Adrian Balint Michael Henn
Oliver Gableske

July 1, 2009

A novel approach to combine a SLS- and a DPLL-solver for the satisfiability problem

Overview

- Introduction
  - Motivation
  - Preliminary Study
- Search Space Partition
  - Definition
  - Using SSPs to check intensification
  - *hybridGM*
- Results
  - Implementation Details
  - Empirical Results
  - Conclusions

Motivation for developing hybrid SAT-solver

Major solving paradigms for the SAT-problem

- DPLL
  - \+ Complete: can solve sat and unsat problems
  - \+ Good at solving industrial and crafted problems
  - \- Not good on random formulas
  - \- Use large amounts of memory
- SLS
  - \+ Fast at solving random sat problems
  - \+ Little memory consumption
  - \- Incomplete
  - \- Scale bad on industrial and crafted problems

Motivation for developing hybrid SAT-solver

Major solving paradigms for the SAT-problem

- DPLL
  - \+ Complete: can solve sat and unsat problems
  - \+ Good at solving industrial and crafted problems
  - \- Not good on random formulas
  - \- Use large amounts of memory
- SLS
  - \+ Fast at solving random sat problems
  - \+ Little memory consumption
  - \- Incomplete
  - \- Scale bad on industrial and crafted problems

Considerable effort has been undertaken to combine these two solving paradigms for gaining: robustness, completeness or speed

Preliminary study towards hybridisation

Our approach

- Look for the weaknesses of a solver and try to overcome it with a solver following the other paradigm
- We have chosen a SLS-solver for our analysis : *gNovelty+* (winner of SAT2007 competition category random sat)

Preliminary study towards hybridisation

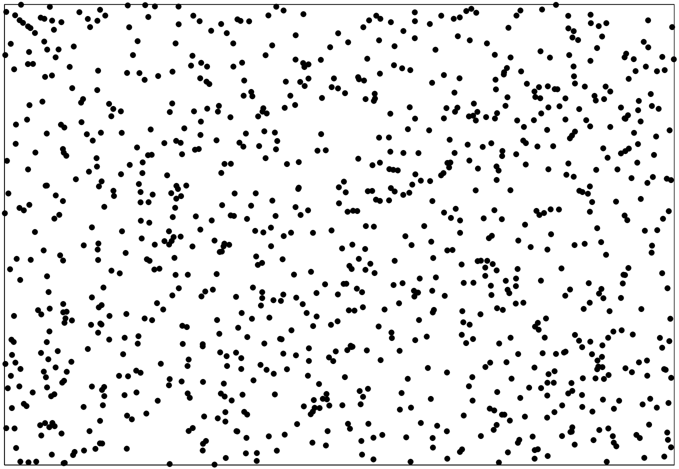
Our approach

- Look for the weaknesses of a solver and try to overcome it with a solver following the other paradigm
- We have chosen a SLS-solver for our analysis : *gNovelty+* (winner of SAT2007 competition category random sat)

Search Properties of a SLS-solver

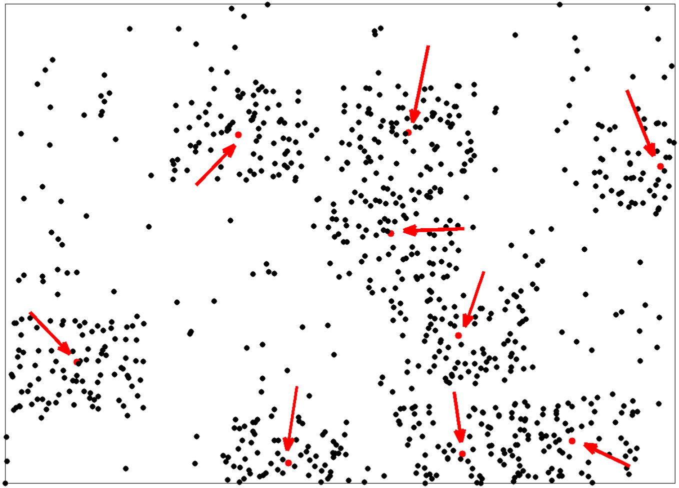
1. Diversification: How good is the search space coverage?
2. Intensification: How good is the search around high quality points?

An Example for Good Diversification



The search space represented two-dimensionally.
Black dots: points visited by the solver during its search

An Example for Good Intensification



Red dots: Good quality local minima

Diversification analysis

Our approach

- Try to cluster the points visited by the SLS $\rightarrow$ impossible due to large data (formula with 4000 variables $\rightarrow$ up to 10^8 flips during the search)
- Confine to search space points where the objective function has low values (i.e. local minima and their neighborhood)
- Save all this points in a bloom filter
- Check how many points are in the close neighborhood of the saved points

Diversification analysis

Our approach

- Try to cluster the points visited by the SLS → impossible due to large data (formula with 4000 variables → up to 10^8 flips during the search)
- Confine to search space points where the objective function has low values (i.e. local minima and their neighborhood)
- Save all this points in a bloom filter
- Check how many points are in the close neighborhood of the saved points

Results

Not more than 2% of the points lay in the close neighborhood of the saved ones → good diversification

Intensification analysis

Why is intensification around good local minima important?

Zhang(2004) showed: the quality of a local minimum is correlated with the hamming distance between the local minimum and the nearest solution

Intensification analysis

Why is intensification around good local minima important?

Zhang(2004) showed: the quality of a local minimum is correlated with the hamming distance between the local minimum and the nearest solution

The optimal SLS-solver

Would intensify the search around those local minima where the neighborhood contains a solution

Intensification analysis

Why is intensification around good local minima important?

Zhang(2004) showed: the quality of a local minimum is correlated with the hamming distance between the local minimum and the nearest solution

The optimal SLS-solver

Would intensify the search around those local minima where the neighborhood contains a solution

How to check if the intensification is sufficient

Search the complete Hamming neighborhood (within a certain distance) of local minima for solutions: possible only for very small formulas

Intensification analysis

Why is intensification around good local minima important?

Zhang(2004) showed: the quality of a local minimum is correlated with the hamming distance between the local minimum and the nearest solution

The optimal SLS-solver

Would intensify the search around those local minima where the neighborhood contains a solution

How to check if the intensification is sufficient

Search the complete Hamming neighborhood (within a certain distance) of local minima for solutions: possible only for very small formulas

Solution

Create a new neighborhood relation that can be searched up "fast" $\longrightarrow$ Search Space Partition (SSP)

Some Definitions

Complete Assignment of a Formula F

$\alpha \in \mathbb{B}^n$ where $\mathbb{B} = \{0, 1\}$ is called complete assignment of the variables $\{x_1, \dots, x_n\}$ of F

Some Definitions

Complete Assignment of a Formula F

$\alpha \in \mathbb{B}^n$ where $\mathbb{B} = \{0, 1\}$ is called complete assignment of the variables $\{x_1, \dots, x_n\}$ of F

Partial Assignment

$\beta \in \mathbb{RB}^n$ where $\mathbb{RB} = \{0, 1, ?\}$ is called partial assignment (not all variables have a value)

The number of ?-symbols in β (size of β) is described by $|\beta|_?$

Some Definitions

Complete Assignment of a Formula F

$\alpha \in \mathbb{B}^n$ where $\mathbb{B} = \{0, 1\}$ is called complete assignment of the variables $\{x_1, \ldots, x_n\}$ of F

Partial Assignment

$\beta \in \mathbb{RB}^n$ where $\mathbb{RB} = \{0, 1, ?\}$ is called partial assignment (not all variables have a value)

The number of ?-symbols in β (size of β) is described by $|\beta|_?$

Flip Trajectory

$(t_1, \ldots, t_w)$ where $t_i \in \{x_1, \ldots, x_n\}$ denote the variables being flipped by the SLS-algorithm during its search (w denotes the total number of flips)

Search Space Partition

Definition

Given a complete assignment α_j of F visited by the solver in the j-th flip and the flip trajectory $(t_1, \ldots, t_w)$ we construct a partial assignment β by starting with $k = 0$ and $\beta = \alpha_j$ and then repeat:

$\beta[t_{j+k}] = ?$ and $\beta[t_{j-k}] = ?$ where $t_{j\pm k}$ are the variables of the flip trajectory until $|\beta|_? \geq c \cdot n$ where c is some fixed constant $c \in (0, 1)$

This so constructed partial assignment is called a Search Space Partition.

Search Space Partition

Definition

Given a complete assignment α_j of F visited by the solver in the j-th flip and the flip trajectory $(t_1, \ldots, t_w)$ we construct a partial assignment β by starting with $k = 0$ and $\beta = \alpha_j$ and then repeat:

$\beta[t_{j+k}] =?$ and $\beta[t_{j-k}] =?$ where $t_{j\pm k}$ are the variables of the flip trajectory until $|\beta|_? \geq c \cdot n$ where c is some fixed constant $c \in (0,1)$

This so constructed partial assignment is called a Search Space Partition.

Example

$\alpha_7 = (0,0,1,1,0,1,0,1,1,1)$ assignment for F with 10 variables ($c = 0.5$)

flip trajectory $(x_2, x_6, x_1, x_9, x_1, x_6, \underline{\mathbf{x_1}}, x_3, x_9, x_1, x_1, x_8, x_3, \ldots)$

for $k = 0$ we have

$\beta = (?,0,1,1,0,1,0,1,1,1)$

Search Space Partition

Definition

Given a complete assignment α_j of F visited by the solver in the j-th flip and the flip trajectory $(t_1, \ldots, t_w)$ we construct a partial assignment β by starting with $k = 0$ and $\beta = \alpha_j$ and then repeat:

$\beta[t_{j+k}] = ?$ and $\beta[t_{j-k}] = ?$ where $t_{j \pm k}$ are the variables of the flip trajectory until $|\beta|_? \geq c \cdot n$ where c is some fixed constant $c \in (0, 1)$

This so constructed partial assignment is called a Search Space Partition.

Example

$\alpha_7 = (0,0,1,1,0,1,0,1,1,1)$ assignment for F with 10 variables $(c = 0.5)$

flip trajectory $(x_2, x_6, x_1, x_9, x_1, \underline{x_6, \mathbf{x_1}, x_3}, x_9, x_1, x_1, x_8, x_3, \ldots)$

for $k = 1$ we have

$\beta = (?, 0, ?, 1, 0, ?, 0, 1, 1, 1)$

Search Space Partition

Definition

Given a complete assignment α_j of F visited by the solver in the j-th flip and the flip trajectory $(t_1, \ldots, t_w)$ we construct a partial assignment β by starting with $k = 0$ and $\beta = \alpha_j$ and then repeat:

$\beta[t_{j+k}] = ?$ and $\beta[t_{j-k}] = ?$ where $t_{j\pm k}$ are the variables of the flip trajectory until $|\beta|_? \geq c \cdot n$ where c is some fixed constant $c \in (0, 1)$

This so constructed partial assignment is called a Search Space Partition.

Example

$\alpha_7 = (0,0,1,1,0,1,0,1,1,1)$ assignment for F with 10 variables ($c = 0.5$)

flip trajectory $(x_2, x_6, x_1, x_9, \underline{x_1, x_6, \mathbf{x_1}, x_3, x_9}, x_1, x_1, x_8, x_3, \ldots)$

for $k = 2$ we have

$\beta = (?, 0, ?, 1, 0, ?, 0, 1, ?, 1)$

Search Space Partition

Definition

Given a complete assignment α_j of F visited by the solver in the j-th flip and the flip trajectory $(t_1, \dots, t_w)$ we construct a partial assignment β by starting with $k = 0$ and $\beta = \alpha_j$ and then repeat:

$\beta[t_{j+k}] =?$ and $\beta[t_{j-k}] =?$ where $t_{j\pm k}$ are the variables of the flip trajectory until $|\beta|_? \geq c \cdot n$ where c is some fixed constant $c \in (0,1)$

This so constructed partial assignment is called a Search Space Partition.

Example

$\alpha_7 = (0,0,1,1,0,1,0,1,1,1)$ assignment for F with 10 variables ($c = 0.5$)

flip trajectory $(x_2, x_6, x_1, \underline{x_9, x_1, x_6, \mathbf{x_1}, x_3, x_9, x_1}, x_1, x_8, x_3, \dots)$

for $k = 3$ we have

$\beta = (?,0,?,1,0,?,0,1,?,1)$

Search Space Partition

Definition

Given a complete assignment α_j of F visited by the solver in the j-th flip and the flip trajectory $(t_1, \ldots, t_w)$ we construct a partial assignment β by starting with $k = 0$ and $\beta = \alpha_j$ and then repeat:

$\beta[t_{j+k}] = ?$ and $\beta[t_{j-k}] = ?$ where $t_{j\pm k}$ are the variables of the flip trajectory until $|\beta|_? \geq c \cdot n$ where c is some fixed constant $c \in (0,1)$

This so constructed partial assignment is called a Search Space Partition.

Example

$\alpha_7 = (0,0,1,1,0,1,0,1,1,1)$ assignment for F with 10 variables ($c = 0.5$)

flip trajectory $(x_2, x_6, \underline{x_1, x_9, x_1, x_6, \mathbf{x_1}, x_3, x_9, x_1, x_1}, x_8, x_3, \ldots)$

for $k = 4$ we have

$\beta = (?,0,?,1,0,?,0,1,?,1)$

Search Space Partition

Definition

Given a complete assignment α_j of F visited by the solver in the j-th flip and the flip trajectory $(t_1, \ldots, t_w)$ we construct a partial assignment β by starting with $k = 0$ and $\beta = \alpha_j$ and then repeat:

$\beta[t_{j+k}] = ?$ and $\beta[t_{j-k}] = ?$ where $t_{j\pm k}$ are the variables of the flip trajectory until $|\beta|_? \geq c \cdot n$ where c is some fixed constant $c \in (0,1)$

This so constructed partial assignment is called a Search Space Partition.

Example

$\alpha_7 = (0,0,1,1,0,1,0,1,1,1)$ assignment for F with 10 variables $(c = 0.5)$

flip trajectory $(x_2, \underline{x_6, x_1, x_9, x_1, x_6, \mathbf{x_1}, x_3, x_9, x_1, x_1, x_8}, x_3, \ldots)$

for $k = 5$ we have

$\beta = (?,0,?,1,0,?,0,?,?,1)$

Using SSPs to check intensification

Algorithm for checking intensification

1. Run SLS-solver and save a portion of the actual flip trajectory
2. Create SSPs around good quality local minima on the fly
3. Apply the partial assignment of the SSP on the formula to get a simplified sub formula
4. Solve the sub formula with a complete DPLL-solver to gain certainty that there is no solution

Using SSPs to check intensification

Algorithm for checking intensification

1. Run SLS-solver and save a portion of the actual flip trajectory
2. Create SSPs around good quality local minima on the fly
3. Apply the partial assignment of the SSP on the formula to get a simplified sub formula
4. Solve the sub formula with a complete DPLL-solver to gain certainty that there is no solution

Details

1. The construction of SSPs is done on the fly in linear time
2. The size can be controlled by the constant c
3. If the DPLL-solver finds a solution within a SSP $\rightarrow$ a new faster hybrid solver


#Unsat
Clauses
1
Flip

Implementation details

hybridGM

- SLS-solver: *gNovelty+* (SAT 2007 Comp. version slightly modified)
- DPLL-solver : *march_ks* (bug fixed version with some slight modification by Marijn Heule)
- Build SSPs around local minima with only 1 unsatisfied clause
- The constant c starts with $1/2$ and increases in steps of $1/20$ if the sub formula is too simple for *march_ks*

Implementation details

hybridGM

- SLS-solver: *gNovelty+* (SAT 2007 Comp. version slightly modified)
- DPLL-solver : *march_ks* (bug fixed version with some slight modification by Marijn Heule)
- Build SSPs around local minima with only 1 unsatisfied clause
- The constant c starts with $1/2$ and increases in steps of $1/20$ if the sub formula is too simple for *march_ks*

Empirical Tests

- Different instances from SAT 2007 Competition
- Each instance is solved 100 times (cutoff 2000 sec.)
- In case of 5-SAT and 7-SAT there was sometimes a memory leak

Some Empirical Results

Instance	gNovelty+	adaptG2-WSAT0	hybridGM3 (gNov,March)	Gain
SAT 2007 Competition random instances				
unif2p-p0.7-v3500-c9345-S1568322528-08	10%	2.91 | 1.63	9.63 | 7.82 (9,91)	**>1**
unif2p-p0.7-v6500-c17355-S1097641288-15	97.64 | 69.35	3.75 | 1.61	4.31 | 2.57 (20,80)	**22.65**
unif2p-p0.7-v6500-c17355-S152598520-02	226.64 | 168.00	10.23 | 1.90	2.17 | 1.66 (27,73)	**104.44**
unif2p-p0.8-v1295-c4027-S1762612346-15	136.06 | 94.03	1.13 | 0.79	2.65 | 2.20 (9,91)	**51.34**
unif2p-p0.8-v1665-c5178-S1363528912-04	20.84 | 16.16	5.11 | 3.33	5.21 | 4.65 (73,27)	**4.00**
unif2p-p0.8-v2405-c7479-S1163137157-19	8.18 | 6.14	4.00 | 2.27	9.67 | 8.01 (28,72)	0.85
unif-k3-r4.261-v650-c2769-S1159448555-06	0.46 | 0.29	0.24 | 0.19	0.67 | 0.53 (76,24)	0.69
unif-k3-r4.2-v10000-c42000-S1173369833-06	73.87 | 50.01	11%	7.28 | 5.61 (23,77)	**10.15**
unif-k3-r4.2-v13000-c54600-S1416986890-04	331.30 | 250.70	0%	22.02 | 17.58 (23,77)	**15.05**
unif-k3-r4.2-v16000-c67200-S1600965758-04	18%	0%	73% (18,55)	**>1**
unif-k3-r4.2-v16000-c67200-S1826381479-08	550.04 | 457.84	0%	38.00 | 33.06 (23,77)	**14.47**
unif-k3-r4.2-v19000-c79800-S1106616038-10	74%	0%	92.02 | 70.75 (39,61)	**>1**
unif-k3-r4.2-v19000-c79800-S1875179522-13	470.69 | 381.00	0%	25.59 | 20.93 (25,75)	**18.39**
unif-k3-r4.2-v4000-c16800-S1178874381-13	8.11 | 6.02	184.52 | 110.63	4.99 | 4.00 (46,54)	**1.63**
unif-k3-r4.2-v4000-c16800-S1580061366-10	2.51 | 1.98	19.21 | 13.11	1.18 | 1.01 (42,58)	**2.13**
unif-k3-r4.2-v7000-c29400-S102550125-14	42.92 | 31.46	82%	6.85 | 5.70 (28,72)	**6.27**
unif-k3-r4.2-v7000-c29400-S1312035429-13	288.97 | 184.15	1%	246.31 | 160.57 (40,60)	**1.17**

Conclusion

Summary

- We have introduced the term of a Search Space Partition for SLS-solvers based on their search trajectory
- We propose a simple generic approach to combine a SLS- and a DPLL-solvers for a speedup with the help of SSPs
- Empirical results show that this approach is promising (SAT Competition 2009 results ...?)

Conclusion

Summary

- We have introduced the term of a Search Space Partition for SLS-solvers based on their search trajectory
- We propose a simple generic approach to combine a SLS- and a DPLL-solvers for a speedup with the help of SSPs
- Empirical results show that this approach is promising (SAT Competition 2009 results ...?)

Outlook

- Dynamical adaptation scheme for when the SSPs are built
- Improve *hybridGM*'s performance at solving 5-SAT and 7-SAT

Adrian Balint Michael Henn
Oliver Gableske
July 1, 2009

A novel approach to combine a SLS- and a DPLL-solver for the satisfiability problem

Pseudocode of *hybridGM*

```
INPUT: formula F, cutoff.
OUTPUT: model for F or UNKNOWN.
hybridGM(F, cutoff) {
    α = α_S = random assignment;
    numFlips = 0;
    c = 0.5;
    barrier = 1;
    collectSSP = FALSE;
    while (numFlips < cutoff) {
        var = pickVar();
        α[var] = 1 − α[var];
        numFlips++;
        if (α is model for F) return α;
        if (numUnsatClauses == barrier) {
            β = α;
            collectSSP = TRUE;
            j = numFlips;
            k = 0;
        }
        if (collectSSP == TRUE) {
            β[ variableIndex( T_S(F, α_S)[j + k] ) ] = ?;
            β[ variableIndex( T_S(F, α_S)[j − k] ) ] = ?;
            k++;
        }
        if (|β|_? ≥ cn) {
            μ = March_ks(F, β);
            if (μ is model for F) return μ;
            else if (unaryConflictOccurred() == TRUE)
                c = c + 0.05 · n;
            collectSSP = FALSE;
        }
        updateParameters(); //noise, scores
    }
    return UNKNOWN;
}
```