Solving weighted CSPs by Successive Relaxations

Erin Delisle Fahiem Bacchus

University of Toronto
Department of Computer Science

September, 2013

Introduction

Weighted CSPs

WCSPs: An optimization variant of regular CSPs. Instead of constraints we have soft constraints.

- Each soft constraint contains tuples of assignments to the variables in its scope.
- Tuples may be hard, having infinite cost and prohibiting assignments
- Tuples may be soft, having finite positive cost

The cost of a complete assignment is the sum of the costs it incurs from each soft constraint. An assignment is a solution if it has finite cost. A solution is optimal if its cost is minimum over all solutions.

Introduction

WCSP Solvers

The current best performing WCSP solver is Toulbar 2

- Uses branch and bound search
- Has sophisticated methods for finding lower bounds on the cost incurred by partial assignments

WCSPs are a CSP version of weighted MAXSAT. Alternate algorithms to branch and bound have been utilized in MAXSAT solvers.

Introduction

Solvers in the Maxsat world

There are two main types of MAXSAT solvers:

- Branch and bound solvers
- Sequence of decision problem solvers

Sequence of decision problem solvers often display better empirical performance on larger problems.

A recent hybrid solver utilizing the sequence of decisions approach is MAXHS.

Introduction

The MaxHS Maxsat solver

Brief overview of MAXHS:

1. Try and find a solution incurring no extra cost
2. On failure, find a set of culprit soft clauses (a core)
3. Find a cost minimal set of clauses that hit all acquired cores
4. Create a new decision problem by removing this cost minimal set of clauses. (This allows the cost of these clauses to be incurred).

Advantages:

- separates logical and arithmetic reasoning by using both a SAT and a MIP solver.
- allows the use of clause learning (nogood learning)

Introduction

MaxHS and WCSPs

Problems with applying the MAXHS method directly to a WCSP:

- Clauses in the hitting set must be falsified
- All tuples in a constraint are mutually exclusive!
- Exclusive assignments can be worked around, but not ideal

Idea: incur cost by constraint instead of by clause (tuple)

Background

Relaxations

Definition

A **weight vector** $< w_1, ..., w_m >$ is a vector of costs with a cost w_i associated with the ith constraint.

Definition

The **relaxation** induced by a weight vector $\vec{w}$ is an unweighted CSP formed from the WCSP. For each constraint c_i in the WCSP we remove all tuples with cost greater than w_i. The remaining tuples are taken to be the satisfying tuples of c_i in the relaxation.

Definition

The **cost** of a weight vector is the sum of its entries. For two weight vectors $\vec{w} =< w_1, ..., w_m >$ and $\vec{y} =< y_1, ..., y_m >$ we say that $\vec{w}$ is **strictly smaller** than $\vec{y}$ if for all i, $w_i < y_i$ or $w_i = y_i = \infty$.

Background

Example

Given a WCSP with $V = \{A, B, C\}$, $D(A) = D(B) = D(C) = \{0, 1\}$

Constraints	$C_1(A, B)$	$C_2(A, C)$	$C_3(B, C)$
$(0, 0)$	0	2	3
$(0, 1)$	1	5	4
$(1, 0)$	2	4	1
$(1, 1)$	3	2	4

Relaxation $\vec{w_1} = (1, 2, 3)$:
$C_1 : (A, B) \in \{(0, 0), (0, 1)\}$
$C_2 : (A, C) \in \{(0, 0), (1, 1)\}$
$C_2 : (B, C) \in \{(0, 0), (1, 0)\}$

Relaxation $\vec{w_2} = (2, 5, 1)$:
$C_1 : (A, B) \in \{(0, 0), (0, 1), (1, 0)\}$
$C_2 : (A, C) \in \{(0, 0), (0, 1), (1, 0), (1, 1)\}$
$C_2 : (B, C) \in \{(1, 0)\}$

Background

Cores

We define a new type of core:

Definition

A **core** is a weight vector $\vec{c}$ such that for all strictly smaller weight vectors $\vec{w}$, the relaxation $\vec{w}$ has no solution.

Cores specify disallowed cost distributions over constraints, allowing us to reason about how much cost must be incurred by the WCSP.

Given a core $\vec{c}$, any relaxation induced by weight vector $\vec{w}$ can only be satisfiable if $w_i \geq c_i$ for some i. A satisfiable relaxation must relax at least one constraint of any core.

Background

Example

Given a WCSP with $V = \{A, B, C\}$, $D(A) = D(B) = D(C) = \{0, 1\}$

Constraints	$C_1(A, B)$	$C_2(A, C)$	$C_3(B, C)$
$(0, 0)$	0	2	3
$(0, 1)$	1	5	4
$(1, 0)$	2	4	1
$(1, 1)$	3	2	4

$\vec{w_1} = (\infty, 2, \infty)$ and $\vec{w_2} = (\infty, \infty, 1)$ are cores.

$\vec{w_3} = (1, \infty, 3)$ is also a core.

Background

Cores

Definition

A weight vector $\vec{w}$ **satisfies a core** $\vec{c}$ if $w_i \geq c_i$ for some i. A weight vector $\vec{w}$ satisfies a set of cores K if it satisfies each core $k \in K$.

Definition

A weight vector satisfying a set of cores is optimal if it has minimum cost over all such weight vectors.

Theorem

Let $\mathcal{K}$ be any set of cores of WCSP *wt$\mathcal{P}$, and $\vec{w}$ be an optimal weight vector satisfying $\mathcal{K}$. If π is a solution to the relaxation $\mathcal{P}_{\vec{w}}$ then π is an optimal solution for wt$\mathcal{P}$.*

Background

Algorithm

Algorithm 1: Algorithm for solving a WCSP $wt\mathcal{P}$

```
1  wtCSP-Solver(wtP)
2  begin
3      Cores ⟵ ∅
4      w⃗ = 0⃗
5      while true do
6          (solvable?, newcore) ⟵ relaxAndSolve(wtP, w⃗)
7          if solvable? then
8              return extractLastSolution()
9      cores ⟵ cores ∪ {newcore}
10     w⃗ ⟵ getOptimalWtVec(cores)
```

Implementation

Extracting cores

We use the clause learning CSP solver MINICSP

- For each distinct positive finite cost value within each constraint we create a blocking variable b_i.
- For every clause in MINICSP arising from some soft tuple, we add the associated blocking variable to the clause.

The blocking variables are used as assumptions in MINICSP. If a CSP is unsatisfiable, a subset of the assumptions causing unsatisfiability is returned. This set of blocking variables can be used to form a core.

Implementation

Example

Given a WCSP with $V = \{A, B, C\}$, $D(A) = D(B) = D(C) = \{0, 1\}$

Constraints	$C_1(A, B)$	$C_2(A, C)$	$C_3(B, C)$
$(0, 0)$	0	1 , b_4	2 , b_6
$(0, 1)$	1, b_1	1 , b_4	2 , b_6
$(1, 0)$	2, b_2	1 , b_4	2 , b_6
$(1, 1)$	3, b_3	4 , b_5	∞

This WCSP would require 6 blocking variables

Implementation

Finding optimal relaxations

We formulate the optimal relaxation as mixed integer programming problem and use CPLEX to solve.

- Blocking variables from MINICSP become 0/1 integer programming variables.
- Larger cost blocking variables within a constraint imply smaller cost blocking variables. In the objective function blocking variables have costs adjusted for these implications.
- A constraint is added for every core to ensure that it is satisfied.

Implementation

Optimizations

- Use the CSP solver to reduce core sizes by testing if blocking variables can be removed.
- Preprocess problems by using Toulbar2 to provide virtual arc consistency on the base problem.
- While possible, use a heuristic in place of the mixed integer programming solver to generate initial cores faster. See "Postponing Optimization to Speed up MAXSAT Solving" (Davies & Bacchus, this conference).

Results

Tests were run on the Celar, Spot5, and Linkage benchmarks.

- Celar: radio frequency link assignment problems, primarily soft constraints.
- Spot5: satellite management, mixture of hard and soft constraints.
- Linkage: probabilistic inference for genetic linkage.

Our solver (CSP-Seq) performed poorly on the Celar benchmark, performance being dominated by Toulbar2.

Results

Spot5:

- 16 problem instances in which both solvers time out.
- 3 problems instances in which both solvers solve in under 1s.
- 3 problem instances in which CSP-Seq outperforms Toulbar2.

Linkage:

- 2 problem instances in which both solvers time out.
- 6 problems instances in which both solvers solve in under 1s.
- 5 problem instances in which Toulbar2 outperforms CSP-Seq.
- 9 problem instances in which CSP-Seq outperforms Toulbar2.

Results

Table : Time(s) for selected instances from the Spot5 and Linkage benchmarks

Problem	CSP-Seq	Toulbar2
Spot5 404	6.14	209.17
Spot5 503	0.48	-
Spot5 505	562.75	-
Linkage pedigree 25	11.64	-
Linkage pedigree 39	36.14	3.32
Linkage pedigree 31	-	779.63
Linkage pedigree 7	1.52	6.98
Linkage pedigree 41	270.22	969.88
Linkage pedigree 51	19.83	-

Results

Table : Time(s) for selected instances from the Spot5 and Linkage benchmarks

Problem	CSP-Seq	Toulbar2
Linkage pedigree 18	14.61	119.15
Linkage pedigree 14	8.92	0.52
Linkage pedigree 30	27.09	240.71
Linkage pedigree 9	63.47	223.85
Linkage pedigree 20	15.04	0.76
Linkage pedigree 44	145.11	-
Linkage pedigree 13	4.06	0.56
Linkage pedigree 33	0.76	10.58

Conclusions

- Provides a new method for solving WCSPs.
- Redefines cores and relaxations to exploit the structure of WCSPs. Reduces cores to a single relaxation weight per constraint.
- For future work, we intend to improve performance of the method by adding addition information to the mixed integer programming model and finding alternate core structures.

Thank you for listening!