UnitMarch

Marijn Heule, Denis de Leeuw Duarte, and Hans van Maaren
marijn@heule.nl, deleeuwduarte@science-and-technology.nl, and h.vanmaaren@tudelft.nl

Introduction

In [2], we observe that propositional Boolean formulas with n variables can be checked on feasibility with a single *multi-bit assignment* of 2^n bits. Compared to conventional checking algorithms, this just exchanges time for space. However, the architecture of today's computers is 32- or 64-bits - which enables execution of 32 (64) 1-bit operations simultaneously. Although many algorithms does not seem suitable for this kind of parallelism, the UnitWalk algorithm [3] appears to be a good first candidate.

Current Satisfiability (Sat) solvers do not use the opportunity of a k-bit processor to simulate parallel 1-bit (Boolean) search on k 1-bit processors. Conventional parallel Sat solving differs from the proposed method: The former realizes performance gain by dividing the workload over multiple processors and some minor changes to the solving algorithm, while the latter uses a single processor and requires significant modifications to the algorithm.

Sat solvers that use multi-bit heuristics frequently (counters for instance), are not very suitable for modification in this respect. However, Sat solvers whose computational "center of gravity" consists of propagating truth values (or other 1-bit operations) may profit from this opportunity. One of such is the state-of-the-art local search Sat solver UnitWalk [1, 3]. We show that UnitWalk can be upgraded using a single k-bit processor. This results in a considerable speed-up.

The UnitWalk algorithm

For a possible application of multi-bit assignments, we focused on local search (incomplete) Sat solvers. In contrast to complete Sat solvers, they are less complicated and work with full assignments. A generic structure of local search Sat solvers is as follows: An assignment φ is generated, earmarking a random Boolean value to all variables. By flipping the truth values of variables, φ can be modified to satisfy the formula at hand. If after a multitude of flips φ still does not satisfy the formula, a new random assignment is generated.

Algorithm 1 Flip_UnitWalk(φ_{master})

```
 1: for $i$ in 1 to MAX_PERIODS do
 2:     if $\varphi_{\text{master}}$ satisfies $F$ then
 3:         break
 4:     end if
 5:     $\pi$ := random ordering of the variables
 6:     $\varphi_{\text{active}} := \emptyset$
 7:     for $j$ in 1 to $n$ do
 8:         while unitclause $u \in \varphi_{\text{active}} \circ F$ do
 9:             $\varphi_{\text{active}}[\ VAR(u)\ ] := TRUTH(u)$
10:         end while
11:         if $\pi(j)$ not assigned in $\varphi_{\text{active}}$ then
12:             $\varphi_{\text{active}}[\ \pi(j)\ ] := \varphi_{\text{master}}[\ \pi(j)\ ]$
13:         end if
14:     end for
15:     if $\varphi_{\text{active}} = \varphi_{\text{master}}$ then
16:         random flip variable in $\varphi_{\text{active}}$
17:     end if
18:     $\varphi_{\text{master}} := \varphi_{\text{active}}$
19: end for
20: return $\varphi_{\text{master}}$
```

The UnitWalk algorithm (see algorithm 1) flips variables in so-called *periods*: Each period starts with an initial assignment (referred to as master assignment φ_{master}), an empty assignment φ_{active} and an ordering of the variables π. First, unit propagation is executed on the empty assignment. Second, the first unassigned variable in π is assigned to its value in φ_{master}, followed by unit propagation of this value. A period ends when all variables are assigned a value in φ_{active}. Notice that *conflicts* - clauses with all literals assigned to false - are more or less neglected, depending on the implementation. A new period starts with the resulting φ_{active} as φ_{master} and a new ordering of the variables.

Multi-Bit Unit Propagation

We will explain multi-bit unit propagation using a detailed period of an example consisting of a 4-bit assignment. Consider the example formula and initial settings below. Unassigned values in φ_{active} are denoted by $*$.

$$\begin{aligned}
\mathcal{F}_{\text{example}} &:= (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \\
&\quad (\neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \\
&\quad \wedge (\neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \\
\varphi_{\text{master}} &:= \{x_1 = \mathsf{0110},\ x_2 = \mathsf{1100}, \\
&\quad x_3 = \mathsf{1010},\ x_4 = \mathsf{0110}\} \\
\varphi_{\text{active}} &:= \{x_1 = ****, x_2 = ****, \\
&\quad x_3 = ****, x_4 = ****\} \\
\pi &:= (x_2, x_1, x_4, x_3)
\end{aligned}$$

Since the formula contains no unit clauses, the algorithm starts by selecting the first variable from the ordering - x_2. We assign this variable to true (as in φ_{master}) and perform unit propagation. This will result in two unit clauses:

$$\begin{aligned}
(x_1 = **** \vee x_2 = \mathsf{1100}) &\quad \Rightarrow \quad x_1 := **\mathsf{11} \\
(\neg x_2 = \mathsf{0011} \vee \neg x_3 = ****) &\quad \Rightarrow \quad x_3 := \mathsf{00}**
\end{aligned}$$

One of them is selected, say x_1 and assigned to its value, resulting in:

$$(\neg x_1 = **\mathsf{00} \vee x_2 = \mathsf{1100} \vee x_3 = \mathsf{00}**) \Rightarrow x_3 := \mathsf{0011}$$

Now we assign x_3 which triggers three clauses:

$$\begin{aligned}
(\neg x_2 = \mathsf{0011} \vee x_3 = \mathsf{0011} \vee \neg x_4 = ****) &\Rightarrow x_4 := \mathsf{00}** \\
(\neg x_2 = \mathsf{0011} \vee x_3 = \mathsf{0011} \vee x_4 = \mathsf{00}**) &\Rightarrow \textbf{conflict} \\
(\neg x_3 = \mathsf{1100} \vee \neg x_4 = \mathsf{11}**) &\Rightarrow x_4 := \mathsf{0000}
\end{aligned}$$

When unit propagation stops, only the first two bits of x_1 are still undefined. These bits are set to their value in φ_{master} assigning all variables. The period ends with $\varphi_{\text{active}} = \{x_1 = \mathsf{0111}, x_2 = \mathsf{1100}, x_3 = \mathsf{0011}, x_4 = \mathsf{0000}\}$ - which satisfies the formula in the third and fourth bit.

The reader may check that: (1) The order in which unit clauses are propagated, as well as the order in which clauses are evaluated is not fixed. The order influences φ_{active} in case of conflicts. For example, evaluating $\neg x_2 \vee x_3 \vee x_4$ before $\neg x_2 \vee x_3 \vee \neg x_4$ results in a different final φ_{active}. (2) In the 4-bit example the third and fourth bit are the same for all variables. This effect could reduce the parallelism, because the algorithm as such does not intervene here and in fact maintains this collapse. This effect is not restricted to formulas with few variables. During our experiments we frequently detected a convergence to identical assignments over a considerable number of bit positions (sometimes even over all 32 positions, when using a 32-bit processor). We implemented a fast detection algorithm which replaces a possible double with a new random assignment. Notice however that by doing so the first "communication" aspect is introduced.

Implementation

We implemented a multi-bit version of the UnitWalk algorithm in our solver UnitMarch. Some aspects are implemented differently compared to UnitWalk. For instance: 1) Instead of using a multi-set data-structure to propagate unit clauses, we use a queue data-structure. This makes the algorithm slightly less random. 2) Our implementation neglects conflicting unit clauses, while UnitWalk prefers the one with the polarity in φ_{master}. This results generally in more flips during a period. For details see [2].

References

[1] D. Le Berre and L. Simon, *The essentials of the SAT'03 Competition.* Springer Verlag, Lecture Notes in Comput. Sci. **2919** (2004), 452–467.

[2] Marijn J.H. Heule and Hans van Maaren. *From Idempotent Generalized Boolean Assignments to Multi-bit Search.* Submitted to SAT 2007.

[3] E. A. Hirsch, A. Kojevnikov. *UnitWalk: A new SAT solver that uses local search guided by unit clause elimination.* Ann. Math. Artif. Intell. **43**(1) (2005), 91–111.