Solving QBF with Heuristic Small World Optimization Search Algorithm

Tao Li[1] and Nanfeng Xiao[2]
[1]Modern Education and Technology Center, South China Agricultural University, China
[2]School of Computer Science and Engineering, South China University of Technology, China

Abstract: *In this paper, we use gaifman graph to describe the topological structure of the Quantified Boolean Formulae (QBF), we mainly study the formula family with the Small World (SW) network topology. We analyze the traditional Davis, Putnam, Logemann and Loveland (DPLL) solving algorithm for QBF, then we improve the DPLL algorithm and propose the solving algorithm framework based on Small World Optimization Search (SWOS) algorithm, we apply this SWOS algorithm to determine the order of the DPLL branch variable. Our result proves that SWOS algorithm has a certain degree of effectiveness to improve the solving efficiency. It is valuable as an incomplete solution algorithm for search-based solver.*

Keywords: *QBF, SW, search algorithm, optimization algorithm.*

Received July 26, 2012; accepted February 11, 2013; published online August 17, 2014

1. Introduction

Boolean Satisfiability (SAT) solvers have become powerful enough to solve many practically relevant problems and they are currently used in numerous industrial tools for circuit and software verification. Building upon this success, the research community has begun to consider the more general, but also more complicated Quantified Boolean Formula (QBF) domain. This allows researchers to encode problems encountered in black box or partial circuit verification, bounded model checking and AI planning more naturally and compactly than in SAT. However, since QBF problems are generally more difficult (PSPACE-Complete vs. NP-Complete), they require dedicated algorithms and increased computation power to solve relevant instances [15]. In this context, the use of different heuristic and especial algorithms is a possible and interesting solution.

In fact, nearly all effective QBF solvers are found on Davis, Putnam, Logemann and Loveland (DPLL) algorithms [6] the main factor affect the efficiency of algorithm is the choice of branch variable. For this reason, the researcher adopt various approach to decide the branch choice, such as random method, walksat heuristic and survey propagation [9, 29]. This paper highlights this problem and provides new heuristic algorithm to improve the branch choice within the DPLL algorithm framework. We use the graph structure to research QBF solving, some QBF which have a Small World (SW) network structure are very difficult to resolve [22, 23]. For the purpose of improving the solution efficiency for this particular type of formula, we develop a heuristic Small World Optimization Search (SWOS) algorithm to seek the optimal variable and then we adopt the optimal variable as the branch variable. To our best knowledge, it is the first time to research the special QBFs having the SW network topological structure.

The paper is structured as follows: Section 2 will start with a description of the QBF problem and how QBF solvers work (section 2.1 and 2.2). In section 3 we introduce our solving architecture for QBFs. In section 4 we introduce the heuristic SWOS algorithm for branch choice. In section 5 we give some technical details and experimental results about our implementation. Section 6 will conclude this paper and giving some future research directions.

2. Preliminaries

2.1. Overview of the QBF Problem

There are many ways to encode a QBF problem [15] but, in our context, they are defined in Conjunctive Normal Form (CNF). A problem in CNF form starts with a variable definition. The variable definition quantifies each variable (either existentially or universally) and assigns each variable to a specific quantification level. Once the variable definition is complete, a set of clauses is given that defines the problem. More formally, a QBF is an expression of the form:

$$\varphi = Q_1 z_1 Q_2 z_2 \ldots Q_n z_n \Phi \quad (n \geq 0) \tag{1}$$

Here, every $Q_i (1 \leq i \leq n)$ is a quantifier, either existential $\exists$ or universal $\forall$, $z_1, \ldots, z_n$ are distinct sets of variables and Φ is a propositional formula. $Q_1 z_1, Q_2 z_2, \ldots, Q_n z_n$ is defined as the prefix and Φ, the propositional formula would contain a set P of clauses. While a variable is defined as an element of P, an occurrence of that variable or its negation in a clause is referred to as a literal. In the following, the literal $\bar{l}$ is defined as the negative occurrence of the variable $|l|$ in P and l is the positive occurrence. In the

following, we also, use true and false as abbreviations for the empty conjunction and the empty disjunction, respectively. For example, an entire problem definition might be as follows:

$$\exists x_1 \forall y \exists x_2 \begin{Bmatrix} \{\overline{x_1} \vee \overline{y} \vee x_2\} \wedge \{\overline{y} \vee \overline{x_2}\} \wedge \{x_2\} \\ \wedge \{x_1 \vee \overline{y}\} \wedge \{y \vee x_2\} \end{Bmatrix} \quad (2)$$

We say that (1) is in CNF when Φ is a conjunction of clauses, where each clause is a disjunction of literals as shown in Equation 2 and that Equation 1 is in Disjunctive Normal Form (DNF) when Φ is a disjunction of cubes, where each cube is a conjunction of literals1. We use constraints when we refer to clauses and cubes indistinctly. We also define:

1. The level of a variable z_i, to be 1+the number of alternations $Q_j z_j, \ldots, Q_{j+1} z_{j+1}$ in the prefix with $j \geq i$ and $Q_j \neq Q_{j+1}$.
2. The level of a literal l is the level of $|l|$.
3. The level of the formula 1 is the level of z_1.
4. So for example, in Equation 2, x_2 is existential and is quantified on level 1, y is universal and is on level 2, x_1 is existential and is on level 3.

QBF solvers are interested in answering the question of whether or not Equation 1 ϕ expresses a true or false assertion, i.e., whether or not ϕ is true or false. The reduction of a CNF formula Φ by a literal l is the new CNF $\Phi|_l$ which is Φ with all clauses containing l removed and $\overline{l}$ the negation of l, removed from all remaining clauses. For example, let $\varphi = \forall x z \exists y(\bar{y}, x, z) \wedge (\overline{x}, y))$, then $\varphi|_x = \forall z \exists y(\bar{y}, z)$. The semantics of a QBF can be defined recursively in the following way:

1. If Φ is the empty set of clauses then ϕ is true.
2. If Φ contains an empty clause then ϕ is false.
3. $\forall v \phi$ is true if both $\phi|_v$ and $\phi|_{\neg v}$ are true.
4. $\exists v \phi$ is true if at least one of $\phi|_v$ and $\phi|_{\neg v}$ is true.

In this paper, we only study the formula with conjunctive normal form.

2.2. QBF Solver

The sequential QBF solvers usually apply one single algorithm to resolve the whole formulae set at runtime. There are many sequential QBF solvers [18, 21]. Most solvers like QuBE, yQuaffle and sSolve are in principal based on the DPLL algorithm [2, 6, 10, 11, 20]. Others, like Quantor or Nenofex [4, 12] try to resolve and expand the formula until no universally quantified variables remain. This allows them then to send their remaining, existentially quantified problem to a SAT solver. This works well on many problems, but it can result in an explosion with respect to the size of the formula. On the other hand, solvers like sKizzo [3] do the opposite of Quantor [1] and use symbolic skolemization to eliminate all the existentially quantified variables in the formula. Some so-called incomplete solvers are also based on stochastic search methods and they can be very effective in solving some categories of problems, but are not able to prove the value of unsatisfiable formulas. A few alternative algorithms for QBF are emerging, e.g., and-Inverter Graphs. Their usage in QBF satisfiability algorithms have been explored at least in [19]. Finally, AQME [21] is a portfolio of solvers considered and the best one is selected using machine learning techniques [25]. In the parallel solving domain, there exist three implementations of parallel solvers for the problem of validity of QBF: PQSOLVE, PaQube and QMiraXT [8, 15, 16].

2.3. SW Network

In mathematics, physics and sociology, a SW network is a type of mathematical graph in which most nodes are not neighbors of one another, but most nodes can be reached from every other by a small number of hops or steps. Specifically, a SW network is defined to be a network where the typical distance L between two randomly chosen nodes (the number of steps required) grows proportionally to the logarithm of the number of nodes N in the network, that is [27]:

$$L \propto log\ N$$

To formalize the notion of a SW, Watts and Strogatz define the clustering coefficient and the characteristic path length. The path length is the number of edges in the shortest path between two nodes. The characteristic path length L is the path length averaged over all pairs of nodes. The clustering coefficient is a measure of the cliqueness of the local neighborhoods. For a node with k neighbors, then at most $k(k-1)/2$ edges can exist between them (this occurs if they form a k-clique). The clustering of a node is the fraction of these allowable edges that occur. The clustering coefficient C is the average clustering over all the nodes in the graph.

Watts and Strogatz define a SW graph as one in which $L \geq L_{rand}$ and $C >> C_{rand}$ where, L_{rand} and C_{rand} are the characteristic path length and clustering coefficient of a random graph with the same number of nodes n and edges e. Rather than this simple qualitative test, it might be useful to have a quantitative measure of "small worldliness". We can then compare the topology of different graphs. To this end, we define the proximity ratio μ as the ratio of C/L normalized by C_{rand}/L_{rand}. In graphs with a SW topology, the proximity ratio $\mu >> 1$. By comparison, the proximity ratio μ is unity in random graphs and small in regular graphs like lattices.

3. Solving Architecture

3.1. Problem Model

Given a QBF φ on the set of variables $Z=\{z_1, \ldots, z_n\}$, its gaifman graph has a vertex set equal to Z with an edge (z, z') for every pair of different elements $z, z' \in Z$

that occur together in some clause of φ. A scheme for a QBF φ having prefix P is a supergraph (Z, E) of the gaifman graph of φ along with an ordering $z_1', ..., z_n'$ of the elements of Z such that:

1. The ordering $z_1', ..., z_n'$ preserves the order of P, i.e., if $i< j$ then z_j' comes after z_i' in P.
2. For any z_k', its lower numbered neighbors form a clique, that is, for all k , if $i< k, j< k$, $(z_i', z_k') \in E$ and $(z_j', z_k') \in E$, then $(z_i', z_j') \in E$ [20, 23].

In Figure 1, we show the gaifman graph corresponding to the structure of Equation 3. The graph is comprised of five nodes and there is an edge between all the variables occurring together in some clause of 3.

$$\begin{aligned} &\forall y_1 \exists x_1 \forall y_2 \exists x_2 \exists x_3 ((y_1 \vee y_2 \vee x_2) \wedge \\ &(y_1 \vee \neg y_2 \vee \neg x_2 \vee \neg x_3) \wedge \\ &(y_1 \vee \neg x_2 \vee x_3) \wedge (\neg y_1 \vee x_1 \vee x_3) \wedge \\ &(\neg y_1 \vee y_2 \vee x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge \\ &(\neg y_1 \vee \neg x_1 \vee \neg y_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3)) \end{aligned} \quad (3)$$

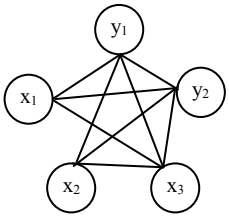

Figure 1. Gaifman graph of the QBF (3).

Pulina *et al.* [21, 23] found there are some formulas families are very difficult to solve, at least by search-based solvers. The author investigated this phenomenon by studying the purely propositional structure of the gaifman graphs of these formulas, with respect to the original formulas, these graphs have an increased clustering coefficient C and a decreased average path length L, i.e., these formulas have SW topology, in a propositional sense. This means that there are clusters of densely connected variables, with occasional cross-cluster connections. It is an important conclusion that the more the structure of the formula resembles a SW, the more it is difficult to solve [5].

The reason for this phenomenon is explained in [26, 27]. In a graph with a SW topology, nodes are highly clustered yet the path length between them is small. Such a topology can make search problems very difficult since local decisions quickly propagate globally. It shows that graphs associated with many different search problems have a SW topology and that the cost of solving such search problems can have a heavy tailed distribution. The strategy of randomization and restarts appears to eliminate these heavy tails. A novel restart schedule in which the cutoff bound is increased geometrically appears particularly effective.

In this paper, we develop a heuristic SWOS algorithm to seek the optimal variable in gaifman graphs and then we adopt the optimal variable as the branch variable, then we can use the DPLL algorithm framework to solve the formulas. In this way, we can improve the solve efficiency to some extent.

3.2. Solving Algorithm Framework

The DPLL algorithm is the most efficient algorithm in SAT solving domain. At the present time, almost all the QBF solver is designed on the base of DPLL algorithm. The basic framework of DPLL algorithm for solving QBF problem is showed in Algorithms 1 [6, 29].

In Algorithms 1, C_ϕ is the empty clause, $C_\forall$ is the clause composed of universal literals, l_v is the literal corresponding to the variable v, $l_\exists$ is a set of existential literals, $l_\forall$ is a set of universal literals, $Q.E|_{v=true}$ is the formula set through assigning a true value to variable v, $Q.E|_{v=fulse}$ is the formula set through assigning a false value to variable v. In the preprocess stage, the DPLL algorithm uses some inference rules, such as pure literal rule, unit literal rule to simplify the QBFs, thus, we can decide whether the conditions for terminating the algorithm is satisfied. If the formula set is empty after simplification, the original formula is satisfiable; otherwise, if the formula set contains the empty clause or contains a clause composed by universal literals merely, the original formula is unsatisfiable. If we can not decide whether the original formula is satisfiable or unsatisfiable, then we split the and/or tree according to the variable constraint by the outermost layer quantifier, in this way, we get two formula collection $Q.E|_{v=true}$ and $Q.E|_{v=fulse}$ If the variable which we choose to split the and/or tree is an existential literals, the original formula is satisfiable at least one of $Q.E|_{v=true}$ and $Q.E|_{v=fulse}$ is satisfiable. If the variable which we choose to split the and/or tree is a universal literals, the original formula is satisfiable only if both $Q.E|_{v=true}$ or $Q.E|_{v=fulse}$ are satisfiable. From the procedures of DPLL algorithm, one of the key factor affecting the entire algorithm efficiency is how to choose an appropriate variable for splitting the and/or tree.

We design the SW algorithm for solving Small World_QBF (SW_QBF) algorithm on the base of DPLL algorithm, the algorithm framework is showed in Algorithms 2. The notations of the SW_QBF algorithm are same to the DPLL algorithm. C_ϕ denotes the empty clause, $C_\forall$ denotes the clause composed all by universal literals, $Q.E|_{v=true}$ is the formula set through assigning a true value to variable v, $Q.E|_{v=false}$ is the formula set through assigning a false value to variable v. In the preprocess stage, the SW_QBF algorithm uses some pure literal rule to simplify the QBFs. In the branch selection stage, SW_QBF algorithm adopts the SWOS algorithm as the heuristic (SWOS_Choosevariable()), it determines the branch variable through providing the global information, reduces the searchspace, thereby decreases the algorithm's rollback times, we will introduce the

SWOS algorithm in chapter 4. In the branch process stage, we apply the conflict reasoning rules to infer the QBFs, this procedure will produce three return value, they are respectively conflict, satisfaction and undetermined. If the original formula *E* has false value on the current assignment, the algorithm returns conflict, then continue the conflict driven learning. If the original formula *E* has true value on the current assignment, the algorithm returns satisfaction, then continue the satisfiability directed implication learning. If the value of formula *E* can not be determined on the on the current assignment, the algorithm returns undetermined, then continue the branch selection. In this stage, we use some reasoning technology such as conflict driven learning and satisfiability directed implication learning to decrease the searchspace, accelerate the problem solving.

- SubFunction *DPLL*(*Q.E*)

Algorithms 1: DPLL algorithm

1. *Preprocess (Q.E).*
2. *if E=Ø then return SAT.*
3. *if* ($C_{Ø} \in E$) $\vee$ ($C_{all\forall} \in E$) *then return UNSAT.*
4. *v←choosevariable(Q.E).*
5. *if* ($l_v \in l_{\exists}$)
6. *then return DPLL(Q.E*$|_{v=true}$*) or DPLL(Q.E*$|_{v=false}$*)*
7. *if* ($l_v \in l_{\forall}$)
8. *then return DPLL(Q.E*$|_{v=true}$*) and DPLL(Q.E*$|_{v=false}$*)*

- SubFunction *SW_QBF*(*Q.E*)

Algorithms 2: Solving QBF with SWOS algorithm

1. *preprocess(Q.E).*
2. *if* ($E = \varnothing$) *then return SAT.*
3. *if* ($C_{Ø} \in E$) $\vee$ ($C_{all\forall} \in E$) *then return UNSAT.*
4. *result=deduce();*
5. *if (result=conflict)*
6. *then analyze_conflict().*
7. *if (result=satisfaction)*
8. *then analyze_satisfaction();*
9. *v←SWOS_Choosevariable(Q.E).*
10. *if* ($l_v \in l_{\exists}$)
11. *then return SW_QBF(Q.E*$|_{v=true}$*) or SW_QBF(Q.E*$|_{v=false}$*)*
12. *if* ($l_v \in l_{\forall}$)
13. *then return SW_QBF(Q.E*$|_{v=true}$*) and SW_QBF(Q.E*$|_{v=false}$*)*

4. Branch Variable Choice Based on SWOS Algorithm

We can explain the basic thinking of branch choice (subfunction SWOS_Choosevariable) which is based on SWOS algorithm as followed [28].

We adopt the gaifman graph showed in Figure 1 to indicate the structure of QBFs, each variable is the node in SW network. We apply the SWOS algorithm in the SW network, assign the variable group in improved SW structure, each variable individual can get more information from other individual, in this way and thus, we can achieve the goal which seeks the optimal branch variable. We take the variable found in searching each time as the branch variable chosen at that time.

4.1. Algorithm Analysis

In the process of our algorithm, there are three type of variable, they are respectively discoverer variable, pursuer variables and patrolman variables.

In each iterative searching, the variable individual which has the optimum fitness in variable group is chosen as the discoverer variable, other variables are divided into pursuer variables and patrolman variables. In our SW search algorithm, each variable individual has it own angle of aspect, and the angle of aspect will be updated in very iteration. The discoverer variable and patrolman variables implement the mechanism of angle searching. The discoverer variable inspects three position variables nearby itself through rotating its searching angle, thus, it is expecting finding a better place. The patrolman variables choose a random orientation to execute a local search through rotating its searching angle randomly, thus it increases the diversity of variables group. The pursuer variables do not implement the mechanism of angle searching, but it close with the discoverer variable directly.

It is supposed that the i^{th} variable individual in the group is pursuer variables when the algorithm implements the k^{th} iteration, then the location update method of variable *i* is explained in Equation 4:

$$X_i^{k+1} = X_i^k + r_3 (X_p^k - X_i^k) \tag{4}$$

In Equation 4, X_i^k is the current location of variable individual *I*, X_i^{k+1} is its location after update, r_3 is a random number which distributes uniformly ranged from 0 to 1. In our algorithm, the pursuer variables have the largest number in the variables group, so the update mechanism of pursuer variables affects the algorithm performance mostly.

In this paper, we introduce the influence of distance on communication into the small world model, i.e., one variable node decide the probability *P* of adding the edge through considering the distance between itself and other variable. At the same time, the value of *P* will increase with the increment of iteration times. In this paper, the value of *P* decreased as an exponential function with the increment of distance among the variable nodes, simultaneously, the value of *P* increases linearly with increment of iteration times.

In swarm intelligence algorithm, we look on the variable group as a network, each variable is a node in the network. These variable nodes implement co-learning and coevolution through sharing information each other, in this way, the variable group achieves the collective goal. Just like the animal hunting in reality, the animal which join the hunting not only pay attention to the discoverer, but also watch the other individual's behavior in its view. In this mode, it has a

great probability to watch the surrounding companion, at the same time, it also has a smaller probability to watch the distant companion. So, we make use of this idea in our search algorithm.

It is supposed that we implement searching in a group having M variable in an N-Dimension Space, at the K^{th} computation, the i^{th} variable is pursuer variables. At first, we compute the probability p_i^k which decide whether to connect with other variables according to the current iteration times and the distance between itself and other variables, the pursuer variables choose its conjoint variable according to p_i^k, thus it can constitute the neighborhood with itself. The probability p_i^k connecting variables i and j is showed as follows Equation 5:

$$p_{ij}^k = w_1 * \frac{k}{MaxInt} + (1 - w_1) exp(\frac{Dis_{ij}^k}{l_{max}}) \quad (5)$$

In Equation 5, $MaxInt$ is the maximum iteration time, Dis_{ij}^k is the Euclidean distance between variables i and j, w_1 is a weight coefficient ($0 \leq w_1 \leq 1$), l_{max} is the maximum step-size in search, l_{max} can be computed in Equation 6:

$$l_{max} = \|U - L\| = \sqrt{\sum_{z=1}^{n} (U_z - L_z)^2} \quad (6)$$

In Equation 6, U_z is the upper bound of the z dimension's border and L_z is the corresponding lower bound.

Because the small world requires $0 \leq p_{ij}^k \leq 1$, we can know from Equation 5, each item in the expressions is nonnegative, so $0 \leq p_{ij}^k$, when $k=MaxInt$ and $Dis_{ij}^k = 0$, p_{ij}^k has the maximum value $w_1+(1-w_1)=1$, so our algorithm model satisfies the requirement of small world network.

In our algorithm, the pursuer variable connects to a certain number of other variables in accordance with the improved small world model in very iteration, in this way it structures its neighborhood, then it choose the variable which has the best fitness as the local optimum variable. The pursuer variable refers simultaneously the local optimum and global optimum.

It supposed that in the k^{th} iteration, the local optimum variable in the neighborhood of the i^{th} individual variable is $Lbest_i^k$, the value of $Lbest_i^k$ is X_{li}^k. The improved pursuer variable's position is updated in Equation 7, r_4 is a random number which uniformly distribute between 0 and 1.

$$X_i^{k+1} = X_i^k + r_3(X_p^k - X_i^k) + r_4(X_{li}^k - X_i^k) \quad (7)$$

In our algorithm, after the variable updates its position, it evaluates the fitness immediately and then compares itself with the current global optimum, if the current variable's position is better than current global optimum variable; The current global optimum variable is replaced. In this way, if the new global optimum variable is generated in the process of iteration, the variable which has not been updated will receive this message and update its position through referring the new global optimum variable.

4.2. Algorithm Processes

The process of the SWOS algorithm can be explained as followed:

- *Step 1*: All the individual variable's position X and search angle φ are initialized randomly. The algorithm computes the orientation of individual variable, give a fitness evaluation to each individual variable's position, get the average fitness R and then select the individual which has the best fitness as the discoverer variable (optimal variable).
- *Step 2*: The algorithm judges whether the terminal condition is satisfied, if the terminal condition is satisfied, then output the result, if not satisfied, go to Step 3.
- *Step 3*: Each individual variable in the candidate variable group executes the operation followed:
  - *Step 3.1*: The algorithm judge whether the individual variable is a discoverer variable, if it is, check three position and then give a fitness evaluation to these three position, if a better position is found, then jump to the better place, otherwise, keep the current position. If the discoverer variable does not find a better place for continuous α generation's iteration, then the search angle returns the value before α generation's iteration. If discoverer variable does find one, then go to step 3.2.
  - *Step 3.2*: A random number rand between 0 and 1 is generated for the variable i, if $rand < R$, then the variable i is a pursuer variable, go to step 3.3, otherwise, it is a patrolman variable, go to step 3.5.
  - *Step 3.3*: The algorithm computes the distance Dis_{ij}^k ($j \neq i$) from variable i to other variable, then computes the probability P_{ij}^k which decides whether variable i connect other variable in the current iteration according to the expression (2). A random number rand between 0 and 1 is generated for the other variable j, if $rand < P_{ij}^k$, then the variable i connects with the variable j.
  - *Step 3.4*: The variables which are connected with the variable i constitute the neighbors of variable i. Then, the algorithm selects the variable which has the best fitness as the local optimum variable $Lbest_i^k$. Thus, the variable i updates its own position according to the Equation 4, go to step 3.6.
  - *Step 3.5*: The patrolman variable updates its position.
  - *Step 3.6*: The algorithm evaluates the fitness of variable i. if the fitness of variable i is superior to the current discoverer variable, then the variable i is chosen as the new discoverer variable.
- *Step 4*: Return to step 2.

The pseudo code of the SWOS algorithm is showed in followed Algorithms 3.

- SubFunction *SWOS_Choosevariable* (*Q.E*)

Algorithms 3: Branch selection based on SWOS algorithm.

```
1. { RandomInitialization(Q.E.X, φ );
2.   Best=Evaluate(Q.E.X);
3.   While(No better)
4.    { Return discoverer;
5.      { if(Q.E.X= discoverer)
6.        { While( α )
7.          { PositionDetection();
8.            Evaluate(Q.E.X);
9.            if(better)
10.             Go the BetterPosition; }
11.         SearchingAngle( α ); }
12.       rand=Rand(i);
13.       if(rand<R)
14.        { Q.Ei= pursuer;
15.          ( Dis_ij^k , P_ij^k )=Compute();
16.          rand=Rand(j);
17.          if(rand< P_ij^k )
18.            { Connect(i,j);
19.              Lbest_i^k = Evaluate(Neighbor(i));
20.              updatePosition(i); }}
21.       else
22.         { Q.Ei= patrolman;
23.           updatePosition(i);
24.           if(Evaluate(Q.Ei)> discoverer);
25.             Q.Ei = discoverer;}}}
```

5. Experimental Evaluations

To evaluate our algorithm architecture, we ran few preliminary tests on some benchmark of QBFLIB (www.qbflib.org). All the experiments that we present hereafter ran on a single PC, running the environment is Ubuntu-11.10-desktop/GNU Linux. On all test runs the CPU time limit was set to 600 seconds.

In order to compare the solution efficiency, we choose three state-of-the-art solvers, namely a hybrid solver AQME10, two sequential solvers QuBE7 and sKizzo. AQME10 is an adaptive QBF Multi-Engine solver; it is robust and efficient than state-of-the-art single-engine sequential solvers. QuBE7 is an efficient search-based solver for QBFs. Maybe it is the best search-based solver. sKizzo is a powerful solver based on a new technique, called symbolic skolemization and on a related form of symbolic reasoning. This approach makes it differ from all the previous QBF solvers. This approach makes it differ from all the previous QBF solvers. All the solvers run the same benchmarks on the same single machine. In our experiments, we ran all the solvers with their default settings, i.e., we did not attempt to optimize any of their parameters for the problem at hand.

Our experiments are to run the solvers on different QBF encodings, with the goal of confirming the validity of the selection above. In particular, we wish to show that the encodings considered are challenging enough given the current state of the art and that the algorithms featured by the solvers are orthogonal, i.e., solvers have complementary abilities across different families.

To prove the effectiveness of our methods, we choose the QBFs family counter, C432, Debug, s3271, term1 as the test set, they are in the formal verification domain. The QBFs made up by encoding of formal verification problems represent a reasonably difficult test set. Family counter has 88 instances; we only describe the experiments result of 10 instances because of the space restraint of this paper. Family debug has 38 instances, we only choose 10 instance.

Table 1 is the cumulative CPU time to solve the benchmarks family counter (unit is second(s)). SMQBF is the solver which applies the SWOS algorithm. We can see from the table that our solver SMQBF have not much different with other three solvers. Two sequential solvers QuBE7 and sKizzo spend no time from cnt01 to cnt05, because these instances are very easy formula, the sequential can resolve them directly. But, AQME10 and our solver are hybrid solver, it must spend a little time to run the classification algorithm in order to accomplish the best algorithm selection. For the more hard formula instances cnt09 and cnt10, four solvers all spend much time, the time gap is not very obvious. We can see the similar result from Table 2. From the above benchmarks, we can see the SMQBF is almost as strong as QuBE7. But, it is slightly worse than QuBE7 because QuBE7 apply some top reasoning technology.

Table 1. Solving time for family counter of domain formal verification.

Instances	AQME10	QuBE7	sKizzo	SMQBF
cnt01	0.403	0	0	0
cnt02	0.2906	0	0	0
cnt03	0.2956	0	0	0
cnt04	0.301	0	0	0
cnt05	0.3166	0	0	0
cnt06	0.3352	0.02	0.12	0.03
cnt07	0.3483	0.05	0.34	0.05
cnt08	0.4656	0.15	0.81	0.16
cnt09	0.9728	0.63	1.22	0.68
cnt10	3.27	3.19	5.01	3.21
Total	6.9987	4.04	7.5	4.13

Table 2. Solving time for benchmarks of domain formal verification.

Benchmarks	AQME10	QuBE7	sKizzo	SMQBF
Counter(10)	6.9987	7.47	7.5	7.56
C432(8)	6.1362	7.34	6.54	7.68
Debug(10)	8.345	8.53	8.56	8.67
s3271(8)	7.829	8.12	8.09	9.32
term1(8)	7.632	7.96	7.99	8.12
Total	36.972	39.42	35.22	41.35

From the above benchmarks, our SMQBF does not have an advantage over other solvers. The reason is that the above benchmark's original QBFs structure is not close to a small world topology, the structure of these formulas is closer to a random world topology. An important fact to be mentioned is that the gaifman

graphs corresponding to the above formulas encodings are sparse. So, the solver apply the SWOS algorithm does not exceed other solvers.

The second experiment detailed in this section is carried out on the same computing platforms above, but here we focus on the following QBF encodings:

- Katz: QBFs resulting from the encoding of symbolic reachability for industrially relevant circuits (20 instances) [7].
- Tipdiam: QBFs resulting from the encoding of symbolic diameter calculation for a variety of circuits (40 instances) [13].

These two families are the formulas whose gaifman graphs have typical small world structure. Some relevant features of the above encodings are summarized Pulina and Tacchella [23] analyze these two families and gets this conclusion. The density distribution of these familie's graph structure can reach relatively high values. These two set match the small world topology more closely than the above families.

Table 3 is the cumulative CPU time to solve the benchmarks family Katz and Tipdiam, it reports the number of instance solved by four solvers (Number) and the total CPU time spent on such instances (Time). A dash on both columns means that the solver does not solve any formula. We can see that all the solvers fell hard to these two families. They all can't resolve all the formulas. The hybrid solver AQME10 is the best in other three solvers, but our solver SMQBF can resolve more numerous formulas than AQME10, even it spends less time. The largest number of instance solved by SMQBF exceed the second about one-third. It proved that heuristic SWOS algorithm is very effective in branch selection for DPLL split.

Table 3. Solving time for benchmarks of small world formula family.

Benchmarks	AQME10		QuBE7		sKizzo		SMQBF	
	Number	Time	Number	Time	Number	Time	Number	Time
Katz(20)	9	75.65	9	80.76	--	--	12	70.43
Tipdiam(100)	78	312.73	75	340.45	65	720.52	85	290.34

From the experimental data, we know that the time gap is not very obvious in formulas have random world topology. On the contrary, but for the more hard formula instances in the formulas have SW topology, the solver based on heuristic SWOS algorithm surpass other three advanced solvers. It is an important result that the more the structure of the instance resembles a SW, the more it is difficult to solve-at least by search-based solvers, but the improved SWOS algorithm can optimize the branch selection and accelerate the solution process. The search-based solvers also can overcome this difficulty and gain the most. Its relatively good performances on this dataset indicate that SWOS algorithm be the key for efficient reasoning.

In the third experiment, we use a large test set having 350 formulas, including Katz, Tipdiam and Formal Verification domain problem. Most of them are formula having a SW topology. We run the above four solvers on this special test set. We record the number of instance solved at some time node, i.e., how many formulas are solved when time pasts 10s, 100s and 200s. As shown in Figure 2, the SMQBF is in fact able to solve more instances than other solvers, because most of the test set is made up of Katz, Tipdiam, etc., SW topology formula. At the beginning, SMQBF does not show the advantage, however, as time grows, it can solve more formula than other solvers. When the time pasts 200s, Performance of AQME and QuBE are declining, performance of SMQBF is stable growth trend. This shows that it has a better robustness. Hence, by simply employing SWOS algorithm, it is possible to gain an advantage over such sophisticated QBF solvers as QuBE in some respects. But, the advantage is not obvious, the small improvement in the efficiency of SMQBF has to be weighted against the performance of QUBE, which shows that the internal preprocessing of SMQBF is adding useless overhead on these encodings. Such overhead is partially reduced by SWOS algorithm.

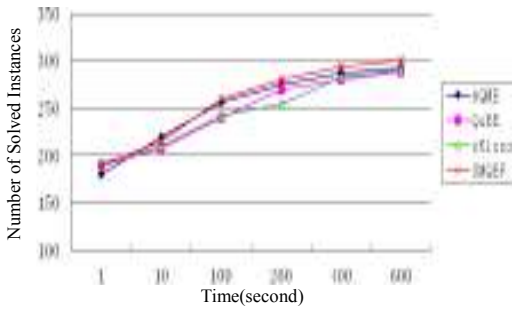

Figure 2. Time scale comparison of several solvers on small world topology benchmarks.

From the above result, it is fair to say that SMQBF alone is able to solve a fairly large number of instances, and most of these instances are those that cannot be solved by search-based solvers in their original formulation. It shows how SMQBF can be beneficial independently of the algorithm featured by the solver and after a branch selection with SWOS they are able to solve more hard formulas. The improvement is more substantial with search based engines rather than variable-elimination ones. This explains also, why SMQBF improve the performances of search-based solvers more than what happens for variable-elimination-based ones.

6. Conclusions

In this paper, we use gaifman graph to describe the topological structure of the QBF, mainly study the formula family with the SW network topology. We analyze the traditional DPLL solving algorithm for QBF and then we improve the DPLL algorithm and propose the solving algorithm framework based on SWOS algorithm, we apply this SWOS algorithm to determine the order of the DPLL branch variable. Our result also proves that SWOS algorithm has a certain degree of effectiveness to improve the solving efficiency for a particular type of formula. It is

valuable as an incomplete solution algorithm for search-based solver. The work in the future include the more accurate optimization search algorithm for SW network and develop more intelligent heuristic algorithm to strategic decision and schedule in choose the branch variable [24]. On the other hand, it is necessary to combine variable elimination and search so that they can interleave during the decision process. Maybe it can further improve the solution efficiency [14, 17].

Acknowledgment

This work is supported by the National Natural Science Foundation of China (Grant No. 61171141), Special Cooperation Research Projects for China Ministry of Education and Guangdong Province (2012B091100448).

References

[1] Baader F. and Voronkov A., "Evaluating QBFs via Symbolic Skolemization," *in Proceedings of Logic for Programming, Artificial Intelligence and Reasoning*, Montevideo, Uruguay, pp. 285-300, 2005.

[2] Balabanov V. and Jiang J., "Unified QBF Certification and its Applications," *Formal Methods in System Design*, vol. 41, no. 1, pp. 45-65, 2012.

[3] Benedetti M., "sKizzo: A Suite to Evaluate and Certify QBFs," *in Proceedings of the 20th International Conference on Automated Deduction*, Tallinn, Estonia, pp. 369-376, 2005.

[4] Biere A., "Resolve and Expand," *in Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, BC, Canada, pp. 59-70, 2005.

[5] Bodlaender H., "Treewidth: Characterizations, Applications and Computations," *in Proceedings of the 32nd International Workshop of Graph-Theoretic Concepts in Computer Science*, Bergen, Norway, pp. 1-14, 2006.

[6] Cadoli M., Schaerf M., Giovanardi A., and Giovanardi M., "An Algorithm to Evaluate Quantified Boolean Formulae and its Experimental Evaluation," *the Journal of Automated Reasoning*, vol. 28, no. 2, pp. 101-142, 2002.

[7] Dershowitz N., Hanna Z., and Katz J., "Bounded Model Checking With QBF," *in Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing*, St Andrews, UK, pp. 408-414, 2005.

[8] Feldmann R., Monien B., and Schamberger S., "A Distributed Algorithm to Evaluate Quantified Boolean Formulae," *in Proceedings of AAAI-00*, Texas, USA, pp. 1-6, 2000.

[9] Gent I., Hoos H., Rowley A., and Smyth K., "Using Stochastic Local Search To Solve Quantified Boolean Formulae," *in Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, Kinsale, Ireland, pp. 348-362, 2003.

[10] Giunchiglia E., Narizzano M., and Tacchella A., "Clause/Term Resolution and Learning in the Evaluation of Quantified Boolean Formulas," *the Journal of Artificial Intelligence Research*, vol. 26, no. 2, pp. 371-416, 2006.

[11] Giunchiglia E., Narizzano M., and Tacchella A., "QuBE++: An Efficient QBF Solver," *in Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design*, Texas, USA, pp. 201-213, 2004.

[12] Janota M., Klieber W., Marques-Silva J., and Clarke E., "Solving QBF with Counterexample Guided Refinement," *in Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing*, Trento, Italy, pp. 114-128, 2012.

[13] Jussila T. and Biere A., "Compressing BMC Encodings with QBF," *Electronic Notes in Theoretical Computer Science*, vol. 174, no. 3, pp. 45-56, 2007.

[14] Kullmann O., "Theory and Applications of Satisfiability Testing," *in Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, Swansea, UK, pp. 430-435, 2009.

[15] Lewis M., Schubert T., Becker B., Marin P., Narizzano M., and Giunchiglia E., "Parallel QBF Solving with Advanced Knowledge Sharing," *Fundamenta Informaticae*, vol. 107, no. 2, pp. 139-166, 2011.

[16] Lewis M., Schubert T., and Becker B., *QMiraXT-A Multithreaded QBF Solver*, Univerlagtuberlin Press, Berlin, 2009.

[17] Marin P., Miller C., and Becker B., "Incremental QBF Preprocessing For Partial Design Verification," *in Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing*, Trento, Italy, pp. 473-474, 2012.

[18] Peschiera C., Pulina L., Tacchella A., Bubeck U., Kullmann O., and Lynce I., "The 7th QBF Solvers Evaluation (QBFEVAL'10)," *in Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing*, Edinburgh, UK, pp. 237-250, 2010.

[19] Pigorsch F. and Scholl C., "Exploiting Structure in an AIG Based QBF Solver," *in Proceedings of Conference on Design, Automation and Test in Europe*, Nice, France, pp. 1596-1601, 2009.

[20] Pulina L. and Tacchella A., "A Self-adaptive Multi-engine Solver for Quantified Boolean

Formulas," *Constraints*, vol. 14, no. 1, pp. 80-116, 2009.
[21] Pulina L. and Tacchella A., "A Structural Approach to Reasoning with Quantified Boolean Formulas," *in Proceedings of the 21st International Joint Conference on Artificial Intelligence*, Italy, pp.596-602, 2009.
[22] Pulina L. and Tacchella A., "An Empirical Study of QBF Encodings: From Treewidth Estimation to Useful Preprocessing," *Fundamenta Informaticae*, vol. 102, no. 3, pp.391-427, 2010.
[23] Pulina L. and Tacchella A., "Hard QBF Encodings Made Easy: Dream or Reality?," *in Proceedings of the 11th International Conference of the Italian Association for Artificial Intelligence*, Reggio Emilia, Italy, pp. 31-34, 2009.
[24] Sadeghi M., Maghooli K., and Moein M., "Using Artificial Immunity Network for Face Verification," *the International Arab Journal of Information Technology*, vol. 11, no. 4, pp.100-106, 2014.
[25] Samulowitz H. and Memisevic R., "Learning to solve QBF," *in Proceedings of the 22nd Conference on Artificial Intelligence*, Canada, pp. 255-260, 2007.
[26] Walsh T., "Search in a Small World," *in Proceedings of the 16th International Joint Conference on Artificial Intelligence*, CA, USA, pp. 1172-1177, 1999.
[27] Watts D. and Strogatz S., "Collective Dynamics of 'Small-World' Networks," *Nature*, vol. 393, no. 6684, pp. 440-442, 1998.
[28] Yan X., Zhao J., and Shi H., "Network-Structure Group Search Optimization Algorithm Based on an Improved Small World Topology and it Application," *Computers and Applied Chemistry*, vol. 28, no. 7, pp. 923-927, 2011.
[29] Yin M., Zhou J., Sun J., and Gu W., "Heuristic Survey Propagation Algorithm for Solving QBF Problem," *Ruanjian Xuebao/Journal of Software*, vol. 22, no. 7, pp. 1538-1550, 2011.

Tao Li is a scientific researcher and project manager in Modern Education and Technology Center, South China Agricultural University and holds an IEEE membership. He completed his PhD degree in School of Computer Science and Engineering, South China University of Technology. His research interests include intelligent computing and data mining.

Nanfeng Xiao is currently a Professor and PhD tutor of computer science in School of Computer Science and Engineering, South China University of Technology. His research interests are in the areas of intelligent computing, intelligent robots and data mining.