A Study of Divide and Distribute Fixed Weights and its Variants

Cayden R. Codel and Marijn J. H. Heule

Based on the PoS submission found at `http://crcodel.com/research/ddfw_pos.pdf` and the larger research thesis found at `http://crcodel.com/research/ddfw_thesis.pdf`.

Introduction

We studied the Divide and Distribute Fixed Weights (DDFW) stochastic local search algorithm

Introduction

We studied the Divide and Distribute Fixed Weights (DDFW) stochastic local search algorithm

DDFW finds satisfying assignments by minimizing unsatisfied clause weight

Introduction

We studied the Divide and Distribute Fixed Weights (DDFW) stochastic local search algorithm

DDFW finds satisfying assignments by minimizing unsatisfied clause weight

In local minima, DDFW distributes weight from satisfied to unsatisfied clauses

Introduction

We studied the Divide and Distribute Fixed Weights (DDFW) stochastic local search algorithm

DDFW finds satisfying assignments by minimizing unsatisfied clause weight

In local minima, DDFW distributes weight from satisfied to unsatisfied clauses

We studied how to best flip variables and distribute weight by testing DDFW against modern hard benchmarks

The DDFW algorithm

Variable flip variants

Weight redistribution variants

The DDFW algorithm

All clauses receive an initial weight ($w_{\text{init}}[= 8]$)

The DDFW algorithm

All clauses receive an initial weight ($w_{\text{init}}[= 8]$)

DDFW flips variables which most reduce the unsatsfied clause weight (W_U)

The DDFW algorithm

All clauses receive an initial weight ($w_{\text{init}}[= 8]$)

DDFW flips variables which most reduce the unsatsfied clause weight (W_U)

In local minima, DDFW moves weight from one satisfied neighbor to each unsatisfied clause

The DDFW algorithm

All clauses receive an initial weight ($w_{\text{init}}[= 8]$)

DDFW flips variables which most reduce the unsatsfied clause weight (W_U)

In local minima, DDFW moves weight from one satisfied neighbor to each unsatisfied clause

DDFW is the only SLS algorithm in the UBCSAT framework to efficiently solve the $n = 7824$ Pythagorean triples instance

The DDFW algorithm as pseudocode

Algorithm 1: DDFW

```
Input: CNF formula 𝓕
Set all clause weights to w_init = 8
α ← randomly generated truth assignment
for MAX-FLIPS times do
    if assignment α satisfies 𝓕 then  return α
    if flipping a variable reduces W_U then
        Flip a literal that reduces W_U the most
    else
        for each unsatisfied clause C_j do
            C_k ← maximum-weight neighbor of C_j
            if weight of C_k > w_init then
                Transfer a weight of 2 from C_k to C_j
            else
                Transfer a weight of 1 from C_k to C_j
return "No satisfying assignment"
```

The DDFW algorithm as pseudocode

Algorithm 2: DDFW

Input: CNF formula $\mathcal{F}$

Set all clause weights to $w_{\text{init}} = 8$

$\alpha \leftarrow$ randomly generated truth assignment

for MAX-FLIPS *times* **do**

    if *assignment* α *satisfies* $\mathcal{F}$ **then** return α

    if *flipping a variable reduces* W_U **then**

        Flip a literal that reduces W_U the most $\leftarrow$

    else

        for *each unsatisfied clause* C_j **do**

            $C_k \leftarrow$ maximum-weight neighbor of C_j

            if *weight of* $C_k > w_{\text{init}}$ **then**

                Transfer a weight of 2 from C_k to C_j $\leftarrow$

            else

                Transfer a weight of 1 from C_k to C_j $\leftarrow$

return "No satisfying assignment"

Benchmark set

Ten encodings of matrix multiplication challenges

Ten random 3-SAT instances from the 2018 SAT Competition

Two encodings of the $n = 7824$ Pythagorean triples problem

Two encodings of asias and three of Steiner triple problems

All CNFs can be found at
`https://github.com/marijnheule/benchmarks` and
`http://satcompetition.org`

The DDFW algorithm

Variable flip variants

Weight redistribution variants

Variable flip variants

DDFW greedily flips variables which reduce W_U the most

Variable flip variants

DDFW greedily flips variables which reduce W_U the most

Idea: flip W_U-reducing variables probabilistically

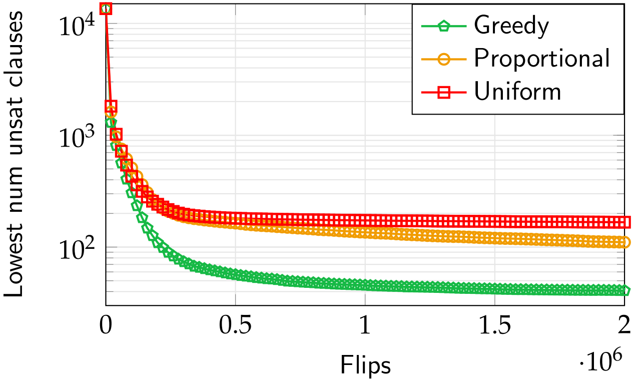
Variable flip variants

DDFW greedily flips variables which reduce W_U the most

Idea: flip W_U-reducing variables probabilistically

We investigated a uniform and a weighted probability distribution

Variable flip variant experimental results



Averaged over all problem instances. The greedy (original) method performed significantly better than the variants.

The DDFW algorithm

Variable flip variants

Weight redistribution variants

Weight redistribution variants

DDFW distributes 1 or 2 units of weight between clauses

Weight redistribution variants

DDFW distributes 1 or 2 units of weight between clauses

We can generalize to a linear rule:

Algorithm 4: Linear weight transfer rule

```
if weight of unsat neighbor C_k > w_init then
    Transfer a weight of a_> × w(C_k) + c_> from C_k to C_j
else
    Transfer a weight of a_≤ × w(C_k) + c_≤ from C_k to C_j
```

if *weight of unsat neighbor* $C_k > w_{\text{init}}$ **then**
  Transfer a weight of $a_{>} \times w(C_k) + c_{>}$ from C_k to C_j
else
  Transfer a weight of $a_{\leq} \times w(C_k) + c_{\leq}$ from C_k to C_j

Weight redistribution variants experimental results

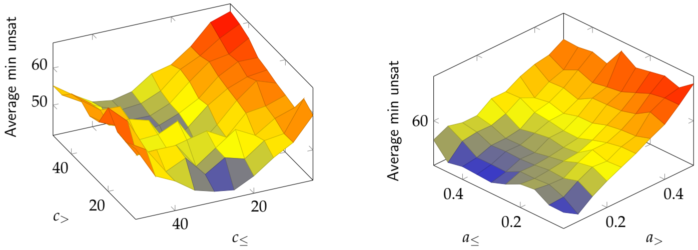
$w_{\text{init}} = 100$. Results averaged over all instances and 100 runs per instance, five million flip timeout

Distribution policy	Avg lowest unsat	Solve %
Original DDFW	36.57	0.85
$(c_>, c_\leq) = (10, 25)$	29.16	1.7
$(a_>, a_\leq) = (0.05, 0.05)$	23.82	2.96
$(a_>, c_>) = (a_\leq, c_\leq) = (0.1, 5)$	22.05	2.67

Linear rule performed about 40% better

Weight redistribution variants experimental results

Parameter searches across the matrix multiplication challenges



Note greater effect of $a_>$ on shape of right plot, while $c_\leq$ determines shape of left plot

Weight redistribution variants continued

Can also distribute weight from entire neighborhoods

Weight redistribution variants continued

Can also distribute weight from entire neighborhoods

Two methods tested: apply linear rule to each clause or to clause proportional to clause weight

Weight redistribution variants continued

Can also distribute weight from entire neighborhoods

Two methods tested: apply linear rule to each clause or to clause proportional to clause weight

Experimental results disappointing but show some promise

Distribution policy	Avg lowest unsat	Matrix lowest unsat
Original DDFW	36.57	57.02
Proportional	46.91	27.0
Direct	45.29	33.91

Conclusions and future work

A simple generalization of the weight distribution method for DDFW yields up to 40% improvement

Conclusions and future work

A simple generalization of the weight distribution method for DDFW yields up to 40% improvement

More complex weight transfer rules may be more effective than a linear one

Conclusions and future work

A simple generalization of the weight distribution method for DDFW yields up to 40% improvement

More complex weight transfer rules may be more effective than a linear one

Spreading weight across more clauses in a neighborhood could cause DDFW to escape local minima faster