Abstraction via Approximate Symmetry

Thomas Ellman
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903
ellman@cs.rutgers.edu

Abstract

Abstraction techniques are important for solving constraint satisfaction problems with global constraints and low solution density. In the presence of global constraints, backtracking search is unable to prune partial solutions. It therefore operates like pure generate-and-test. Abstraction improves on generate-and-test by enabling entire subsets of the solution space to be pruned early in a backtracking search process. This paper describes how abstraction spaces can be characterized in terms of approximate symmetries of the original, concrete search space. It defines two special types of approximate symmetry, called "range symmetry" and "domain symmetry", which apply to function finding problems. It also presents algorithms for automatically synthesizing hierarchic problem solvers based on range or domain symmetry. The algorithms operate by analyzing declarative descriptions of classes of constraint satisfaction problems. Both algorithms have been fully implemented. This paper concludes by presenting data from experiments testing the two synthesis algorithms and the resulting problem solvers on NP-hard scheduling and partitioning problems.

1. Introduction

Abstraction techniques are important for solving constraint satisfaction problems (CSPs) with global constraints and low solution density. Examples of such problems include the Partition problem and the Multiprocessor Scheduling problem, both of which are NP-hard [Garey and Johnson, 1979]. In the presence of global constraints, backtracking search is unable to prune partial solutions [Nadel, 1988]. It therefore operates like pure generate-and-test. When overall solution density is low, this approach is not effective, except when applied to small problems. Abstraction can improve the performance of backtracking by enabling entire subsets of the solution space to be pruned early in the search process.

The value of abstraction can be illustrated by considering the Multiprocessor Scheduling problem. (See Figure 2.) A solution to this problem is an assignment of beginning times to jobs that meets deadlines, obeys precedence constraints and has a limited number of jobs running at once. In this problem, the constraint on the number of simultaneously running jobs is global, i.e., in general, one cannot verify the constraint without knowing the beginning times of all the jobs. Backtracking algorithms therefore cannot easily use this constraint to prune partial solutions. In contrast, the following abstraction strategy introduces a new level of description that enables partial solutions to be pruned: First partition the set of possible beginning times into contiguous, disjoint time windows. Then construct an abstract solution in which each job is assigned a beginning window, rather than a beginning time. If possible, prune any window assignments that can be shown to violate the problem constraints. Finally, refine the window assignments into specific beginning times.

As a second illustration of the value of abstraction, consider the Partition problem. (See Figure 3.) A solution to this problem is a subset M of a set E such that the weight of elements included in M equals the weight of elements excluded from M. This constraint is global, since one cannot verify equality of the total weights without knowing the inclusion/exclusion status of each element. Backtracking algorithms therefore cannot easily use this constraint to prune partial solutions. The following abstraction strategy overcomes this limitation: First divide the elements of E into disjoint classes such that elements of roughly equal weight are grouped together. Then select a quota for each class that represents the number of elements of the class to be included in M. If possible, prune any quota assignments that can be shown to violate the problem constraints. Finally, refine the quotas into an actual selection of elements to be included in the set M.

The behavior of a hierarchic problem solver is illustrated in Figure 1. The illustration shows a search tree broken into two parts. The upper portion represents a space of "abstract states", (e.g., a space of window or quota assignments). Backtracking tests each complete abstract solution against an "abstract goal" (e.g., testing problem constraints against window or quota assignments). The lower portion represents a space of "concrete states", (e.g., a space of beginning times or subset membership assignments). Backtracking tests each complete concrete solution against the original "concrete goal", (e.g., testing problem constraints against beginning times or subset membership assignments). Search in the concrete space is also guided by "constrained variable range generators". These require each concrete state variable to assume values that are consistent with the abstract solution, (e.g., obeying previously specified window or quota assignments). Whenever the problem goal includes local constraints, the search may also be guided by "localized" abstract or concrete goals which serve to prune partial abstract or concrete solutions.

Algorithms for synthesizing hierarchic problem solvers are presented in Section 3. These synthesis algorithms can be defined in terms of the components of the hierarchic problem solver illustrated in Figure 1. Each algorithm inputs a description of a concrete search space and a concrete goal. Each outputs an abstract search space, an abstract goal and a set of constrained variable range generators. (Methods of constructing local goals and local abstract goals are described in [Braudaway and Tong, 1989].)

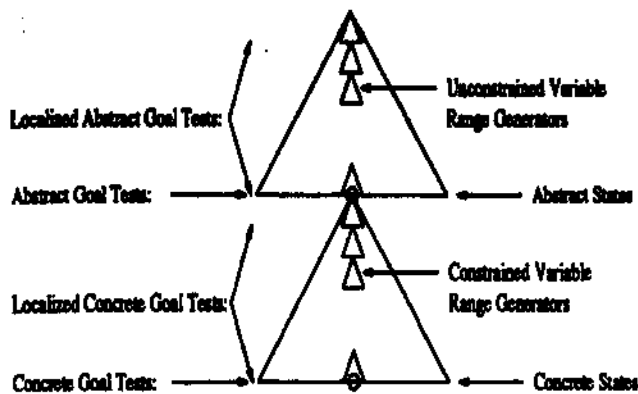

Figure 1: Hierarchic Problem Solver

2. Symmetry and Abstraction

An "abstraction space" A may be defined formally as a partition of the states in the underlying "concrete space" S. Thus each abstract state $a \epsilon A$ is a set of concrete states $s \epsilon S$. Abstract search spaces may also be characterised in terms of permutation groups whose elements operate on states in the concrete search space. In particular, given a group Σ of permutations that operate on states $s \epsilon S$, one may define an abstraction space A as follows: For all pairs of states s_1 and s_2 drawn from S, s_1 and s_2 lie in a common abstract state a if and only if there exists a $\sigma \epsilon \Sigma$, such that $\sigma(s_1) = s_2$. (The group axioms guarantee that this definition yields a partition of S.) On the other hand, given an abstraction space A, one can construct a group Σ of permutations that will induce A according to the definition given above. For example, one may take Σ to be the direct product of the symmetric subgroups $\mathcal{A}(a)$ for all $a \epsilon A$. (The symmetric group $\mathcal{A}(X)$ is just the set of all permutations of the set X.) Notice that this last construction is not unique, since several permutation groups may induce the same partition. As a result of the connection between permutation groups and abstraction spaces, one can characterise various abstraction strategies in terms of the structure of permutation groups, as described below.

Exact Symmetry: Consider an arbitrary group Σ of permutations of the states in the concrete space S. A goal function $G : S \rightarrow \{True, False\}$ is said to be *exactly symmetric* with respect to Σ if for each $s \epsilon S$, and each $\sigma \epsilon \Sigma$, $G(\sigma(s)) = G(s)$. If the group Σ is used to induce a partition of S into an abstraction space A, the space A will have the following useful property. For each abstract state a, all concrete states $s \epsilon a$ have the same value of the goal function. Thus if any single element of a fails to satisfy the goal function, then all elements of a will fail. The abstract state a may then be safely pruned without risk of losing completeness of the problem solver. Furthermore if any single element of a succeeds in satisfying the goal function, then all elements of a will succeed. All members of the abstract state a may then be immediately identified as solutions.

Approximate Symmetry: Consider once again an arbitrary group Σ of permutations of the states in the concrete space S. A goal function $G : S \rightarrow \{True, False\}$ is said to be *approximately symmetric* with respect to Σ if there exists a good approximation $\tilde{G}$ of G such that $\tilde{G}$ is exactly symmetric with respect to Σ. Once again, if the group Σ is used to induce a partition of S into an abstraction space A, the space A will have useful properties. Two interesting cases result from considering the ways in which G and $\tilde{G}$ are related:

- Necessary Approximations: $G \Rightarrow \tilde{G}$, i.e., G entails $\tilde{G}$: In this case, given an abstract state a, if any single element of a fails to satisfy the approximate goal $\tilde{G}$, all elements of a will fail to satisfy $\tilde{G}$, and therefore all elements of a will fail to satisfy the original goal G. The abstract state a may then be safely pruned.
- Sufficient Approximations: $\tilde{G} \Rightarrow G$, i.e., $\tilde{G}$ entails G: In this case, given an abstract state a, if any single element of a succeeds in satisfying the approximate goal $\tilde{G}$, all elements of a will satisfy $\tilde{G}$, and therefore all elements of a will satisfy the original goal G. All elements of the abstract state a will then be valid solutions.

Necessary approximations are generally more important in practice. In particular, necessary approximations are useful for finding solutions when solution density is low (the usual case), while sufficient approximations are useful for finding non-solutions when solution density is high (not the usual case). The distinction between necessary and sufficient approximations is quite similar to the difference between "TI abstraction" an "TD abstraction" defined in [Giunchiglia and Walsh, 1992], and the distinction between the "Upward Solution Property" and the "Downward Solution Property" described in [Knoblock *et al.*, 1991].

Types of Approximate Symmetry: Interesting special types of approximate symmetry result from considering CSPs whose solution requires finding a function $f : D \rightarrow R$ to meet specified constraints. Example problems of this type include the Multiprocessor Scheduling problem, which requires finding a function mapping jobs to beginning times, and the Partition problem, which requires finding a boolean valued function of a set of elements. For problems of this type, the concrete search space S is the set of all functions from D into R. Abstractions of this space can be defined in the following three steps: (1) Select some group Φ of permutations of the domain D or the range R; (2) Use the group Φ to construct a group Σ of permutations of the search space S such that Φ is a homomorphic image of Σ; (3) Use Σ to induce a partition A of S. Various types of abstractions result depending on the structure of the original group Φ, on whether members of Φ operate on the domain D or the range R and on the details of constructing Σ from Φ.

Range Symmetry: Suppose that Φ is a group of permutations of the range R of the function f. Define the group Σ in the following steps: (1) For each permutation ϕ of Φ, and each element d of D define $\sigma_{\phi d}$ such that $\sigma_{\phi d}(f)(d) = \phi(f(d))$ and for all $d' \epsilon D$, if $d \neq d'$ then $\sigma_{\phi d}(f)(d') = f(d')$. (2) For each $d \epsilon D$, let Σ_d be the set of all $\sigma_{\phi d}$ such that $\phi \epsilon \Phi$. (3) Let Σ be the direct product of all Σ_d for $d \epsilon D$. Using this construction, two functions f_1 and f_2 are equivalent with respect to Σ whenever f_1 and f_2 differ only by returning values that are equivalent with respect to Φ. In this case, Σ induces an abstraction A of the original space S that can be represented in the following way: Each element $a \epsilon A$ is a function $\hat{f}$ that accepts an element d in D and returns a set $\hat{r}$ appearing in the partition $\hat{R}$ induced by Φ.

Consider an application of range symmetry to the Multiprocessor Scheduling problem. (See Figure 2.) In

this example, the role of f is played by an integer valued function b that defines the beginning times of jobs. The role of D is taken by the set J of jobs. The role of R is taken by the set T of possible beginning times. The group Φ could be selected to induce a partition $\hat{T}$ of T such that the sets $\hat{t}\epsilon\hat{T}$ are disjoint sequences $[l \ldots u]$ of beginning times. Each element a in the abstraction space A will represent a function $\tilde{b}$ that assigns to each job j a range $\hat{t}$ of possible beginning times. The range $\hat{t}$ constrains the beginning time of job d to lie in a time window, but does not specify the precise beginning time of any job. Range symmetry thus captures the abstraction strategy of the Multiprocessor Scheduling example described above.

Domain Symmetry: Now suppose that Φ is a group of permutations of the domain D of the function f. Suppose further that Φ is a direct product of symmetric subgroups $A(\hat{d})$ for all $\hat{d}$ in some partition $\hat{D}$ of D. Define the group Σ by taking each element ϕ of Φ and forming a corresponding element σ_ϕ of Σ such that for each $f\epsilon S$ and each $d\epsilon D$, $\sigma_\phi(f)(d) = f(\phi(d))$. Thus two functions f_1 and f_2 are equivalent with respect to Σ whenever f_1 and f_2 differ by permuting their values at points in D that are equivalent with respect to Φ. In this case, Σ induces an abstraction A of original space S that can be represented in the following way: Each element $a\epsilon A$ is a function $\tilde{f}$ that accepts a subset $\hat{d}$ in the partition $\hat{D}$ and returns a multiset of cardinality $|\hat{d}|$ of values drawn from the range $\hat{R}$.

Consider an application of domain symmetry to the Partition problem. (See Figure 3.) In this example, the role of f is played by a boolean function m, which specifies the elements included in the set M. The role of D is taken by the set E of elements to be partitioned. The role of R is taken by the set $\{True, False\}$ of boolean values. The group Φ could be selected to induce a partition $\hat{E}$ of E such that each set $\hat{e}\epsilon\hat{E}$ contains elements of nearly equal weight. Each element a in the abstraction space A will represent a function $\tilde{m}$ that assigns to each set $\hat{e}$ a multiset of boolean values. The number of $True$ values in the multiset will represent the number of elements of $\hat{e}$ that are included in M. Likewise, the number of $False$ values in the multiset will represent the number of elements of $\hat{e}$ that are not included in M. An abstract solution a therefore implicitly assigns a quota to each set $\hat{e}$, but does not precisely specify the inclusion status of any particular element. Domain symmetry thus captures the abstraction strategy of the Partition example described above.

3. Synthesis of Problem Solvers

A general strategy for synthesising hierarchic problem solvers can be formulated in terms of approximate symmetry: (1) Select a group Σ of permutations of the search space S. (2) Transform the original goal G into an approximate goal $\tilde{G}$ such that $G \Rightarrow \tilde{G}$ and $\tilde{G}$ is exactly symmetric with respect to Σ. (3) Use Σ and $\tilde{G}$ to construct components of the hierarchic problem solver shown in Figure 1. Algorithms that instantiate this generic abstraction strategy for the special cases of domain symmetry and range symmetry are described below.

Parameterised CSPs: Classes of constraint satisfaction problems are represented using a notation called "parameterised constraint satisfaction problems" (PCSPs). A PCSP includes a *signature* S and a *goal-function* G. The signature is a declaration of the types of sets and functions that appear in problem instances, including: A list of names and data types (symbol or integer) of finite sets; A list of names of functions along with a declared domain and range for each; A list of names and data types of constants. Furthermore, the signature also declares each function to be either "known" or "unknown". The known functions are supplied as part of a problem instance specification. The unknown functions must be found in order to solve the problem. We use the notation $f(e,p)$ to indicate that the known function f depends on the problem instance p, as well as its argument e. We use the notation $f(e,s)$ to indicate that the unknown function f depends on the solution s, as well as its argument e. The problem instance p or state s parameter will occasionally be omitted from $f(e,p)$ or $f(e,s)$, when its role is clear from the context.

The goal function G serves to determine how the unknown functions may be specified in order to solve the problem. G is conceptually a map from the known and unknown functions into the booleans. We use the notation $G(p,s)$ to indicate that G depends on the problem instance p (i.e., the known functions) and the solution s, (i.e., the unknown functions). A particular goal function G is represented by a boolean formula that may reference the known and unknown functions; primitive arithmetic $(+, -, *, /)$, relational $(<, \leq, >, \geq)$ and boolean $(\neg, \vee, \wedge)$ operations; as well as conditionals $If(c,x,y)$ and absolute values $Abs(x)$. G may also include universal $\forall$ or existential $\exists$ quantification and sums Σ or products Π of functions over sets declared in the signature. Signatures and goal functions for the Partition and Multiprocessor Scheduling problems are found Figures 2 and 3.

- Given: A set J of jobs; a set T of beginning times; a working time $w(j)$ for each job j; a deadline $d(j)$ for each job j; a precedence relation $p(j,k)$ on the jobs; a number n of processors. Find: An assignment of beginning times to jobs that meets the deadlines, obeys the precedence relation and has at most n jobs running at once.
- Signature:
  - Sets: Jobs J (Symbol); Times T (Integer).
  - Known:
    Working-Time $w : J \rightarrow T$
    Deadline $d : J \rightarrow T$
    Precedence $p : J \times J \rightarrow \{True, False\}$
    Number of Processors n of type Integer.
  - Unknown: Beginning-Time $b : J \rightarrow T$
- Goal Function:

$$
\begin{aligned}
G(p,s) &= g_1(p,s) \wedge g_2(p,s) \wedge g_3(p,s) \\
g_1(p,s) &= (\forall j \epsilon J(p)) \quad e(j,p,s) \leq d(j,p) \\
g_2(p,s) &= (\forall j, j' \epsilon J(p)) \neg p(j,j',p) \vee q(j,j',p,s) \\
g_3(p,s) &= \neg \quad (\exists j \epsilon J(p)) \quad l(j,p,s) > n(p) \\
q(j,j',p,s) &= b(j',s) \geq e(j,p,s) \\
l(j,p,s) &= \textstyle\sum_{k \epsilon J(p)} if(r(k,b(j),p,s),1,0) \\
r(j,t,p,s) &= \{b(j,s) \leq t\} \wedge \{e(j,p,s) > t\} \\
e(j,p,s) &= b(j,s) + w(j,p)
\end{aligned}
$$

Figure 2: Multiprocessor Scheduling Problem Class

- **Given: A set E and a weight $w(e)$ for each element e of E. Find: A subset M of E such that the total weight of M equals the total weight of $E - M$.**
- **Signature:**
  - **Sets: Elements E (Symbol); Weights W (Integer).**
  - **Known: Weight $w : E \rightarrow W$**
  - **Unknown: Membership $m : E \rightarrow \{True, False\}$**
- **Goal Function:**

$$
\begin{aligned}
G(p,s) &= s_1(p,s) = s_2(p,s) \\
s_1(p,s) &= \sum_{e \epsilon E(p)} if(m(e,s), w(e,p), 0) \\
s_2(p,s) &= \sum_{e \epsilon E(p)} if(m(e,s), 0, w(e,p))
\end{aligned}
$$

Figure 3: Partition Problem Class

Synthesis Algorithms: Algorithms for synthesizing hierarchic problem solvers based on range and domain symmetry are shown in Figure 4. Each algorithm begins by partitioning the range R or the domain D of the unknown function / into a set R or D of equivalence classes. The partitioning is carried out by clustering R or D based on similarity of values of some known function whose domain is R or D respectively, and which is referenced by the goal function G(p, a). (These clustering techniques are described [Ellman, 1993].) Next, the original goal function G(p, *s)* is transformed into an abstract goal *G(p, a)* in order that it will operate on abstract states *a* that represent abstractions / of the unknown function /. In each algorithm this transformation is achieved, in part, by a process of replacing operations on objects with corresponding operations on sets, or corresponding symmetric operations. The revised goal is then surrounded with a test for the appearance of *True* in the returned set of boolean values. The resulting abstract goal *G(p,*a) is, by construction, a *necessary* condition on solvability of the abstract state a. Finally, a refinement function R_d is constructed for each element *D* of the domain *d.* The function *Rd(p,* a, *s)* takes the problem specification p, an abstract solution a, and a partially specified concrete solution s, and returns the set of possible values for /(d) that are consistent with *a* and *a.*

The abstract goal is synthesized at compile time, i.e., when the problem class is specified, but no problem instances are at hand. Once the abstract goal is constructed, the system applies a series of equivalence-preserving optimizing transformations, such as distributive laws and factorization of sums, products and quantifications, to improve the computational efficiency of the abstract goal. (The same optimizations are applied to the original, concrete goal and to localized versions of both abstract and concrete goals). All set arithmetic operations appearing in abstract goals (E.g., +, —, /, *) are implemented as operations on real intervals. Since these set operations take constant time, the abstract goal has the same asymptotic complexity as the original goal.

The domain *D* or range *R* is partitioned at run time, i.e., after a problem instance has been specified. Input taken from the user at compile time is used to determine which known function will guide partitioning of *D* or *R.* (In the Partition and Multiprocessor Scheduling examples, the user actually has only one choice.) Depending on the set *U* to be partitioned, and the known function f selected to guide the partitioning, one of two clustering algorithms is used: (1) If / is the identity function, and

Definitions:

- **Set Functions: Given an arbitrary function $h : A \rightarrow B$, define the corresponding set function $\hat{h} : A \cup 2^A \rightarrow 2^B$, such that $\hat{h}(x) = \{h(x)\}$, if $x \epsilon A$, and $\hat{h}(x) = \{h(y) | y \epsilon x\}$ if $x \subseteq A$.**
- **Symmetric Functions: Given an arbitrary function $h : A \rightarrow B$, and a partition $\hat{A}$ of A, define the corresponding symmetric function $\bar{h} : A \cup 2^A \rightarrow 2^B$ such that $\bar{h}(x) = \{h(y) | y \epsilon Class(x)\}$, if $x \epsilon A$, and $\bar{h}(x) = \cup_{y \epsilon x} \{h(z) | z \epsilon Class(y)\}$, if $x \subseteq A$.**

Algorithms: Given a search space S representing all functions f from D into R, and a goal function $G(p,s)$ defined on all $s \epsilon S$ and problem instances p.

1. **Define an abstraction space A:**
   - **Range Symmetry: Define a partition $\hat{R}$ of R by clustering R based on similarity of values of R itself or some known function defined on R that is referenced by the goal function $G(p,s)$. Let the abstraction space A represent all functions $\tilde{f} : D \rightarrow \hat{R}$.**
   - **Domain Symmetry: Define a partition $\hat{D}$ of D by clustering D based on similarity of values of D itself or some known function defined on D that is referenced by the goal function $G(p,s)$. Let the abstraction space A represent all functions $\tilde{f}$ with domain $\hat{D}$ such that for all $\hat{d} \epsilon \hat{D}$, $\tilde{f}(\hat{d})$ returns a multiset over R of cardinality $|\hat{d}|$.**
2. **Synthesize an abstract goal $\tilde{G}(p,a)$ defined on all $a \epsilon A$ and problem instances p:**
   - (a) **Construct an approximate goal function $\tilde{G}(p,s)$ that is exactly symmetric with respect to $\hat{R}$ or $\hat{D}$:**
     - i. **Transform the original goal function $G(p,s)$ into a boolean set valued function $\hat{G}(p,s)$:**
       - **Range Symmetry: Replace each reference to the unknown function $f(d)$ with the expression $Class(f(d))$. Replace each primitive or known function h with the corresponding set function $\hat{h}$.**
       - **Domain Symmetry: Replace each reference to a known function h defined on domain D, with the corresponding symmetric function $\bar{h}$. Replace each primitive or known function h defined on $D' \neq D$, with the corresponding set function $\hat{h}$.**
     - ii. **Let $\tilde{G}(p,s) = True \epsilon \hat{G}(p,s)$.**
   - (b) **Define the abstract goal $\tilde{G}(p,a)$ to select an arbitrary $s \epsilon a$ and return $\tilde{G}(p,s)$.**
3. **Synthesize functions $R_d(p,a,s)$ that incorporate constraints between the abstract space A and the concrete space S. For each $d \epsilon D$, $R_d(p,a,s)$ is the set of values of $f(d)$ that are consistent with the abstract solution a and the partially specified concrete solution s.**
   - **Range Symmetry: $R_d(p,a,s)$ is the set $\tilde{f}(d,a)$.**
   - **Domain Symmetry: $R_d(p,a,s)$ is the set of elements in the multiset difference between $\tilde{f}(\hat{d},a)$, and the multiset of all $f(d,s)$ $(d \epsilon \hat{d})$ such that $f(d)$ is assigned a value in state s.**

Figure 4: Synthesis Algorithms and Definitions

$$
\begin{aligned}
\ddot{G}(p,a) &= (\forall j, j' \epsilon J(p)) \;\; g_0(j,j',p,a) \\
g_0(j,j',p,a) &= g_1(j,p,a) \wedge g_2(j,j',p,a) \wedge g_3(j,p,a) \\
g_1(j,p,a) &= True \; \epsilon \; \{\hat{e}(j,p,a) \hat{\leq} \hat{d}(j,p)\} \\
g_2(j,j',p,a) &= \neg p(j,j',p) \vee True \; \epsilon \; \hat{q}(j,j',p,s) \\
g_3(j,p,a) &= False \; \epsilon \; \{\hat{l}(j,p,a) \hat{>} n(p)\} \\
\hat{q}(j,j',p,s) &= \ddot{b}(j',s) \hat{\geq} \hat{e}(j,p,s) \\
\hat{l}(j,p,a) &= \hat{\sum}_{k \epsilon J(p)} \hat{if}(\hat{r}(k,\bar{b}(j),p,a),\{1\},\{0\}) \\
\hat{r}(j,t,p,a) &= \{\ddot{b}(j,a) \hat{\leq} t\} \hat{\wedge} \{\hat{e}(j,p,a) \hat{>} t\} \\
\hat{e}(j,p,a) &= \ddot{b}(j,a) \hat{+} \hat{w}(j,p)
\end{aligned}
$$

Figure 5: Abstract Multiprocessor Scheduling Goal

$$
\begin{aligned}
\ddot{G}(p,a) &= True \; \epsilon \; (\hat{s}_1(p,a) \hat{=} \hat{s}_2(p,a)) \\
\hat{s}_1(p,a) &= \hat{\sum}_{d \epsilon \hat{E}(p)} \hat{\sum}_{(e,v) \epsilon (d, \hat{m}(d,a))} if(v, \bar{w}(e,p), \{0\}) \\
\hat{s}_1(p,a) &= \hat{\sum}_{d \epsilon \hat{E}(p)} \hat{\sum}_{(e,v) \epsilon (d, \hat{m}(d,a))} if(v, \{0\}, \bar{w}(e,p))
\end{aligned}
$$

Figure 6: Abstract Partition Goal

U is an unbroken sequence of integers (U = [/...u), then *U* is partitioned into intervals of equal or nearly equal size; (2) Otherwise, *U* is partitioned in a bottom up fashion, starting with singleton sets, and repeatedly merging two sets *x* and *y* such that / has minimal variation over *xUy.* In either case, the final partition is chosen to yield an abstract space whose size is the square root of the size of the concrete space, based on a rough analogy with Korf's result on the optimal size of abstraction spaces [Korf, 1987].

When the range symmetry algorithm is applied to the Multiprocessor Scheduling problem description in Figure 2, it generates the abstract goal function shown in Figure 5. This goal function *G(p,a)* performs interval arithmetic, interval comparisons, and boolean-set algebra on the time window assignments b(j, a) represented by an abstract state *a:* For example, *G(p, a)* checks deadlines by converting beginning time windows into ending time windows, and noticing whether any job's deadline occurs earlier than the start of its ending window. Likewise, G(p, *a)* checks for processor overload by computing lower bounds on the numbers of jobs running at each point in time, and noticing whether any lower bound exceeds the available number of processors. The abstract goal G(p, *a)* thereby computes a necessary condition on the original goal.

When the domain symmetry algorithm is applied to the Partition problem description in Figure 3, it generates the abstract goal function shown in Figure 6. This fjoal function G(p,a) selects an arbitrary concrete so- ution s consistent with the quota assignments m(e, a) represented by an abstract state a. It then applies to s an approximation G(p, *s)* of the original goal function that is symmetric with respect to the partition *E.* This symmetric goal function G(p, a) treats each *eeE* as if it has an interval *w(e)* of weights, rather than an actual weight *w(e),* i.e., an interval spanning the weights of all elements in *Class(e).* It first computes an interval i_1 bounding the total weight of elements included in *M* and an interval S_2 bounding the total weight of elements excluded from *M.* It then checks whether these intervals overlap. The abstract goal *G(p,a)* thereby computes a necessary condition on the original goal.

4. Experimental Results

A series of experiments was run to evaluate the performance of automatically synthesized problem solvers based on range and domain symmetry. Results of these experiments are shown in Figures 7 and 8. Each graph compares the performance of a hierarchic problem solver, which uses abstraction, to the performance of a flat problem solver, which uses no abstraction. The hierarchic and fiat problem solvers use the same underlying backtracking code, but with different input specifications. Each graph plots problem size against performance measured in terms of CPU time. CPU time is a better measure than alternatives, like numbers of goal tests or numbers of states generated, for two reasons: First, the computational cost of generating one state or testing one goal may not be the same in the flat and hierarchic problem solvers. Second, the relative importance of state generation and goal testing may not be the same in the flat and hierarchic problem solvers.

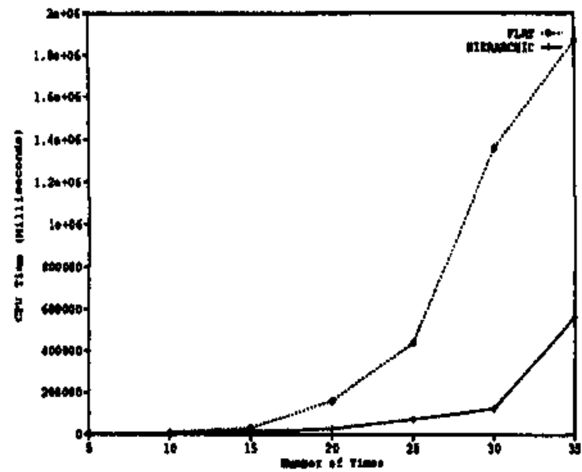

Figure 7: Multiprocessor Scheduling Problem Results: Hierarchic (Solid) v. Flat (Dotted)

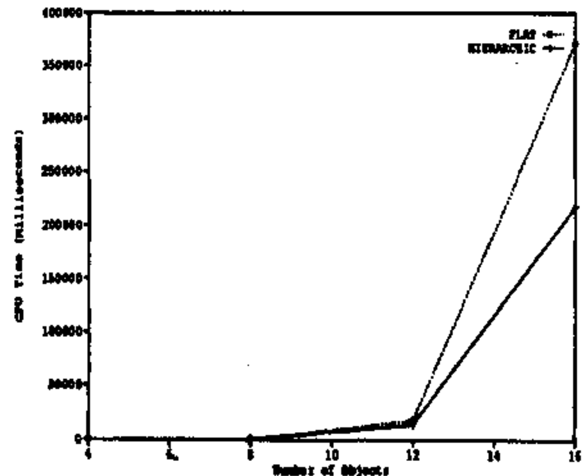

Figure 8: Partition Problem Results: Hierarchic (Solid) v. Flat (Dotted)

The Multiprocessor Scheduling; problem graph compares performance on problems with a fixed domain size (i.e., 6 jobs), and a varying range size (i.e., the number of time slots running from 5 to 35). Each data point represents an average over a set of 100 randomly generated problems. The multiprocessor problem search was guided by localized versions of the abstract and concrete goals in order to realize pruning from application of the local constraints that appear in this problem. The localized goals were constructed automatically from the corresponding unlocalized goals using standard dependency-tracing techniques. The Partition problem graph compares performance on problems with a varying domain size (i.e., the number of elements running from 4 to 16) and a fixed boolean range. Each data point represents an average over a set of 10 randomly generated problems. The Partition problem search was guided by unlocalized versions of the abstract and concrete goals, since no local constraints appear in the Partition problem. In each of the two test domains, the hierarchic solver outperforms the flat solver for sufficiently large problems. A series of paired, single-tailed T-Tests shows hierarchic scheduling to be faster than flat scheduling with significance greater than 99% on the five largest problem sizes. A series of paired, single-tailed T-Tests shows hierarchic partitioning to be faster than flat partitioning with significance greater than 99% on the two largest problem sizes.

5. Related Work

Abstraction techniques have been studied previously in the context of planning [Knoblock et a/., 1991] and theorem proving [Giunchiglia and Walsh, 1992]. In contrast, the research presented here is focused on abstraction techniques for constraint satisfaction problems. A program called "HiT" for automatically constructing abstraction spaces for CSPs is presented in [Mohan, 1991]. HiT uses the ranges of functions appearing in the goal definition as the basis for constructing abstraction spaces. HiT may be seen as a special case of the range symmetry technique described here. Methods of attacking hierarchical CSPs are discussed in [Mackworth et al, 1985]; however, these methods exploit existing hierarchies and do not construct new ones. Abstractions based on quotas have been studied in the context of resource allocation problems [Lowry and Linden, 1992]. Approximate symmetry provides a rational reconstruction of the quota concept. Furthermore, approximate symmetry is more general, because it depends only on the algebraic form of the problem and not on semantic notions such as "resources". Abstractions based on windows operate in a manner similar to interval constraint propagation [Davis, 1987]. Approximate symmetry provides a means of using such interval-based methods to attack problems with global constraints, to which constraint propagation techniques to not immediately apply. Techniques for recognizing and exploiting exact symmetries have been investigated in the context of propositional satisfiability [Crawford, 1992]. In contrast, the methods presented here construct and exploit symmetries that are not present in the original problem. Approximations and abstractions have been used to construct heuristic evaluation functions in the context of constraint satisfaction [Dechter and Pearl, 1987] and state space search [Prieditis, 1991]. A system that selects and combines multiple heuristics to synthesize CSP algorithms is described in [Minton, 1993a] and [Minton, 1993b].

6. Acknowledgments

This research is supported by the National Science Foundation. (Grants IRI-9017121 and IRI-9021607). It has benefited from discussions with Haym Hirsh and Christopher Tong and programming by Saibal Patra.

References

W. Braudaway and C. Tong. Automated synthesis of constrained generators. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, MI, 1989.

J. Crawford. A theoretical analysis of reasoning by symmetry in first order logic. Working Notes of the AAAI Workshop on Tractable Reasoning, San Jose, CA, 1992.

E. Davis. Constraint propagation with interval labels. Artificial Intelligence, 32:281 - 332, 1987.

R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. Artificial Intelligence, 34:1-38, 1987.

T. Ellman. Synthesis of abstraction hierarchies for constraint satisfaction by clustering approximately equivalent objects. In Proceedings of the Tenth International Conference on Machine Learning, Amherst, MA, 1993.

M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York, NY, 1979.

F. Giunchiglia and T. Walsh. A theory of abstraction. Artificial Intelligence, 57:323-389,1992.

C. Knoblock, J. Tennenberg, and Q. Yang. Characterizing abstraction hierarchies for planning. In Proceedings of the Ninth National Conference on Artificial Intelligence, Anaheim, CA, 1991.

R. E. Korf. Planning as search: A quantitative approach. Artificial Intelligence, 33(1):65 - 88, 1987.

M. Lowry and T. Linden. Generation and exploitation of aggregation abstractions for scheduling and resource allocation. Working Notes of the AAAI Workshop on Approximation and Abstraction of Computational Theories, San Jose, CA, 1992.

A. Mackworth, J. Mulder, and W. Havens. Hierarchical arc consistency: Exploiting structured domains in constraint satisfaction problems. Computational Intelligence, 1:118 - 126, 1985.

S. Minton. An analytic learning system for specializing heuristics. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Chambery, France, 1993.

S. Minton. Integrating heuristics for constraint satisfaction problems: A case study. In Proceedings of the Eleventh National Conference on Artificial Intelligence, Washington, D.C., 1993.

S. Mohan. Constructing hierarchical solvers for functional constraint satisfaction problems. Working notes of the AAAI Spring Symposium on Constraint-Based Reasoning, Stanford, CA, 1991.

B. Nadel. Tree search and arc consistency in constraint satisfaction algorithms. In L. Kanal and V. Kumar, editors, Search in Artificial Intelligence, pages 287 - 342. Springer Verlag, New York, NY, 1988.

A. Prieditis. Machine discovery of effective admissable heuristics. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, Sidney, Australia, 1991.