Tactic-Driven Synthesis of Global Search Algorithms based on Constraint Satisfaction

Srinivas Nedunuri
University of Texas at Austin

Advisor: Prof. William Cook

April 11, 2009

Outline

Motivation

- Some programs are difficult for a person to write manually
  - difficulty is ensuring correctness, efficiency, and optimality
  - often involve search
  - examples are planning, scheduling, and configuration
  - eventual goal: *synthesize* such programs from their *specification* by a *methodical* process
- We have looked at simpler (but still interesting) algorithmic problems that can be expressed as constraint satisfaction problems

Motivation

- Some programs are difficult for a person to write manually
  - difficulty is ensuring correctness, efficiency, and optimality
  - often involve search
  - examples are planning, scheduling, and configuration
  - eventual goal: *synthesize* such programs from their *specification* by a *methodical* process
- We have looked at simpler (but still interesting) algorithmic problems that can be expressed as constraint satisfaction problems

What is Constraint Satisfaction?

Constraint Satisfaction

Given a set of variables, $\{v\}$, assign a value, drawn from some domain D_v, to each variable, in a manner that satisfies a given set of constraints.

- Many problems can be expressed as constraint satisfaction problems
  - Knapsack problems
  - Graph problems
  - Integer Programming
- We want to show that doing so leads to efficient algorithms
- There are good off-the-shelf generic constraint satisfaction solvers available
- Our eventual goal is the synthesis of fast *problem-specific* constraint satisfaction solvers

What is Constraint Satisfaction?

Constraint Satisfaction

Given a set of variables, $\{v\}$, assign a value, drawn from some domain D_v, to each variable, in a manner that satisfies a given set of constraints.

- Many problems can be expressed as constraint satisfaction problems
  - Knapsack problems
  - Graph problems
  - Integer Programming
- We want to show that doing so leads to efficient algorithms
- There are good off-the-shelf generic constraint satisfaction solvers available
- Our eventual goal is the synthesis of fast *problem-specific* constraint satisfaction solvers

Example problem for which we will synthesize a constraint satisfaction algorithm

Maximum Independent Segment Sum (MISS)

Maximize the sum of a selection of elements from a given array, with the restriction that no two adjacent elements can be selected.

The synthesis approach we follow starts with a formal specification of the problem..

Format of Specifications

Structure of Specification

- An input type, D
- A result type, R
- A cost type, C
- An output condition (postcondition), $o : D \times R \longrightarrow Boolean$
- A cost criterion, $cost : D \times R \longrightarrow C$

A Specification Instance: Maximum Independent Segment Sum (MISS)

Instantiation for MISS

$$\begin{array}{rcl}
D & \mapsto & maxVar : Nat \times vals : \{D_v\} \times data : [Int] \\
D_v & = & \{False, True\} \\
R & \mapsto & m : Map(Nat \to D_v) \times cs : \{D_v\} \\
C & \mapsto & Int \\
o & \mapsto & \lambda(x,z).\ dom(z.m) = \{1..(x.maxVar)\} \wedge nonAdj(z) \\
nonAdj & = & \lambda z.\ \forall i : 1 \leq i < \#z.m.\ z_i \Rightarrow \neg z_{i+1} \\
cost & \mapsto & \lambda(x,z).\ -\sum_{i=1}^{\#z}(z_i \to x_i \,|\, 0)
\end{array}$$

Example

3	9	-2	-10	0	1	x.data
1	2	3	4	5	6	1.. x.maxVars
T	F	F	T	F	T	z.m

Solve It Using Search

Take the *solution space* (potentially infinite) and partition it. Each element of the partition is called a *subspace*, and is recursively partitioned until a singleton space is encountered, called a *solution*[1]

Partial Solution or Space ($\widehat{z}$)

An assignment to some of the variables. Can be extended into a (complete) solution by assigning to all the variables.

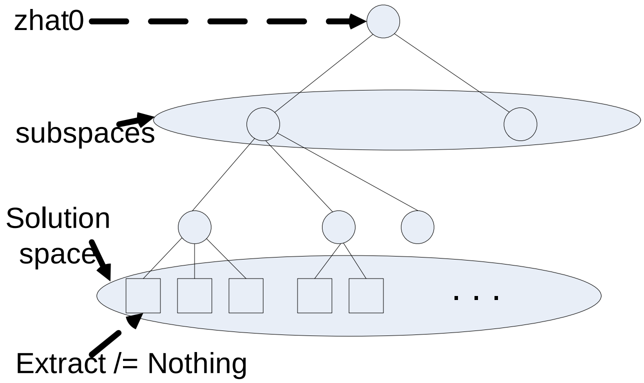
Feasible Solution (z)

A solution which satisfies the output condition

[1]based on N. Agin, "Optimum Seeking with Branch and Bound", Mgmt. Sci. 1966

Picture

The recursive search procedure just described leads to a tree structure, called a *search tree*



An algorithm class

Global Search with Optimization (GSO)

- An algorithm class that consists of a *program schema* (template) containing *operators* whose semantics is axiomatically defined
- operators must be instantiated by the user (developer). They are typically *calculated* (Dijkstra style)
- Two groups of operators: the basic space forming ones and more advanced ones which control the search.

The “Space Forming” Operators

GSO Extension

Operator	Type	Description
extract	$D \times R \rightarrow R$	determines whether the given space corresponds to a leaf node, returns it if so, otherwise Nothing
subspaces	$D \times R \rightarrow \{R\}$	partitions the given space into subspaces
$\sqsubseteq$	$\{R \times R\}$	if $r \sqsubseteq s$ then s is a subspace of r (any solution contained in s is contained in r)
$\widehat{z}_0$	D$\rightarrow R$	forms the initial space (root node)

These can usually be written down by inspection of the problem

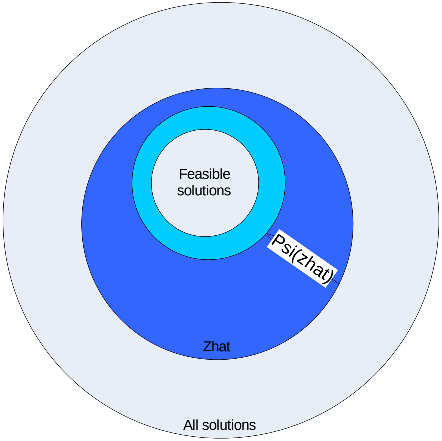
The “Search Control” Operators

GSO Extension

Operator	Called	Type	Description
Φ	Necessary Filter	$D \times R \to$ *Boolean*	Necessary condition for a space to contain feasible solutions
$\psi(\xi)$	Necessary (Consistent) Tightener	$D \times R \to R$	Tightens a given space to eliminate infeasible solutions. Preserves *all* (*at least one*) feasible solutions
lb (ib)	Lower (Initial) Bound	$D \times R \to C$	returns a lower(inital) bound on the cost of the best solution in the given space

These are usually derived from their specification

Illustration of Search Control Operator ψ



Program Schema for GSO class

```
function fo :: D -> {R}
fo(x) =
    if phi($x, r_0(x)$) $\wedge$lb(x,$r_0$(x))$\leq$ib(x)
    then f_gso(x,{$r_0(x)$},{})
    else {}

function f_gso :: D x {R} x {R} -> {R}
f_gso(x, active, solns) =
    if empty(active)
    then solns
    else let
           ($r$, rest) = arbsplit(active)
           solns' = opt(c, solns $\cup${z | extract(z,$r$)$\wedge$o(x,z)})
           ok_subs = {propagate(x,$s$) :
                                 $s$ $\in$subspaces($r$)
                                 $\wedge$ propagate(x,$s$)$\neq$Nothing}
           subs' = {$s$ : $s$ $\in$ok_subs
                          $\wedge$ lb(x,$s$)$\leq$ub(x,solns')}
         in f_gso(x, rest $\cup$ subs, solns')
```

GSO Program Schema (contd.)

```
function ub :: D x {R} -> C
ub(x, solns) =
    if empty(solns) then ib(x) else cost(x,arb(solns))
propagate x r =
    if phi(x,r) then (iterateToFixedPoint psi x r)  else Noth-
ing
iterateToFixedPoint f x z =
    let fz = f(x,z) in
    if fz=z then fz else iterateToFixedPoint f x fz
```

Operator Instantiations for MISS

We already have D, R, C, o, and *cost* (from the specification). The space forming operators can be instantiated by inspection:

Generic Instantiation (CSOT)

$$\begin{array}{rcl}
\widehat{z}_0 & \mapsto & \lambda x.\ \{m = \emptyset, cs = x.vals\} \\
subspaces & \mapsto & \lambda(x, \widehat{z}).\ \{\widehat{z}'\ :\ v = chooseVar(\{1..x.maxVar\} - dom(\widehat{z}.m)), \\
 & & \qquad\qquad \exists a \in \widehat{z}.cs.\ \widehat{z}'m = \widehat{z}.m \oplus (v \mapsto a)\} \\
extract & \mapsto & \lambda(x, \widehat{z}).\ dom(\widehat{z}.m) = \{1..x.maxVar\} \rightarrow \widehat{z} \mid Nothing \\
\sqsubseteq & \mapsto & \{(\widehat{z}, \widehat{z}') | \widehat{z}.m \subseteq \widehat{z}'.m\} \\
ib & \mapsto & maxBound
\end{array}$$

$\oplus$ denotes adding a pair to a map and is defined as

$$m \oplus (x \mapsto a) \triangleq m - \{(x, a')\} \cup \{(x, a)\}$$

The search control operators Φ, ψ, lb are calculated (not shown). We now have a working implementation of an algorithm for MISS.

Are we done?

- With this instantiation, the abstract program is correctly instantiated into a working solver. But even with all the search control operator instantiations it still has exponential complexity! (The search space grows exponentially).
- So we incorporate a concept that has been used in operations research for several decades: *dominance relations*

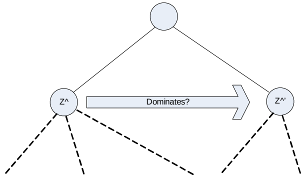
Are we done?

- With this instantiation, the abstract program is correctly instantiated into a working solver. But even with all the search control operator instantiations it still has exponential complexity! (The search space grows exponentially).
- So we incorporate a concept that has been used in operations research for several decades: *dominance relations*

Dominance Relations

What are dominance relations?

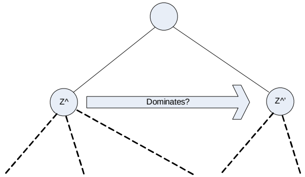
- Enables the comparison of one partial solution with another to determine if one of them can be discarded
- Given $\widehat{z}$ and $\widehat{z}'$ if the best possible solution in $\widehat{z}$ is better than the best possible solution in $\widehat{z}'$ then $\widehat{z}'$ can be discarded



Dominance Relations

What are dominance relations?

- Enables the comparison of one partial solution with another to determine if one of them can be discarded
- Given $\widehat{z}$ and $\widehat{z}'$ if the best possible solution in $\widehat{z}$ is better than the best possible solution in $\widehat{z}'$ then $\widehat{z}'$ can be discarded



Restricted dominance

To make the calculation of dominance a bit easier, we borrow a technique used by Smith to demonstrate a slightly different form of dominance. First some notation and definitions:

- We overload $\oplus$ in $\widehat{z} \oplus e$ to be a new partial solution $\widehat{z}'$ for which $\widehat{z}'.m = \widehat{z}.m \cup e.m$ (unless otherwise stated, we will always be assuming $dom(\widehat{z}.m) \cap dom(e.m) = \emptyset$)
- e is called an *extension*.
- When $\widehat{z} \oplus e$ is a (feasible) complete solution, e is called the *(feasible) completion* of $\widehat{z}$.

Informally, if $\widehat{z}$ is semi-congruent with $\widehat{z}'$ then any feasible completion of $\widehat{z}'$ is a feasible completion of$\widehat{z}$

Restricted dominance

To make the calculation of dominance a bit easier, we borrow a technique used by Smith to demonstrate a slightly different form of dominance. First some notation and definitions:

- We overload $\oplus$ in $\widehat{z} \oplus e$ to be a new partial solution $\widehat{z}'$ for which $\widehat{z}'.m = \widehat{z}.m \cup e.m$ (unless otherwise stated, we will always be assuming $dom(\widehat{z}.m) \cap dom(e.m) = \emptyset$)
- e is called an *extension*.
- When $\widehat{z} \oplus e$ is a (feasible) complete solution, e is called the *(feasible) completion* of $\widehat{z}$.

Informally, if $\widehat{z}$ is semi-congruent with $\widehat{z}'$ then any feasible completion of $\widehat{z}'$ is a feasible completion of$\widehat{z}$

Definitions

Definition: Semi-Congruence

is a relation $\rightsquigarrow \subseteq R^2$ such that

$$\forall e, \widehat{z}, \widehat{z}' \in R : \ \widehat{z} \rightsquigarrow \widehat{z}' \ \Rightarrow o(\widehat{z}' \oplus e) \Rightarrow o(\widehat{z} \oplus e)$$

Then we need to say something about when one space is "better" than another. We call this weak dominance. if $\widehat{z}$ weakly dominates $\widehat{z}'$, then any feasible completion of $\widehat{z}$ is no more expensive than the same feasible completion of $\widehat{z}'$

Definition: Weak Dominance

is a relation $\widehat{\delta} \subseteq R^2$ such that

$$\forall e, \widehat{z}, \widehat{z}' \in R : \ \widehat{z}\widehat{\delta}\widehat{z}' \Rightarrow \ o(\widehat{z} \oplus e) \wedge o(\widehat{z}' \oplus e) \Rightarrow c(\widehat{z} \oplus e) \leq c(\widehat{z}' \oplus e)$$

Definitions

Definition: Semi-Congruence

is a relation $\rightsquigarrow \subseteq R^2$ such that

$$\forall e, \widehat{z}, \widehat{z}' \in R : \; \widehat{z} \rightsquigarrow \widehat{z}' \; \Rightarrow o(\widehat{z}' \oplus e) \Rightarrow o(\widehat{z} \oplus e)$$

Then we need to say something about when one space is "better" than another. We call this weak dominance. if $\widehat{z}$ weakly dominates $\widehat{z}'$, then any feasible completion of $\widehat{z}$ is no more expensive than the same feasible completion of $\widehat{z}'$

Definition: Weak Dominance

is a relation $\widehat{\delta} \subseteq R^2$ such that

$$\forall e, \widehat{z}, \widehat{z}' \in R : \; \widehat{z}\widehat{\delta}\widehat{z}' \Rightarrow \; o(\widehat{z} \oplus e) \wedge o(\widehat{z}' \oplus e) \Rightarrow c(\widehat{z} \oplus e) \leq c(\widehat{z}' \oplus e)$$

Definitions

Definition: Semi-Congruence

is a relation $\rightsquigarrow \subseteq R^2$ such that

$$\forall e, \widehat{z}, \widehat{z}' \in R : \ \widehat{z} \rightsquigarrow \widehat{z}' \ \Rightarrow o(\widehat{z}' \oplus e) \Rightarrow o(\widehat{z} \oplus e)$$

Then we need to say something about when one space is "better" than another. We call this weak dominance. if $\widehat{z}$ weakly dominates $\widehat{z}'$, then any feasible completion of $\widehat{z}$ is no more expensive than the same feasible completion of $\widehat{z}'$

Definition: Weak Dominance

is a relation $\widehat{\delta} \subseteq R^2$ such that

$$\forall e, \widehat{z}, \widehat{z}' \in R : \ \widehat{z}\widehat{\delta}\widehat{z}' \Rightarrow \ o(\widehat{z} \oplus e) \wedge o(\widehat{z}' \oplus e) \Rightarrow c(\widehat{z} \oplus e) \leq c(\widehat{z}' \oplus e)$$

Dominance Relations (contd.)

To get a dominance test, combine the two

Theorem (Dominance)

$$\forall \widehat{z}, \widehat{z}' \in R : \ \widehat{z} \widehat{\delta} \widehat{z}' \wedge \widehat{z} \rightsquigarrow \widehat{z}' \Rightarrow cost^*(\widehat{z}) \leq cost^*(\widehat{z}')$$

ie., if $\widehat{z}$ is semi-congruent with $\widehat{z}'$ and $\widehat{z}$ weakly dominates $\widehat{z}'$ then the cost of the best solution in $\widehat{z}$ is no more than the cost of the best solution in $\widehat{z}'$

When $cost^*(\widehat{z}) \leq cost^*(\widehat{z}')$ we say $\widehat{z}$ *dominates* $\widehat{z}'$, written $\widehat{z}\,\delta\,\widehat{z}'$

How does this fit into CSOT? Following is a cheap way to get a weak-dominance condition:

Theorem (Cost Distribution)

If *cost* distributes over $\oplus$ and $cost(\widehat{z}) \leq cost(\widehat{z}')$ then $\widehat{z}\ \widehat{\delta} \widehat{z}'$

Dominance Relations (contd.)

To get a dominance test, combine the two

Theorem (Dominance)

$$\forall \widehat{z}, \widehat{z}' \in R : \ \widehat{z}\widehat{\delta}\widehat{z}' \wedge \widehat{z} \rightsquigarrow \widehat{z}' \Rightarrow cost^*(\widehat{z}) \leq cost^*(\widehat{z}')$$

ie., if $\widehat{z}$ is semi-congruent with $\widehat{z}'$ and $\widehat{z}$ weakly dominates $\widehat{z}'$ then the cost of the best solution in $\widehat{z}$ is no more than the cost of the best solution in $\widehat{z}'$

When $cost^*(\widehat{z}) \leq cost^*(\widehat{z}')$ we say $\widehat{z}$ *dominates* $\widehat{z}'$, written $\widehat{z}\,\delta\,\widehat{z}'$
How does this fit into CSOT? Following is a cheap way to get a weak-dominance condition:

Theorem (Cost Distribution)

If *cost* distributes over $\oplus$ and $cost(\widehat{z}) \leq cost(\widehat{z}')$ then $\widehat{z}\ \widehat{\delta}\widehat{z}'$

..Back to MISS

First calculate the semi-congruence condition $\leadsto$ between $\widehat{z}$ and $\widehat{z}'$. Working backwards from the conclusion of the definition of semi-congruence:

$$
\begin{array}{l}
o(\widehat{z} \oplus e) \\
= \{\text{unfold defn, let } L = \#\widehat{z}, L' = \#\widehat{z}'\} \\
dom(\widehat{z}.m) + dom(e.m) = [1..(x.maxVar)] \\
\wedge\, nonAdj(\widehat{z}) \,\wedge\, nonAdj(e) \,\wedge\, (\widehat{z}_L \Rightarrow \neg e_1) \\
\Leftarrow \{nonAdj(\widehat{z}') \wedge nonAdj(e) \wedge (\widehat{z}'_L \Rightarrow \neg e_1), \text{ from.} o(\widehat{z}' \oplus e)\} \\
dom(\widehat{z}.m) + dom(e.m) = [1..(x.maxVar)] \\
\quad \wedge\, nonAdj(\widehat{z}) \,\wedge\, ((\widehat{z}'_L \Rightarrow \neg e_1) \Rightarrow (\widehat{z}_L \Rightarrow \neg e_1)) \\
= \{\text{anti-monotonicity of } (k \Leftarrow)\} \\
dom(\widehat{z}.m) + dom(e.m) = [1..(x.maxVar)] \\
\quad \wedge\, nonAdj(\widehat{z}) \,\wedge\, (\neg\widehat{z}'_L \Rightarrow \neg\widehat{z}_L) \\
= \{\text{vars assigned consecutively } \& dom(\widehat{z}'.m) + dom(e.m) = [1..(x.maxVar)]\} \\
L = L' \,\wedge\, nonAdj(\widehat{z}) \,\wedge\, (\neg\widehat{z}'_L \Rightarrow \neg\widehat{z}_L) \\
= \{\text{simplification}\} \\
L = L' \,\wedge\, nonAdj(\widehat{z}) \,\wedge\, (\widehat{z}_L \Rightarrow \widehat{z}'_L)
\end{array}
$$

Dominance Relation for MISS

Since c is a distributive cost function, the definition for δ follows immediately: $\widehat{z} \rightsquigarrow \widehat{z}' \wedge cost(\widehat{z}) \leq cost(\widehat{z}')$
This dominance test reduces the complexity of the MISS algorithm from exponential to polynomial.

Greedy Algorithms

Definition

A greedy algorithm is one which repeatedly makes a locally optimal choice

For some problems, this leads to a globally optimal solution. Some attempts to capture the class of such problems:

- Weighted Matroids, Greedy Algorithm - Whitney (1935), Edmonds (1971)
- Specialization of Dynamic Programming - Bird and deMoor (1993)
- Property Based Classification - Curtis (2003)

Weighted Matroids (on one slide)

Weighted Matroid (Whitney)

A structure $\langle S, I, w\rangle$ where S is a finite set, I a collection of *independent* subsets of S, $w : I \rightarrow \mathbb{N}$ is a weight function, satisfying the following axioms:

- (Nonemptiness)$\{\} \in I$
- (Hereditary Property) if $B \in I$ and $A \subseteq B$ then $A \in I$
- (Exchange Property) if $A, B \in I$ with $|A| > |B|$ then $\exists x \in A - B.\ B \cup \{x\} \in I$

- Greedy Algorithm (Edmonds) finds the subset in I with largest w value
- Examples: Kruskal's MST algorithm & algorithm for solving $1/1/U_i$
- But *not*: Prim's MST algorithm or Huffman's min. length

Weighted Matroids (on one slide)

Weighted Matroid (Whitney)

A structure $\langle S, I, w \rangle$ where S is a finite set, I a collection of *independent* subsets of S, $w : I \to \mathbb{N}$ is a weight function, satisfying the following axioms:

- (Nonemptiness)$\{\} \in I$
- (Hereditary Property) if $B \in I$ and $A \subseteq B$ then $A \in I$
- (Exchange Property) if $A, B \in I$ with $|A| > |B|$ then $\exists x \in A - B.\ B \cup \{x\} \in I$

- Greedy Algorithm (Edmonds) finds the subset in I with largest w value
- Examples: Kruskal's MST algorithm & algorithm for solving $1/1/U_i$
- But *not*: Prim's MST algorithm or Huffman's min. length

Weighted Matroids (on one slide)

Weighted Matroid (Whitney)

A structure $\langle S, I, w \rangle$ where S is a finite set, I a collection of *independent* subsets of S, $w : I \rightarrow \mathbb{N}$ is a weight function, satisfying the following axioms:

- (Nonemptiness)$\{\} \in I$
- (Hereditary Property) if $B \in I$ and $A \subseteq B$ then $A \in I$
- (Exchange Property) if $A, B \in I$ with $|A| > |B|$ then $\exists x \in A - B.\ B \cup \{x\} \in I$

- Greedy Algorithm (Edmonds) finds the subset in I with largest w value
- Examples: Kruskal's MST algorithm & algorithm for solving $1/1/U_i$
- But *not*: Prim's MST algorithm or Huffman's min. length

We propose: Dominance Based Characterization of Greedy Algorithms

Claim

Let $\langle subspaces', \delta \rangle$ be the set of subspaces of the current active space ($\widehat{z}$) remaining after tightening and filtering, ordered by δ[a]. Then the least element of *subspaces′* (if it exists) is the optimal greedy choice.

[a] technically $\delta'(x)$, where δ' is the curried form of the dominance relation δ

Example: $1//\sum C_i$

We can derive a dominance relation for this problem. We can then show that that among the subspaces of a space, there is a least subspace. By the claim, this is the optimal greedy choice. Repeatedly carrying out this greedy choice results in an algorithm that is equivalent to the Shortest Processing Time first rule discovered by W.E. Smith in 1956

And now for something completely different..

- Calculation of the operators ψ, Φ, δ, etc. can be sometimes be non-obvious.
- Provide hints to the developer based on the shape of the formula.
- We will illustrate with two examples of calculation the long way and then shortcut using tactics

Example 1: 01-ILP

Example: Binary Integer Linear Programming (01-ILP)

Given $\mathbf{A}, \mathbf{b}$, find a z satisfying $\mathbf{A} \cdot \mathbf{z} \leq \mathbf{b}$ where $z \in \{0,1\}^*$

$$\begin{array}{rcl}
D & \mapsto & \mathit{maxVar} : \mathit{Nat}, \mathit{vals} : \{D_v\}, \\
 & & \mathbf{A} : \mathit{Map}(\{1..M\}, \{1..N\} \mapsto \mathit{Real}), \mathbf{b} : \mathit{Map}(\{1..N\} \mapsto \mathit{Real}) \\
D_v & \mapsto & \{0,1\} \\
R & \mapsto & m : \mathit{Map}(\mathit{Nat} \to D_v) \\
o & \mapsto & \lambda(x,z).\ \mathit{dom}(z.m) = \{1..(x.\mathit{maxVar})\} \wedge (x.\mathbf{A}) \cdot (z.m) \leq x.\mathbf{b}
\end{array}$$

Calculate Filter for 01-ILP

$o(x, z)$
$\Rightarrow$ {relevant part of o}
$(x.\mathbf{A}) \cdot (z.m) \leq x.\mathbf{b}$
$=$ {defn of dot product}
$\forall i \in [1..M].\ \sum A_{ij} \cdot z_j \leq b_i$
$=$ {$\widehat{z} \sqsubseteq z$, let $z = \widehat{z} \oplus e$ where $\widehat{z}$ is a partial solution, e an *extension*}
$\forall i \in [1..M].\ \sum_{j \in dom(\widehat{z}.m)} A_{ij} \cdot \widehat{z}_j + \sum_{j \in dom(e.m)} A_{ij} \cdot e_j \leq b_i$
$=$ {property of min, and transitivity}
$\forall i \in [1..M].\ \sum_{j \in dom(\widehat{z}.m)} A_{ij} \cdot \widehat{z}_j + \sum_{j \in dom(e.m)} \min_{a \in \{0,1\}} \{A_{ij} \cdot a\} \leq b_i$

the last line is something of a “rabbit”[a]

[a] Dijkstra’s term for a step that appears seemingly out of nowhere

Example 2: Calculate Filter for MISS

$$cost(x,z) \leq c^*$$
$$=$$
$$-\sum_{i=1}^{\#z}(z_i \rightarrow x_i \mid 0) \leq c^*$$
$$=$$
$$-\sum_{i=1}^{\#\hat{z}}(\hat{z}_i \rightarrow x_i \mid 0) - \sum_{i=\#\hat{z}+1}^{\#z}(e_i \rightarrow x_i \mid 0) \leq c^*$$
$$=$$
$$-\sum_{i=1}^{\#\hat{z}}(\hat{z}_i \rightarrow x_i \mid 0) - \sum_{i=\#\hat{z}+1}^{\#z} max(x.data(i), 0) \leq c^*$$
$$=$$
$$cost(x,\hat{z}) - \sum_{i=\#\hat{z}+1}^{\#z} max(x.data(i), 0) \leq c^*$$

Notice similarity with shape of derivation of filter for 01-ILP

What we'd like to do

- Provide hints to the developer about where to begin based on the shape of the formula
- Elide actual tedious calculation by the use of pattern matching rules
- Call such aids *tactics*
- Analogous in intent to tactics used in integration eg. "integration by parts", "integration by partial fractions", "integration by change of variable", etc.

What we'd like to do

- Provide hints to the developer about where to begin based on the shape of the formula
- Elide actual tedious calculation by the use of pattern matching rules
- Call such aids *tactics*
- Analogous in intent to tactics used in integration eg. "integration by parts", "integration by partial fractions", "integration by change of variable", etc.

What we'd like to do

- Provide hints to the developer about where to begin based on the shape of the formula
- Elide actual tedious calculation by the use of pattern matching rules
- Call such aids *tactics*
- Analogous in intent to tactics used in integration eg. "integration by parts", "integration by partial fractions", "integration by change of variable", etc.

Tactic for constructing a filter (Φ)

Theorem

If

$$o(x,z) \Rightarrow \bigotimes_{i \in I} F_i(z_i) \preceq K$$

for:

some $K : T$

some family of functions $\{F_i : D_v \rightarrow T\}$

$\bigotimes : T \times T \rightarrow T$ *a monotone associative operator*

$\preceq$ *forms a meet semi-lattice over* T*,*

then

$$o(x, \widehat{z} \oplus e) \Rightarrow \left(\bigotimes_{1 \leq i \leq \# \widehat{z}} F_i(\widehat{z}_i) \otimes \bigotimes_{1 \leq i \leq \# e} f_i \right) \preceq K$$

where $f_i = \sqcap_{a \in x.vals} F_i(a)$

Go back and apply the tactic to 01-ILP

Look at the interesting part of o:

$$(x.\mathbf{A}) \cdot (z.m) \leq x.\mathbf{b}$$

and "match" it against

$$\bigotimes_{j \in I} F_j(z_j) \preceq K$$

with a substitution

$$\begin{array}{ccc} \otimes & \mapsto & + \\ F_j & \mapsto & (A_{ij} \cdot) \\ \preceq & \mapsto & \leq \end{array}$$

(for each i) which yields the $\Phi(x, \widehat{z})$ we calculated earlier

$$\forall i.\, 1 \leq i \leq 1. \sum_{j \in dom(\widehat{z}.m)} \mathbf{A}_{ij} \cdot \widehat{z}_j + \sum_{j \in dom(e.m)} (\min_{a \in \{0,1\}} \{\mathbf{A}_{ij} \cdot a\}) \leq \mathbf{b}_i$$

(this can be simplified by a simple case analysis and algebra)

We can do the same for MISS

Look at the form of the bound

$$cost(x, z) \leq c^*$$

and “match” it against

$$\bigotimes_{j \in I} F_j(z_j) \preceq K$$

with a substitution

$$\begin{array}{ccl} \otimes & \mapsto & + \\ F_j & \mapsto & \lambda z_j.\ z_i \rightarrow x.data(j) \mid 0 \\ \preceq & \mapsto & \leq \end{array}$$

.which immediately gives us the bounds test we derived earlier:

$$cost(x, \widehat{z}) - \sum_{i=\#\widehat{z}+1}^{\#x.data} max(x.data(i), 0) \leq c^*$$

Other Tactics We Have Discovered

- Tactic for constructing a *tightener* (tightens a space to remove infeasible solutions)
  - Used to derive tightener for the MISS problem that (along with finite differencing) reduced the algorithm complexity from quadratic to linear, allowing us to outperform a published algorithm derived by program transformation.
- Tactics for constructing *dominance relations* (compares two partial solutions to see if one can be discarded)
  - Used to derive two dominance relations for the Unbounded Knapsack Problem (UKP), one of which is the basis of a previously unpublished greedy algorithm for the UKP problem.
  - Also dominance relations for the 1-machine scheduling problem (with 2 different optimality criteria) which forms the basis for another greedy algorithm.

Other Tactics We Have Discovered

- Tactic for constructing a *tightener* (tightens a space to remove infeasible solutions)
    - Used to derive tightener for the MISS problem that (along with finite differencing) reduced the algorithm complexity from quadratic to linear, allowing us to outperform a published algorithm derived by program transformation.
- Tactics for constructing *dominance relations* (compares two partial solutions to see if one can be discarded)
    - Used to derive two dominance relations for the Unbounded Knapsack Problem (UKP), one of which is the basis of a previously unpublished greedy algorithm for the UKP problem.
    - Also dominance relations for the 1-machine scheduling problem (with 2 different optimality criteria) which forms the basis for another greedy algorithm.

What have we done so far?

- Proposed a theory within GSO for constraint satisfaction problems (CSO)
  - Incorporated dominance relations into CSO
  - Shown how dominance relations can be usefully applied to CSO problems
  - Shown how dominance relations could form the basis of near-greedy algorithms
- Tactics
  - Motivated the need for tactics
  - Proposed tactics for assisting a developer, and gave examples of where they have been applied
  - Showed how one of the tactics could be used to elide calculation

What have we done so far?

- Proposed a theory within GSO for constraint satisfaction problems (CSO)
  - Incorporated dominance relations into CSO
  - Shown how dominance relations can be usefully applied to CSO problems
  - Shown how dominance relations could form the basis of near-greedy algorithms
- Tactics
  - Motivated the need for tactics
  - Proposed tactics for assisting a developer, and gave examples of where they have been applied
  - Showed how one of the tactics could be used to elide calculation

Further Work

1. Investigate optimality analogues of ψ and ξ
2. Formal proofs of claims
3. Formalize tactics, discover others
4. Applications

1. Investigate Optimality Analogues of ψ and ξ

Necessary Optimality Tightener ν_0

- Tightens a given space to eliminate unoptimal solutions. Retains all optimal solutions
- Defined as any predicate ν_o over $D \times R$ satisfying

$$\forall x, z, \widehat{z}.\ \widehat{z} \sqsubseteq z \ \wedge\ z \in opt(x, cost, \widehat{z}) \Rightarrow \nu_o(x, \widehat{z}) \sqsubseteq z$$

Consistent Optimality Tightener κ_0

- Tightens a given space to eliminate unoptimal solutions. Retains at least one optimal solution
- Defined as any predicate κ_o over $D \times R$ satisfying

$$\forall x, \widehat{z}.\ (\exists z.\ \widehat{z} \sqsubseteq z \wedge z \in opt(x, cost, \widehat{z})) \Rightarrow (\exists z.\ \kappa_o(x, \widehat{z}) \sqsubseteq z \wedge z \in opt(x, cost, \widehat{z}$$

1. Investigate Optimality Analogues of ψ and ξ

Necessary Optimality Tightener ν_0

- Tightens a given space to eliminate unoptimal solutions. Retains all optimal solutions
- Defined as any predicate ν_o over $D \times R$ satisfying

$$\forall x, z, \hat{z}.\ \hat{z} \sqsubseteq z \ \wedge\ z \in opt(x, cost, \hat{z}) \Rightarrow \nu_o(x, \hat{z}) \sqsubseteq z$$

Consistent Optimality Tightener κ_0

- Tightens a given space to eliminate unoptimal solutions. Retains at least one optimal solution
- Defined as any predicate κ_o over $D \times R$ satisfying

$$\forall x, \hat{z}.\ (\exists z.\ \hat{z} \sqsubseteq z \wedge z \in opt(x, cost, \hat{z})) \Rightarrow (\exists z.\ \kappa_o(x, \hat{z}) \sqsubseteq z \wedge z \in opt(x, cost, \hat{z}$$

Example of ν_o (for MISS)

Spec. of MISS problem (from earlier)

$$
\begin{array}{rcl}
D & \mapsto & maxVar : Nat \times vals : \{D_v\} \times data : [Int] \\
D_v & = & \{False, True\} \\
R & \mapsto & m : Map(Nat \rightarrow D_v) \times cs : \{D_v\} \\
C & \mapsto & Int \\
o & \mapsto & \lambda(x, z).\ dom(z.m) = \{1..(x.maxVar)\} \wedge nonAdj(z) \\
nonAdj & = & \lambda z.\ \forall i : 1 \leq i < \#z.m.\ z_i \Rightarrow \neg z_{i+1} \\
cost & \mapsto & \lambda(x, z).\ -\sum_{i=1}^{\#z} z_i \rightarrow x_i \mid 0
\end{array}
$$

along with

$$\sqsubseteq \mapsto \{(\widehat{z}, \widehat{z}') \mid \widehat{z}.m \subseteq \widehat{z}'.m\}$$

Example of ν_o (contd.)

Instantiation of ν_o for MISS

Plug the MISS instantiations into the definition of ν_o :

$$\begin{aligned}
&\widehat{z}.m \subseteq z.m \land CSOT.o \land o(x,z) \\
&\quad \land \forall z'.\ o(x,z') \Rightarrow cost(x,z) \leq cost(x,z') \\
&\Rightarrow \\
&\nu_0(\widehat{z}).m \subseteq z.m
\end{aligned}$$

Calculation yields $\nu_0(\widehat{z}) = x_{\#\widehat{z}+1} < 0 \rightarrow (\widehat{z} \oplus False)|\widehat{z}$. That is, for an optimum solution, $\widehat{z}$ is extended with the next variable assigned *False* exactly when the next data value is negative.

Investigate Solution Space Reductions

Often take the form: “nothing is lost by restricting attention to ...”
Examples:

- In linear programming, there is a theorem that states that if an optimal feasible solution exists in the polytope defined by the constraints, then it can be found at the vertices of the polytope
- In scheduling, there is a result that says that when looking for optimum schedules, it is sufficient to restrict attention to semi-active schedules
- When locating absolute p-medians of a graph, it is sufficient to look at p-medians

How can these be discovered for other kinds of problems?

Investigate Solution Space Reductions

Often take the form: “nothing is lost by restricting attention to ...”
Examples:

- In linear programming, there is a theorem that states that if an optimal feasible solution exists in the polytope defined by the constraints, then it can be found at the vertices of the polytope
- In scheduling, there is a result that says that when looking for optimum schedules, it is sufficient to restrict attention to semi-active schedules
- When locating absolute p-medians of a graph, it is sufficient to look at p-medians

How can these be discovered for other kinds of problems?

Investigate Solution Space Reductions

Often take the form: “nothing is lost by restricting attention to ...”
Examples:

- In linear programming, there is a theorem that states that if an optimal feasible solution exists in the polytope defined by the constraints, then it can be found at the vertices of the polytope
- In scheduling, there is a result that says that when looking for optimum schedules, it is sufficient to restrict attention to semi-active schedules
- When locating absolute p-medians of a graph, it is sufficient to look at p-medians

How can these be discovered for other kinds of problems?

2. Formal proofs of correctness

We have informally argued for the correctness of a number of ideas. Need to

- Establish correctness of near-greedy algorithm characterization
    - What is relationship to an existing characterization of greedy algorithms?
- Correctness of program schema with dominance relations
    - Analogous to proof of correctness of GSO schema (Smith, 1988) but build on top of the definitions for semi-congruence, and weak dominance, and the dominance theorem

2. Formal proofs of correctness

We have informally argued for the correctness of a number of ideas. Need to

- Establish correctness of near-greedy algorithm characterization
  - What is relationship to an existing characterization of greedy algorithms?
- Correctness of program schema with dominance relations
  - Analogous to proof of correctness of GSO schema (Smith, 1988) but build on top of the definitions for semi-congruence, and weak dominance, and the dominance theorem

3. Tactics and Applications

- Formalize the other tactics as pattern-matching rules
- Ultimately, the usefulness of these ideas is whether they can be applied to real problems. We would like to look at Planning
    - Srivastava and Kambhapati (1998) used KIDS to synthesize fast problem-specific planners. We would like to synthesize optimal planners, as well as tactics that generalize some of their domain specific rules
- Hope to form a library of useful tactics

The End!

3. Tactics and Applications

- Formalize the other tactics as pattern-matching rules
- Ultimately, the usefulness of these ideas is whether they can be applied to real problems. We would like to look at Planning
    - Srivastava and Kambhapati (1998) used KIDS to synthesize fast problem-specific planners. We would like to synthesize optimal planners, as well as tactics that generalize some of their domain specific rules
- Hope to form a library of useful tactics

The End!

3. Formalize the tactics & discover others

- With the exception of the Φ tactic, the tactics are currently expressed somewhat informally
  - in the style of design patterns
  - provides developer with idea of where to start
  - also serve communication purpose ("I used the capacity tactic here")
- Ultimately, capture as pattern-matching rules
  - allows developer to bypass much tedious calculation
  - similar to the tactics used in integration
- Form a useful library of tactics

3. Formalize the tactics & discover others

- With the exception of the Φ tactic, the tactics are currently expressed somewhat informally
  - in the style of design patterns
  - provides developer with idea of where to start
  - also serve communication purpose ("I used the capacity tactic here")
- Ultimately, capture as pattern-matching rules
  - allows developer to bypass much tedious calculation
  - similar to the tactics used in integration
- Form a useful library of tactics

3. Formalize the tactics & discover others

- With the exception of the Φ tactic, the tactics are currently expressed somewhat informally
  - in the style of design patterns
  - provides developer with idea of where to start
  - also serve communication purpose ("I used the capacity tactic here")
- Ultimately, capture as pattern-matching rules
  - allows developer to bypass much tedious calculation
  - similar to the tactics used in integration
- Form a useful library of tactics

4. Applications

Ultimately, the usefulness of these ideas is whether they can be applied to real problems. We would like to look at two:

- Planning
    - Srivastava and Kambhapati (1998) used KIDS to synthesize fast custom planners. We would like to synthesize optimal planners, as well as tactics that generalize some of their domain specific rules
- Mapping abstract execution models (e.g. PIM or Real-time task spec) to actual execution environments (e.g. Platforms or actual architectures)
    - There has been work formulating both kinds of problems as search. Propose to improve on this work

4. Applications

Ultimately, the usefulness of these ideas is whether they can be applied to real problems. We would like to look at two:

- Planning
  - Srivastava and Kambhapati (1998) used KIDS to synthesize fast custom planners. We would like to synthesize optimal planners, as well as tactics that generalize some of their domain specific rules
- Mapping abstract execution models (e.g. PIM or Real-time task spec) to actual execution environments (e.g. Platforms or actual architectures)
  - There has been work formulating both kinds of problems as search. Propose to improve on this work

And Finally..

Questions?

Expected Timeline

1. Formal proofs of claims (days/weeks)
2. Formalize tactics (weeks/months)
3. Investigating optimality analogues of ψ and ξ and discovering other tactics will probably happen in the context of applying these ideas to the two application area previously mentioned (˜ 1 year)

Motivation

- Some programs are difficult for people to write manually
    - challenge is ensuring correctness *and* efficiency
    - problems involving search often in this category
    - Examples: planning, scheduling, configuration
- One solution is program synthesis
    - The approach we follow: start with a formal specification of the problem and systematically derive efficient solutions that are correct by construction

Motivation

- Some programs are difficult for people to write manually
  - challenge is ensuring correctness *and* efficiency
  - problems involving search often in this category
  - Examples: planning, scheduling, configuration
- One solution is program synthesis
  - The approach we follow: start with a formal specification of the problem and systematically derive efficient solutions that are correct by construction

Motivation

- Some programs are difficult for people to write manually
    - challenge is ensuring correctness *and* efficiency
    - problems involving search often in this category
    - Examples: planning, scheduling, configuration
- One solution is program synthesis
    - The approach we follow: start with a formal specification of the problem and systematically derive efficient solutions that are correct by construction

Motivation

- Some programs are difficult for people to write manually
  - challenge is ensuring correctness *and* efficiency
  - problems involving search often in this category
  - Examples: planning, scheduling, configuration
- One solution is program synthesis
  - The approach we follow: start with a formal specification of the problem and systematically derive efficient solutions that are correct by construction

Motivation

- Some programs are difficult for people to write manually
    - challenge is ensuring correctness *and* efficiency
    - problems involving search often in this category
    - Examples: planning, scheduling, configuration
- One solution is program synthesis
    - The approach we follow: start with a formal specification of the problem and systematically derive efficient solutions that are correct by construction

Motivation

- Some programs are difficult for people to write manually
  - challenge is ensuring correctness *and* efficiency
  - problems involving search often in this category
  - Examples: planning, scheduling, configuration
- One solution is program synthesis
  - The approach we follow: start with a formal specification of the problem and systematically derive efficient solutions that are correct by construction

The Not So Easy Operators (contd.)

Necessary Filter (Φ)

- Eliminates spaces that cannot possibly contain feasible solutions
- Ideal filter: Given a space r, test $\exists x, z.\ r \sqsubseteq z \wedge o(x, z)$
- We settle for a weaker test: Given r, test Φ, where $\forall x, z.\ r \sqsubseteq z \wedge o(x, z) \Rightarrow \Phi(x, r)$

The Not So Easy Operators (contd.)

Necessary Tightener ψ

- Tightens a given space to only eliminate infeasible solutions
- Defined as any predicate over $D \times R$ satisfying

$$\forall x, z.\ r \sqsubseteq z \wedge o(x,z) \Rightarrow \psi(r) \sqsubseteq z$$

Consistent Tightener ξ

- Tightens a given space to eliminate infeasible solutions. Preserves existence of at least one feasible solution
- Defined as any predicate over $D \times R$ satisfying

$$\forall x.\ (\exists z.\, r \sqsubseteq z \wedge o(x,z)) \Rightarrow (\exists z.\, \xi(r) \sqsubseteq z \wedge o(x,z))$$

The Not So Easy Operators (contd.)

Necessary Tightener ψ

- Tightens a given space to only eliminate infeasible solutions
- Defined as any predicate over $D \times R$ satisfying

$$\forall x, z.\ r \sqsubseteq z \wedge o(x, z) \Rightarrow \psi(r) \sqsubseteq z$$

Consistent Tightener ξ

- Tightens a given space to eliminate infeasible solutions. Preserves existence of at least one feasible solution
- Defined as any predicate over $D \times R$ satisfying

$$\forall x.\ (\exists z.\, r \sqsubseteq z \wedge o(x, z)) \Rightarrow (\exists z.\, \xi(r) \sqsubseteq z \wedge o(x, z))$$

Example: Schedule jobs on a machine so as to minimize the sum of the completion times

- Given a partial schedule $\widehat{z} = \widehat{y} \oplus \{a \mapsto s\} \oplus \{b \mapsto t\}$ and an interchange permutation of it, $\widehat{z}' = \widehat{y} \oplus \{a \mapsto t\} \oplus \{b \mapsto s\}$, (assume) we can calculate the following dominance relation

$$\delta(x, \widehat{z}, \widehat{z}') = s.p \leq t.p$$

- When a is the next variable, it can be shown the least $\widehat{z}$ in the dominance ordering is the one in which $\widehat{z}_a$ is the least.
- By the Claim above, picking such a $\widehat{z}$ is the correct greedy choice.
- This corresponds to the Shortest Processing Time first (SPT) rule, discovered by W.E. Smith in 1956.

Example: Schedule jobs on a machine so as to minimize the sum of the completion times

- Given a partial schedule $\widehat{z} = \widehat{y} \oplus \{a \mapsto s\} \oplus \{b \mapsto t\}$ and an interchange permutation of it, $\widehat{z}' = \widehat{y} \oplus \{a \mapsto t\} \oplus \{b \mapsto s\}$, (assume) we can calculate the following dominance relation

$$\delta(x, \widehat{z}, \widehat{z}') = s.p \leq t.p$$

- When a is the next variable, it can be shown the least $\widehat{z}$ in the dominance ordering is the one in which $\widehat{z}_a$ is the least.
- By the Claim above, picking such a $\widehat{z}$ is the correct greedy choice.
- This corresponds to the Shortest Processing Time first (SPT) rule, discovered by W.E. Smith in 1956.

Example: Schedule jobs on a machine so as to minimize the sum of the completion times

- Given a partial schedule $\widehat{z} = \widehat{y} \oplus \{a \mapsto s\} \oplus \{b \mapsto t\}$ and an interchange permutation of it, $\widehat{z}' = \widehat{y} \oplus \{a \mapsto t\} \oplus \{b \mapsto s\}$, (assume) we can calculate the following dominance relation

$$\delta(x, \widehat{z}, \widehat{z}') = s.p \leq t.p$$

- When a is the next variable, it can be shown the least $\widehat{z}$ in the dominance ordering is the one in which $\widehat{z}_a$ is the least.
- By the Claim above, picking such a $\widehat{z}$ is the correct greedy choice.
- This corresponds to the Shortest Processing Time first (SPT) rule, discovered by W.E. Smith in 1956.

Example: Schedule jobs on a machine so as to minimize the sum of the completion times

- Given a partial schedule $\widehat{z} = \widehat{y} \oplus \{a \mapsto s\} \oplus \{b \mapsto t\}$ and an interchange permutation of it, $\widehat{z}' = \widehat{y} \oplus \{a \mapsto t\} \oplus \{b \mapsto s\}$, (assume) we can calculate the following dominance relation

$$\delta(x, \widehat{z}, \widehat{z}') = s.p \leq t.p$$

- When a is the next variable, it can be shown the least $\widehat{z}$ in the dominance ordering is the one in which $\widehat{z}_a$ is the least.
- By the Claim above, picking such a $\widehat{z}$ is the correct greedy choice.
- This corresponds to the Shortest Processing Time first (SPT) rule, discovered by W.E. Smith in 1956.

Example of Value Choice Tactic

Example

The Unbounded Knapsack Problem (UKP)
Martello and Toth provide the following dominance relation (Theorem 3.2) for the UKP (NB: p_i refers to the profit from including the ith item in the knapsack, w_i to the weight of the item)

> *Given any instance of the UKP and item index k if there exists an item index j such that $p_k \leq \lfloor \frac{w_k}{w_j} \rfloor p_j$ then k is dominated (can be eliminated)*

Where did this come from? Can I do something similar for my problem?

Applying the tactic

Suppose we have a partial solution $\widehat{z} = \widehat{y} \oplus (\widehat{z}_k \rightarrow v_k) \oplus (\widehat{z}_{k+1} \rightarrow v_{k+1})$ for some $\widehat{y}$ .The Value Choice tactic suggests trying to set an item value to 0 (the least it can have), meaning the item is completely replaced (from a utility perspective) by more instances of another item.
Removing the kth item and adding some proportion h of its v_k instances to $\widehat{z}_{k+1}$ (or any $\widehat{z}_j$ - we use $\widehat{z}_{k+1}$to simplify the presentation) is described by the partial solution $\widehat{z}' = \widehat{y} \oplus (z_k \rightarrow 0) \oplus (z_{k+1} \rightarrow v_{k+1} + h.v_k)$.

Under what conditions does $\widehat{z}'$dominate $\widehat{z}$?

Applying the tactic

Suppose we have a partial solution $\widehat{z} = \widehat{y} \oplus (\widehat{z}_k \rightarrow v_k) \oplus (\widehat{z}_{k+1} \rightarrow v_{k+1})$ for some $\widehat{y}$ .The Value Choice tactic suggests trying to set an item value to 0 (the least it can have), meaning the item is completely replaced (from a utility perspective) by more instances of another item.
Removing the kth item and adding some proportion h of its v_k instances to $\widehat{z}_{k+1}$ (or any $\widehat{z}_j$ - we use $\widehat{z}_{k+1}$to simplify the presentation) is described by the partial solution $\widehat{z}' = \widehat{y} \oplus (z_k \rightarrow 0) \oplus (z_{k+1} \rightarrow v_{k+1} + h.v_k)$.

Under what conditions does $\widehat{z}'$dominate $\widehat{z}$?

Value Choice Tactic (UKP)

Applying the above tactic and calculating we can derive the following semi-congruence condition:

$$\widehat{z} \rightsquigarrow \widehat{z}' = h \leq w_k / w_{k+1}$$

The weak dominance relation $\widehat{\delta}$ is

$$\widehat{z}\widehat{\delta}\widehat{z}' = h \geq p_k / p_{k+1}$$

One way of combining the two is to let $h = \lfloor w_k / w_{k+1} \rfloor$ (the largest integral value it can take) which gives the dominance relation of Martello and Toth.

Value Choice Tactic (UKP)

Applying the above tactic and calculating we can derive the following semi-congruence condition:

$$\widehat{z} \rightsquigarrow \widehat{z}' = h \leq w_k / w_{k+1}$$

The weak dominance relation $\widehat{\delta}$ is

$$\widehat{z}\widehat{\delta}\widehat{z}' = h \geq p_k / p_{k+1}$$

One way of combining the two is to let $h = \lfloor w_k / w_{k+1} \rfloor$ (the largest integral value it can take) which gives the dominance relation of Martello and Toth.

Value Choice Tactic (UKP)

Applying the above tactic and calculating we can derive the following semi-congruence condition:

$$\widehat{z} \rightsquigarrow \widehat{z}' = h \leq w_k / w_{k+1}$$

The weak dominance relation $\widehat{\delta}$ is

$$\widehat{z}\widehat{\delta}\widehat{z}' = h \geq p_k / p_{k+1}$$

One way of combining the two is to let $h = \lfloor w_k / w_{k+1} \rfloor$ (the largest integral value it can take) which gives the dominance relation of Martello and Toth.

Value Choice Tactic (UKP)

But another way not considered by them is to let $h = \lceil p_k / p_{k+1} \rceil$which gives a different dominance condition

$$w_k \geq \lceil p_k / p_{k+1} \rceil w_{k+1}$$

In fact by applying the tactic in the other way, ie. assigning $\widehat{z}_k$ the maximum possible value ($\lfloor \frac{W}{w_k} \rfloor$), we were able to derive a previously unpublished dominance relation and greedy algorithm for the UKP.the former.

Value Choice Tactic (UKP)

But another way not considered by them is to let $h = \lceil p_k / p_{k+1} \rceil$which gives a different dominance condition

$$w_k \geq \lceil p_k / p_{k+1} \rceil w_{k+1}$$

In fact by applying the tactic in the other way, ie. assigning $\widehat{z}_k$ the maximum possible value ($\lfloor \frac{W}{w_k} \rfloor$), we were able to derive a previously unpublished dominance relation and greedy algorithm for the UKP.the former.

4. Example of Interchange Tactic

Try and derive a dominance relation by comparing a partial solution to a variant obtained by interchanging the values assigned to a pair of variables

Example

Scheduling $1//\sum C_i$ (from earlier)
Earlier, we assumed a dominance relation for this problem had already been derived. Now we show how it came about

Interchange Tactic (contd.)

The interchange tactic suggests the following: Suppose we extend some partial schedule by picking task s to assign to position a and then sometime later pick task t to assign in position b . When is this better than picking t for position a and s for position b? Let $z = \widehat{z} \oplus \{a \mapsto s\} \oplus \{b \mapsto t\} \oplus e$ and $z' = \widehat{z} \oplus \{a \mapsto t\} \oplus \{b \mapsto s\} \oplus e$. It is not difficult to derive the following semi-congruence condition is trivially true (assuming $s \neq t$). We can calculate the following weak dominance relation

$$\widehat{z}\widehat{\delta}\widehat{z}' = s.p \leq t.p$$

which we used earlier to get a greedy algorithm for this problem
Other Examples: $1//L_{max}$, Set Covering Problem

Weighted Matroids (Whitney 1935, Edmonds 1971)

- An algebraic structure $\langle S, I, w \rangle$ satisfying a set of axioms
- Has associated Greedy Algorithm that finds the subset in I with largest w
- Examples: Kruskal's MST algorithm, algorithm for solving $1/1/U_i$
- But not: Prim's MST algorithm or Huffman's min. length encoding algorithm

Weighted Matroids (Whitney 1935, Edmonds 1971)

- An algebraic structure $\langle S, I, w \rangle$ satisfying a set of axioms
- Has associated Greedy Algorithm that finds the subset in I with largest w
- Examples: Kruskal's MST algorithm, algorithm for solving $1/1/U_i$
- But not: Prim's MST algorithm or Huffman's min. length encoding algorithm

Weighted Matroids (Whitney 1935, Edmonds 1971)

- An algebraic structure $\langle S, I, w \rangle$ satisfying a set of axioms
- Has associated Greedy Algorithm that finds the subset in I with largest w
- Examples: Kruskal's MST algorithm, algorithm for solving $1/1/U_i$
- But not: Prim's MST algorithm or Huffman's min. length encoding algorithm

Weighted Matroids (Whitney 1935, Edmonds 1971)

- An algebraic structure $\langle S, I, w \rangle$ satisfying a set of axioms
- Has associated Greedy Algorithm that finds the subset in I with largest w
- Examples: Kruskal's MST algorithm, algorithm for solving $1/1/U_i$
- But not: Prim's MST algorithm or Huffman's min. length encoding algorithm

Specialization of DP (Bird and de Moor, 1993)

- Under certain conditions, a dynamic programming algorithm simplifies to a greedy algorithm
- Calculate the algorithm using their Relational Algebra and then test for the greediness condition
- Example: algorithm for solving $1//L_{max}$

Specialization of DP (Bird and de Moor, 1993)

- Under certain conditions, a dynamic programming algorithm simplifies to a greedy algorithm
- Calculate the algorithm using their Relational Algebra and then test for the greediness condition
- Example: algorithm for solving $1//L_{max}$

Specialization of DP (Bird and de Moor, 1993)

- Under certain conditions, a dynamic programming algorithm simplifies to a greedy algorithm
- Calculate the algorithm using their Relational Algebra and then test for the greediness condition
- Example: algorithm for solving $1//L_{max}$

Property Based Classification (Curtis, 2003)

- Top-level class (Best Global) with 3 subclasses (Better Global, Best Local, Better Local) that cover all greedy algorithms
- Each class has conditions. If you can establish the conditions the associated greedy algorithm is optimal
- Of course you need to establish the conditions..

Property Based Classification (Curtis, 2003)

- Top-level class (Best Global) with 3 subclasses (Better Global, Best Local, Better Local) that cover all greedy algorithms
- Each class has conditions. If you can establish the conditions the associated greedy algorithm is optimal
- Of course you need to establish the conditions..

Property Based Classification (Curtis, 2003)

- Top-level class (Best Global) with 3 subclasses (Better Global, Best Local, Better Local) that cover all greedy algorithms
- Each class has conditions. If you can establish the conditions the associated greedy algorithm is optimal
- Of course you need to establish the conditions..

Example: 1 machine scheduling

$1//\sum C_i$

$$\begin{aligned} &D_v \mapsto Task \\ &Task = id : Id \times p : Time \\ &o \mapsto \lambda(x,z).\ CSOT.o \wedge bijective(z.m) \\ &c \mapsto \lambda(x,z).\ \textstyle\sum_{i=1}^{n} ct(z,i) \text{ where } ct(z,i) = \sum_{j=1}^{i} z_j.p \end{aligned}$$

Example: 1 machine scheduling (contd.)

Given a partial schedule $\widehat{z} = \widehat{y} \oplus \{a \mapsto s\} \oplus \{b \mapsto t\}$ and an interchange permutation of it, $\widehat{z}' = \widehat{y} \oplus \{a \mapsto t\} \oplus \{b \mapsto s\}$, we show later how to derive the following dominance relation:
$\delta(x, \widehat{z}, \widehat{z}') = s.p \leq t.p$.
It can be shown the least $\widehat{z}$ (when a is the next variable) in the dominance ordering is the one in which $\widehat{z}_a$is the least.
By the Claim above, picking such a $\widehat{z}$ is the correct greedy choice.
This corresponds to the Shortest Processing Time first (SPT) rule, discovered by W.E. Smith in 1956.

1. Tactic for constructing a filter (Φ)

If a conjunct from the output condition , o matches the form

$$f_1(z_1) \otimes f_1(z_2) \otimes \cdots \otimes f_1(z_n) \preceq K$$

then a possible Φ is of the form

$$f_1(\widehat{z}_1) \otimes f_1(\widehat{z}_2) \otimes \cdots \langle \textit{best possible} \rangle \preceq K$$

where $\langle \textit{best possible} \rangle$means the least value assignments for the remaining variables, K is some constant, and f and $\otimes$ are found by the user.

Section Related Work

Related Work: Constraint Satisfaction

- Looks at generic properties of the constraints (e.g. path consistency, directionality, tree structure, etc.) and solution methods (backtracking, backjumping).
- We look at how problem-specific knowledge can be integrated into the generic model to *synthesize* a *custom* constraint satisfaction solver
- Complementary to our work

Related Work (Other)

- <1->Model Driven Architecture (MDA) from the OMG proposes to transform platform independent models to platform specific models
    - starting point is a high level design, not a specification. Behavior has been difficult
- <1->Pummel (Cook) polytypic definitions and partial evaluation of models written in a DSL using information derived from the metamodel
    - starting point is a model, behavior is inferred, based on the metamodel
- <1->AHEAD (Batory) is synthesis-in-the-large by composition of features
    - starting point is a collection of features, some of which might be algorithms
- <1->Z/B method (Abrial)
    - uses stepwise refinement (as opposed to calculation) to derive *entire* program

With exception of Z/B, the other approaches focus on systems or application generation

Planning

- Srivastava and Kambhapati (1998) used KIDS to synthesize fast custom planners
- Used filters, finite differencing, and context dependent simplification
- We would like to add optimality to this, and look for dominance relations, as well as tactics that could generalize some of their domain specific rules (e.g. across Blocks World and Tyre World)

Planning

- Srivastava and Kambhapati (1998) used KIDS to synthesize fast custom planners
- Used filters, finite differencing, and context dependent simplification
- We would like to add optimality to this, and look for dominance relations, as well as tactics that could generalize some of their domain specific rules (e.g. across Blocks World and Tyre World)

Planning

- Srivastava and Kambhapati (1998) used KIDS to synthesize fast custom planners
- Used filters, finite differencing, and context dependent simplification
- We would like to add optimality to this, and look for dominance relations, as well as tactics that could generalize some of their domain specific rules (e.g. across Blocks World and Tyre World)

Mapping Platform Independent Models to Platform Specific Models

- The PIM contains components that have specific computational, memory, and resource requirements.
- The platform "devices" have limitations such as processing speed, available memory, and communication bandwith limits
- The task becomes one of determining a good mapping of PIM to platform that satisfies the PIM requirements while respecting PSM limitations
- Related problem is real-time task scheduling
- Both kinds of problems have been formulated as global search (Wang, Merrick and Shin, 2004; Mutka and Li, 1995)
- Can we do better using synthesis?

Mapping Platform Independent Models to Platform Specific Models

- The PIM contains components that have specific computational, memory, and resource requirements.
- The platform "devices" have limitations such as processing speed, available memory, and communication bandwith limits
- The task becomes one of determining a good mapping of PIM to platform that satisfies the PIM requirements while respecting PSM limitations
- Related problem is real-time task scheduling
- Both kinds of problems have been formulated as global search (Wang, Merrick and Shin, 2004; Mutka and Li, 1995)
- Can we do better using synthesis?

Mapping Platform Independent Models to Platform Specific Models

- The PIM contains components that have specific computational, memory, and resource requirements.
- The platform “devices” have limitations such as processing speed, available memory, and communication bandwith limits
- The task becomes one of determining a good mapping of PIM to platform that satisfies the PIM requirements while respecting PSM limitations
- Related problem is real-time task scheduling
- Both kinds of problems have been formulated as global search (Wang, Merrick and Shin, 2004; Mutka and Li, 1995)
- Can we do better using synthesis?

Mapping Platform Independent Models to Platform Specific Models

- The PIM contains components that have specific computational, memory, and resource requirements.
- The platform “devices” have limitations such as processing speed, available memory, and communication bandwith limits
- The task becomes one of determining a good mapping of PIM to platform that satisfies the PIM requirements while respecting PSM limitations
- Related problem is real-time task scheduling
- Both kinds of problems have been formulated as global search (Wang, Merrick and Shin, 2004; Mutka and Li, 1995)
- Can we do better using synthesis?

Mapping Platform Independent Models to Platform Specific Models

- The PIM contains components that have specific computational, memory, and resource requirements.
- The platform “devices” have limitations such as processing speed, available memory, and communication bandwith limits
- The task becomes one of determining a good mapping of PIM to platform that satisfies the PIM requirements while respecting PSM limitations
- Related problem is real-time task scheduling
- Both kinds of problems have been formulated as global search (Wang, Merrick and Shin, 2004; Mutka and Li, 1995)
- Can we do better using synthesis?

Mapping Platform Independent Models to Platform Specific Models

- The PIM contains components that have specific computational, memory, and resource requirements.
- The platform “devices” have limitations such as processing speed, available memory, and communication bandwith limits
- The task becomes one of determining a good mapping of PIM to platform that satisfies the PIM requirements while respecting PSM limitations
- Related problem is real-time task scheduling
- Both kinds of problems have been formulated as global search (Wang, Merrick and Shin, 2004; Mutka and Li, 1995)
- Can we do better using synthesis?