Synthesis of Fast Programs for Maximum Segment Sum Problems

Srinivas Nedunuri & William R. Cook
University of Texas at Austin

October 5, 2009

Motivation

- Given
  - Behavioral specifications
  - Pre/Post condition
- Synthesize
  - Efficient algorithms
- Primary Tools
  - Algorithm Theories
    - Global Search
    - Local Search
    - Divide and Conquer
  - "Calculation" (derivation) of program components
- Global Search $\rightarrow$ Constraint Satisfaction

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

General versus Specific Constraint Solvers

- *Not* a generic constraint solver
- *Instead...*
- Synthesize algorithm for specific constraint-based problem

Example problem

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), $o : D \times R \rightarrow Boolean$
- A benefit criterion, $profit : D \times R \rightarrow C$

Maximum Independent Segment Sum (MISS)

Instantiation for MISS

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

Example

1	2	3	4	5	6	1.. x.maxVars
3	9	-2	-10	0	1	x.data
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 ($\widehat{z}$)

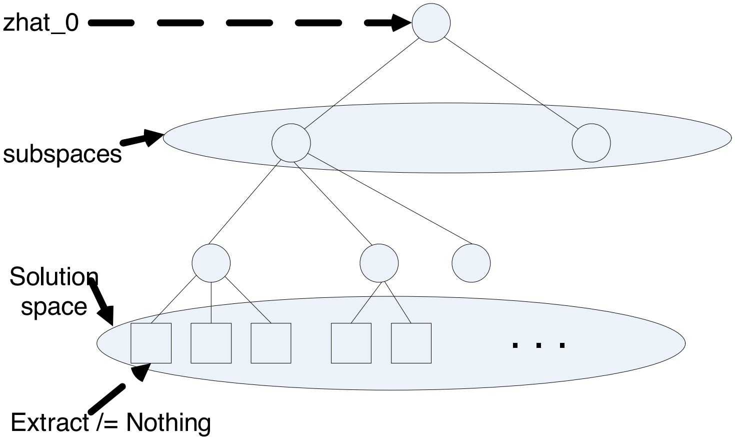
An assignment to some of the variables. Can be extended into a (complete) solution by assigning to all the variables.

Feasible Solution (z)

A solution which satisfies the output condition

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

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	$D \times R \rightarrow$ *Boolean*	Necessary condition for a space to contain feasible solutions
$\psi(\xi)$	Necessary (Consistent) Tightener	$D \times R \rightarrow R$	Tightens a given space to eliminate infeasible solutions. Preserves *all* (*at least one*) feasible solutions
ub(ib)	Upper (Initial) Bound	$D \times R \rightarrow C$	returns a upper(inital) bound on the profit of the best solution in the given space

These are usually derived from their specification by the application of domain knowledge

Global Search Optimization: generic algorithm in Haskell

```
fo :: D -> {R}
fo(x) =
    if phi(x, r0(x)) ∧ lb(x, r0(x)) ≤ ib(x)
    then f_gso(x, {r0(x)}, {})
    else {}

f_gso :: D x {R} x {R} -> {R}
f_gso(x, active, soln) =
    if empty(active)
    then soln
    else let
           (r, rest) = arbsplit(active)
           soln' = opt(profit, soln ∪{z | extract(z, r) ∧ o(x,z)})
           ok_subs = {propagate(x, s) :
                                    s ∈ subspaces(r)
                                    ∧ propagate(x, s) ≠ Nothing}
           subs' = {s : s ∈ ok_subs
                           ∧ ub(x, s) ≥ lb(x, soln')}
         in f_gso(x, rest ∪ subs, soln')
```

Global Search Optimization (cont.)

```
ub :: D x {R} -> C
ub(x, solns) =
    if empty(solns) then ib(x) else profit(x, arb(solns))

propagate x r =
    if phi(x, r) then (iterateToFixedPoint psi x r)  else Nothing

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)

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

$\oplus$ 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, ub are given default definitions (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 it has exponential complexity! (The search space grows exponentially). Even with good definitions for the search control operator it still grows exponentially
- So we incorporate a concept that has been used in operations research for several decades: *dominance relations*

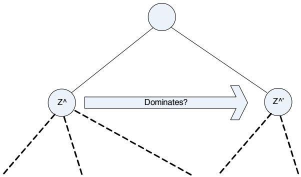
Are we done?

- With this instantiation, the abstract program is correctly instantiated into a working solver. But it has exponential complexity! (The search space grows exponentially). Even with good definitions for the search control operator it still 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?

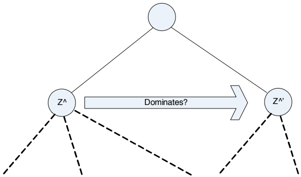
- Enables the comparison of one partial solution with another to determine if one of them can be discarded
- Given $\widehat{z}$ and $\widehat{z}'$ if the best possible solution in $\widehat{z}$ is better than the best possible solution in $\widehat{z}'$ then $\widehat{z}'$ can be discarded



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

One way to derive dominance is to focus on a restricted case: dominance relative to equivalent extensions.

- Let $\widehat{z} \oplus e$ denote combining a partial solution $\widehat{z}$ with an *extension* e.
- When $\widehat{z} \oplus e$ is a (feasible) complete solution, e is called the *(feasible) completion* of $\widehat{z}$.

A special case of dominance arises when all feasible completions of a space are also feasible completions for another space, and the first solution is always better than the second solution.

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 at least as beneficial as the same feasible completion of $\widehat{z}'$

Definition: Weak Dominance

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

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

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 at least as beneficial as the same feasible completion of $\widehat{z}'$

Definition: Weak Dominance

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

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

Definitions

Definition: Semi-Congruence

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

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

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 at least as beneficial as the same feasible completion of $\widehat{z}'$

Definition: Weak Dominance

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

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

Dominance Relations (contd.)

To get a dominance test, combine the two

Theorem (Dominance)

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

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 $\widehat{z}$ at least as beneficial as the best solution in $\widehat{z}'$

When $\mathit{profit}^*(\widehat{z}) \geq \mathit{profit}^*(\widehat{z}')$ 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 (Profit Distribution)

If *profit* distributes over $\oplus$ and $\mathit{profit}(\widehat{z}) \geq \mathit{profit}(\widehat{z}')$ then $\widehat{z}\ \widehat{\delta}\widehat{z}'$

Dominance Relations (contd.)

To get a dominance test, combine the two

Theorem (Dominance)

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

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 $\widehat{z}$ at least as beneficial as the best solution in $\widehat{z}'$

When $\mathit{profit}^*(\widehat{z}) \geq \mathit{profit}^*(\widehat{z}')$ 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 (Profit Distribution)

If *profit* distributes over $\oplus$ and $\mathit{profit}(\widehat{z}) \geq \mathit{profit}(\widehat{z}')$ then $\widehat{z}\ \widehat{\delta}\widehat{z}'$

..Back to MISS

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

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

Dominance Relation for MISS

Since *profit* is a distributive profit function, the definition for δ follows immediately: $\widehat{z} \rightsquigarrow \widehat{z}' \wedge \mathit{profit}(\widehat{z}) \geq \mathit{profit}(\widehat{z}')$
This dominance test reduces the complexity of the MISS algorithm from exponential to polynomial. This is good but we can do better.

A Necessary Tightener for MISS

Apply a "Neighborhood" tactic to calculate a tightener for a space: If a segment is selected, then the next segment must *not* be selected.

An upper bound

- An upper bound on a partial solution is the value of the best possible solution obtainable from that partial solution
- Combine the profit of the partial solution with the best possible profit obtainable from the remaining variables

$$upperBound(x, \widehat{z}) = p(x, \widehat{z}) + \sum_{i=\#\widehat{z}}^{\#x.sqnce} \max(x.sqnce(i), 0)$$

What is the cumulative effect of all the operators?

For input $x = [1 \ldots 10]$:

Operator Added	# of Nodes in Search Tree
None	2047
+ dominates	486
+tighten	12
+upperBound	12

- Dominance and Tightening are very significant in eliminating large swathes of the search space
- But the algorithm is still not linear time..

Finite Differencing (Page & Koenig, 1982)

Incrementally update an expensive computation rather than computing it each time in the loop.

Requires introducing accumulating arguments into the main search loop.

Tedious, but not difficult.

Final Algorithm

Theorem

Algorithm MISS runs in linear time

Following table shows the results of running on sequences of randomly generated numbers of varying length

Input Length	NC (s)	Sasano (s)
1000	0.00	0.00
10,000	0.12	0.14
20,000	0.22	0.28
40,000	0.43	0.72
80,000	0.75	1.8
100,000	1.1	2.8
200,000	2.2	8.9
400,000	4.6	stack overflow

Related Segment Sum Problems

- Using the same approach, and with several small changes to the derivation, we have synthesized efficient linear-time algorithms for variations on the problem, specifically Maximum Multi-Marking and Maximum Alternating Segment Sum (see the paper for the details)
- In all cases we outperform the code produced by Sasano et al. using program transformation

Summary & Conclusions

- We have shown how the addition of dominance relations can significantly improve the complexity of an algorithm
- We have applied the ideas of program synthesis to some useful and well-known problems
- Program synthesis is an effective way of generating effective and efficient code
- The methodology we have applied can be used to generate algorithms for a family of related programs, with sharing of derivations. In contrast, program transformation requjires a completely new transformation for each variation