Duality and Decomposition

Sjoerd van Hagen

20-10-2006
Duality in Constraint Satisfaction

- Introduction
- What is duality?
- 'straightforward' CSP dual and decomposition
  - What is the 'straightforward' CSP dual?
  - Solving CSP’s if we are lucky
  - Decomposition
  - Solving CSP’s if we are not lucky
- Other kinds of duals
  - inference dual
  - relaxation dual
Introduction

- two opinions
  - the book is not finished yet
  - the book could have been a better basis for further study
- so before I start I will try to improve a little on the book
- the rest of this talk will be about subjects that are not in the book and I really think they should be
Graphs versus Hypergraphs

- the book gives a graph representation of CSP’s
- if two variables occur in the same constraint there is an edge
- so constraints with more than two variables are translated to a clique
- why not build a hypergraph of the CSP? (short example)
  - now we can store the constraints with the individual edges
  - we do not introduce cycles
  - cycles make the problem computationally harder
- look again at theorem 5.49, replace arc by hyperarc, it becomes more useful immediately!
- also note that in a tree we can enforce hyperarc consistency really fast
What is duality?

- Generally speaking, dualities translate concepts, theorems or mathematical structures into other concepts, theorems or structures, in a one-to-one fashion.
- Usually if applied twice, we get the original problem.
  - Example: points and lines in a plane.
- Exceptions: graph theory.
  - Where a graph can have multiple different duals.
  - Example: graph with more than one dual.
- So be careful, the meaning of the word duality is not always the same, as we will see again later on.
Straightforward CSP dual

- constraints are translated to variables
  - tuples of constraint will be values of the variable
- variables are translated to constraints
  - there is a constraint between two dual variables iff a variable occurs in both primal constraints
  - the constraints of the dual problem enforce the same values for all variables shared by two constraints
- if applied twice you will get the same (primal) CSP again
- an example to clarify what happens
example

Variables: \( \{ x, y, z \} \)

Domains:

\[
\begin{array}{ccc}
 x & y & z \\
 1 & 1 & 1 \\
 2 & 2 & 2 \\
 3 & 3 & 3 \\
 4 & 4 & 4 \\
\end{array}
\]

Constraints:

\[
\begin{array}{cc}
 y & z \\
 1 & 4 \\
 2 & 1 \\
\end{array}
\]

\[
\begin{array}{cc}
 C_1 & C_2 \\
 1 & 2 \\
 3 & 1 \\
\end{array}
\]

\[
\begin{array}{cc}
 C_1 & C_2 \\
 1 & 2 \\
 3 & 1 \\
 4 & 4 \\
\end{array}
\]
example

Variables: \{x, y, z\}

Domains:

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Constraints:

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

\{C_1, C_2\}

\begin{align*}
C_1 & \rightarrow C_{12} \\
C_2 & \rightarrow C_{12}
\end{align*}

\begin{array}{c|c|c}
\hline
& C_1 & C_2 \\
\hline
1,2 & 2,1 \\
3,1 & 1,4 \\
\hline
\end{array}
just like the primal problem we can visualize the dual problem as a graph
Join graphs and join trees

A graph obtained from the dual graph by removing some redundant edges is called a join graph. Join graphs are equivalent to the original dual problem.
A strong assumption

Assume that we are extremely lucky...
Join graphs and join trees

- obviously, if the primal graph is already a tree or can be made a tree we will solve it immediately
- a join graph can be a tree, if it is we call it a join tree
- if the join graph is a tree we can easily solve the problem
- questions popping up
  - how to find a join tree efficiently?
  - when will we be able to get a join tree?
How do we find a join tree?

- assign weight $n$ to an edge if the vertices connected by that edge share $n$ variables
- property: if a dual graph has a join tree, then all maximal-weight spanning trees of the graph are join trees
- finding join trees
  - step 1: assign weights
  - step 2: find maximal-weight spanning tree
  - step 3: check if it enforces equality of all variables
When will we be able to get a join tree?

- we can also check whether the dual graph has a join tree by looking at the primal graph
  - if the primal graph is chordal
  - if the primal graph is conformant with the problem, which means that all variables of every maximal clique of the primal are the scope of a constraint
- luckily, finding a maximum clique is polynomial in chordal graphs
- show by example
Join graphs and join trees

- seems like a nice way to solve CSP’s
- the requirements however are very restrictive
  - primal graph is a tree or can be made a tree
  - dual graph is a tree or can be made a tree
- still it looks promising...
A strong assumption

Assume that we are not that lucky...

- maybe we can extract ’tree-like structure’ from our graph...
- so let’s talk about decomposition methods
decomposition methods

- for binary problems (graphs)
  - Biconnected components*
  - Cycle cutset*
  - Tree decomposition*

- for arbitrary problems (hypergraphs)
  - Biconnected components
  - Cycle cutset
  - Tree decomposition
  - Hinge decomposition
  - Tree clustering*
  - Hinge/clustering decomposition
  - Query decomposition
  - Hypertree decomposition
  - Generalized hypertree decomposition
biconnected components

- easiest method, but can be very bad
- a separating vertex of a graph is a vertex s.t. if you remove that vertex the number of connected components increases
- a biconnected component is a maximal set of nodes whose induced subgraph is connected but does not have a separating vertex
- graph theory: biconnected components and the separating vertices of a graph form a tree
- nodes are biconnected components and separating vertices
- edges only connect a biconnected component with a separating vertex, which is contained in the biconnected component
example
cycle cutset

- again an easy method, but not very good
- a cycle cutset is a set of vertices if removed from the graph a tree remains i.e. it cuts all cycles (those who have taken the course on probabilistic reasoning have seen this already)
- so now we have a tree :-)
- but no equivalence :-(
- solved by adding the variables of the cutset to all nodes and edges
- obviously we would like the cutset to be as small as possible, unfortunately finding the minimum cycle cutset is NP-hard
tree decomposition in general

- a tree decomposition is a mapping of a graph to a tree
- often used to speed up computations
- given a graph $G = (V, E)$, a tree decomposition is a tuple $(X, T)$, where $X = \{X_1, \ldots, X_n\}$ is a set of subsets of $V$, and $T$ is a tree whose nodes are the subsets $X_i$, such that:
  1. the union of all $X_i \in X$ equals $V$
  2. for every edge $(v, w) \in E$, there is a node $X_i$ that contains both $v$ and $w$
  3. if $X_i$ and $X_j$ both contain a vertex $v$, then all nodes $X_z$ of the tree in the (unique) path between $X_i$ and $X_j$ contain $v$

- width of a tree decomposition $\max(|X_i|) - 1$, treewidth is the minimum width of all tree decompositions
- it is NP-hard to find a tree decomposition whose width is the treewidth of the graph, the treewidth is equal to the size of the maximum clique minus one
tree decomposition applied to CSP

- tree decomposition can be used directly to help solve CSP’s
- reformulated
  - nodes are associated with sets of variables, such that the union of these sets equals the variables of the CSP
  - the scope of each constraint is contained in a set of variables of some node
  - if two sets contain a variable, all nodes of the tree on the path contain that variable as well
- best method
  - because it generalizes both other methods
  - if you can find a good decomposition
- treewidth bound from below by max number of variables in one constraint minus one
- related to bucket elimination which I will not cover but is a well known technique in CS, info on http://en.wikipedia.org/wiki/Bucket_elimination#Bucket_elimination
Tree clustering

- recall that a CSP has a join tree if
  1. the primal graph is chordal
  2. the primal graph is conformant with the problem
- can we make the primal graph chordal?
  1. yes, by introducing 'fake' constraints
  2. small drawback is that we possibly introduce new maximal cliques
- can we make the primal graph conformant?
  1. yes, by merging constraints
  2. for every maximal clique we introduce a constraint with scope all variables in the clique
example
this seems like a nice approach...
  ▶ it is easy to understand
  ▶ it is quite fast...
  ▶ if you pick the right order for adding edges to make the graph chordal
  ▶ more precisely, it is exponential in the width of the graph with respect to the order
  ▶ computing the order which minimizes this width is NP-hard

a reasonable ordering would probably be the maximal cardinality ordering because it does not add edges if the graph is already chordal, also it is easy to find maximum cliques using this order
Solving CSP’s

- assume we have a decomposition of size polynomial in the input of the original CSP
- assume that we can solve subproblems in polynomial time i.e. the width of the decomposition is bounded by a constant
- now we can, in polynomial time, create a structure that can give us any solution we like, by enforcing arc consistency
The constraints shown should be read as the summary of the subproblem at that node.
example

Summary of subproblem is propagated upwards.
example

And again.
other duals

- besides the straightforward dual there are other translations of CSP also called duals
- these duals are also used with optimization
- can be divided into two groups
  - inference duals
  - relaxation duals
shorthand notations

- optimization problem

\[
\min_{x \in D} \{ f(x) | C \}
\]

- constraint satisfaction problem, testing whether

\[
\min_{x \in D} \{ 0 | C \}
\]

is equal to 0 or ∞
find the largest lowerbound on $f(x)$ that can be derived from $C$

$$\max_{P \in \mathcal{P}} \{ v | C \vdash^P (f(x) \geq v) \}$$

$v$ is a lower bound on the solution of the primal problem, this property is known as weak duality.

the difference between the best lower bound given a proof system $\mathcal{P}$ and the best solution to the primal is called the duality gap.

a proof system or family is complete if there is no duality gap, this is known as strong duality.
Inference Dual

- can be used for nogood-based search
  - exclude portions of the search space that already have been examined in some way
  - say we are splitting domains i.e. adding constraints
  - let $B$ be the set of constraints that have been added

  $$\max_{P \in \mathcal{P}} \{ v | \mathcal{C} \cup B \vdash^{P} (f(x) \geq v) \}$$

- say $(\overline{v}, P)$ solves the dual
- now we identify a subset $\mathcal{N}$ which includes all constraints used as premises in $P$
- we can add to $\mathcal{C}$ the nogood

  $$\mathcal{N} \rightarrow (f(x) \geq \overline{v})$$

- we can use this to guide the search by avoiding branches where $\overline{v}$ is large
can be used for sensitivity analysis where we would like to
determine the sensitivity of the optimal value to a certain
variable

- say we have found an optimal solution to both the primal and
the dual, $z^*$ and $(v^*, P^*)$ respectively
- simple sensitivity statement: all constraints not in the premises
to $P^*$ can be altered or dropped without reducing the optimal
value below $v^*$
- if there is no duality gap the optimal value will not decrease
- in other words as long as we do not invalidate the optimal
solution it does not change
- interesting is checking how much we can alter individual
constraints without invalidating the proof
a parameterized relaxation of an optimization problem can be written as

$$\min_{x \in D} \{ f(x, u) | C(u) \}$$

where $u \in U$ is a vector of dual variables.

$C(u)$ is a relaxation of $C$, every $x \in D$ that satisfies $C$ also satisfies $C(u)$

consequently, the result of the relaxation dual is a lowerbound on the solution to the primal problem
the relaxation dual is finding a vector $u \in U$ such that the lower-bound is maximized, formalized:

$$\max\{\min_{u \in U, x \in D}\{f(x, u)|C(u)\}\}$$

we can use this lowerbound for example in branch and bound

a commonly used relaxation is dropping the integrality constraint
Summary

What I would like you to remember when solving a CSP:

- find out whether you are lucky
  - primal is a tree?
  - primal is chordal and every maximal clique is the scope of a constraint?
- if not lucky, use decomposition
  - Tree decomposition is most general therefore it dominates all other methods, however it does not give a method to compute the decomposition
  - ”In this paper, we give, for constant $k$, a linear time algorithm, that given a graph $G = (V, E)$, determines whether the treewidth of $G$ is at most $k$, and if so, finds a tree-decomposition of $G$ with treewidth at most $k.”$ , Hans L. Bodlaender
  - Solving the problem can be thus be done in time exponential in the treewidth
  - Question to Hans: does this algorithm work for hypergraphs as well?
Any Questions left?