branches. In choosing branches process, we select branches obeying quantification orders. For variables in the same quantification level, we compute positive and negative biases with SP. Then we fix the biased variable in the same quantification level; that is, one variable has the largest bias $|W_i^{(+)} - W_i^{(-)}|$. Having selected the variable, we assign the variable depending on the current quantification level. If the current quantification level is existential, we assign the biased value to the variable; otherwise we assign the reverse biased value to the variable.

7. Branched Treatment

After choosing branches, we can perform branched treatment on the chosen variable. The purpose of treatment is to decide whether the current branch of search tree is satisfiable. In branched treatment process, we employ some efficient technologies, such as conflict reasoning, conflict driven learning, and satisfiability directed learning.

7.1. Conflict Driven Learning. Conflict driven learning utilizes the knowledge learned from failures in certain search space to help prune search in future spaces. If the current branch of the search tree is not satisfied, we can perform conflict driven learning. It is carried out by the analyze conflict routine in Algorithm 3.

The analyze conflict routine analyzes the current status and brings the search to a new space by backtracking. At first, we explain some terms in the routine. As an algorithm based on DPLL, EHSPQBF is a branch and search procedure. Each branch has a decision level (dL). The first branch has decision level 1 and so forth. And variables implied by a decision variable will have the same decision level as the decision variable. Now we introduce the analyze conflict routine. At the beginning, it decides whether there exist conflict clauses in the current QBF formula (line 1). If they exist, we utilize function gen_clause() for generating learned clause (line 2). The learned clause can guarantee that the decision level which the algorithm backtracks to is unique. Then function add_clause() performs conflict driven learning when the result of deduction is *conflict*, which causes a learned clause constructed and added to the database (line 3). At last, we decide whether the decision level that the algorithm backtracks to 0 (line 4). If it is not 0, EHSPQBF will backtrack to the decision level (line 5). The aim of conflict driven learning is to memorize the conflict branch. If search space has the same branch in the further, we can prune the branch directly, which need not go on searching.

```
Routine analyze_conflict()
(1)  conflictcl = find_conflict_clause();
(2)  newcl = gen_clause (conflict_cl);
(3)  add_clause (new_cl);
(4)  if (dl (new_cl) ≠ 0)
(5)  then backtrack (dl(new_cl));
(6)  end
```

Algorithm 3: Analyze conflict routine.

```
Routine analyze_SAT()
(1)  entity = find_sat_entity();
(2)  if (entity = NULL)
(3)  then entity = gen_sat_induced_entity();
(4)  if (!termnate_condition (entity))
(5)  then newen = consensus_gen_entity (entity);
(6)  add_newentity (newen);
(7)  if (dl (newen) ≠ 0)
(8)  then backtrack (dl (newen));
(9)  end
```

Algorithm 4: Analyze SAT routine.

7.2. Satisfiability Directed Learning. Satisfiability directed learning utilizes the knowledge learned from satisfactions to reduce the number of satisfying leaves which need to be visited. When the result of deduction is *satisfaction*, we can perform satisfiability directed learning. It is carried out by the analyze SAT routine in Algorithm 4. At first, we show relevant definitions used in the routine.

Definition 3. (extended CNF) Let $E(x_1, \ldots, x_n)$ be a CNF. An extended CNF is $E' = E(x_1, \ldots, x_n) \vee EN_1 \vee \ldots \vee EN_m$, where EN_i $(1 \le i \le m)$ is an entity, that is, a conjunction of literals.

Now let us introduce the detail of the routine. At the beginning, it judges whether there exist satisfying entities in the current QBF in extended CNF (line 1). If they exist, the routine goes to line 4; otherwise, *a* entity from the current variable assignment is generated by function gen_sat_induced_entity() (lines 2-3). In lines 4-5, a learned entity is not generated by function consensus_gen_entity() until the generated entity meets the following conditions. (1) Among all its universal variables, one and only one has the highest decision level, supposing the variable is V; (2) V is at a decision level with a universal variable as the decision variable; (3) all existential literals with quantification level smaller than V's are assigned *true* before V's decision level. The decision level of universal variable V is the decision level which the algorithm will backtrack to. Then function add_newentity() performs satisfiability directed learning (line 6). It occurs when the result of deduction is *satisfaction*, which causes a learned entity constructed and added to the database. At last, we decide whether the decision level of variable V is

Table 1: Comparisons of EHSPQBF-EQR, EHSPQBF-HBR, and EHSPQBF.

Instances	EHSPQBF-EQR	EHSPQBF-HBR	EHSPQBF
(3, 60, 80, 5, 0)	10.470	15.390	**10.209**
(3, 60, 80, 5, 1)	56.342	41.230	**40.725**
(3, 60, 80, 5, 2)	230.230	153.690	**153.436**
(3, 60, 80, 5, 3)	1.783	3.467	**1.659**
(3, 60, 80, 5, 4)	98.456	132.678	**87.513**
(3, 60, 80, 5, 5)	12.479	30.745	**11.873**

0 or not (line 7). If it is not 0, EHSPQBF will backtrack to the decision level of variable V (line 5). The aim of satisfiability directed learning is memorizing the satisfied branch. If search space has the same branch in the further, we can decide the branch is satisfiable directly and go on searching the other branch of the universal variable.

8. Experimental Evaluation

In this section, to test the performance of the solver EHSPQBF, three algorithms are compared. Our solver EHSPQBF is written in C++. All experiments are carried out on a DELL PowerEdge 2650 computer with 2 Intel Xeon 2.00 GHz CPU and 1G RAM running RedHat ES3.0. Table 1 presents the results of the comparisons of the solvers. In the table, EHSPQBF-HBR is the solver that removes the hyperbinary resolution rule from EHSPQBF; EHSPQBF-EQR is the solver that removes the equality reduction rule from EHSPQBF. All instances are generated by a random generator. (3, 60, 80, 5, 0) denotes a formula indexing 0 that contains 3 quantification levels, 60 variables, 80 clauses, and each clause length is not more than 5.

From the results of the EHSPQBF, we can see that both hyperbinary resolution rule and equality reduction rule contribute much compared with the results of EHSPQBF-EQR and EHSPQBF-HBR. Consequently the run-time of the algorithms decreases. Therefore, by introducing the hyperbinary resolution rule and equality reduction rule, EHSPQBF algorithm increases the efficiency for solving QBF problem.

9. Conclusions

In this paper, we present a novel QBF solver EHSPQBF. In preprocessing, we adopt the equality reduction and the hyperbinary resolution to simplify the formulae. When choosing branches, the system uses survey propagation to select next branch; when handling branches, the system employs conflict reasoning and conflict driven learning as well as satisfiability directed learning together. Experimental results show that, as a heuristic, SP can choose branches effectively. And other techniques such as conflict driven learning also speed up the solving process. As a whole, EHSPQBF is an efficient QBF solver.

Acknowledgments

This work was fully supported by the National Natural Science Foundation of China (Grant nos. 11226275, 61103136, and 61070084), the Fundamental Research Funds for the Central Universities (Grant no. 11QNJJ006), the Ministry of Education (Grant no. 20120043120017), the Postdoctoral Science Foundation of China (2012M520658), and the opening fund of top key discipline of computer software and theory in Zhejiang provincial colleges at Zhejiang Normal University (Grant no. ZSDZZZZXK37).

References

[1] J. Rintanen, "Asymptotically optimal encodings of conformant planning in QBF," in *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pp. 1045–1050, July 2007.

[2] P. Marin, C. Miller, and B. Becker, "Incremental QBF preprocessing for partial design verification," in *Proceeding of the 15th International Conference of Theory and Applications of Satisfiability Testing*, pp. 473–474, 2012.

[3] U. Egly, T. Eiter, H. Tompits, and S. Woltran, "Solving advanced reasoning tasks using quantified Boolean formulas," in *Proceeding of the 12th AAAI Conference on Artificial Intelligence*, pp. 417–422, 2000.

[4] G. Pan and M. Y. Vardi, "Optimizing a BDD-based modal solver," in *Proceedings of the 19th International Conference on Automated Deduction*, pp. 75–89, August 2003.

[5] I. P. Gent, H. H. Hoos, A. G. D. Rowley, and K. Smyth, "Using stochastic local search to solve quantified Boolean formulae," in *Proceeding of the Principles and Practice of Constraint Programming*, pp. 348–362, 2003.

[6] I. Gent, E. Giunchiglia, M. Narizzano, A. Rowley, and A. Tachella, "Watched data structures for QBF solvers," in *Proceeding of the 6th International Conference on Theory and Applications of Satisfiability Testing*, pp. 348–355, 2003.

[7] E. Giunchiglia, M. Narizzano, and A. Tacchella, "QuBE: a system for deciding quantified Boolean formulas satisfiability," in *Proceeding of the 17th International Joint Conference on Artificial Intelligence*, pp. 18–23, 2001.

[8] J. Rintanen, "Partial implicit unfolding in the Davis Putnam procedure for quantified Boolean formulae," in *Proceeding of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, pp. 362–376, 2001.

[9] L. Zhang and S. Malik, "Towards a symmetric treatment of satisfaction and conflicts in quantified Boolean formula evaluation," in *Proceeding of the Principles and Practice of Constraint Programming*, pp. 200–215, 2002.

[10] F. Bacchus, "Enhancing Davis Putnam with extended binary clause reasoning," in *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI '02)*, pp. 613–619, August 2002.

[11] M. Mézard, G. Parisi, and R. Zecchina, "Analytic and algorithmic solution of random satisfiability problems," *Science*, vol. 297, no. 5582, pp. 812–815, 2002.