Hillclimbing in a Hierarchy of Abstraction Spaces

Thomas Ellman
Saibal Patra
Department of Computer Science
Hill Center for Mathematical Sciences
Rutgers University
New Brunswick, New Jersey 08903
(908) 932 - 4184
{ellman,patra}@cs.rutgers.edu
LCSR-TR-198

Abstract

Hillclimbing search has been shown to be useful for solving constraint satisfaction problems that are too large to be attacked using backtracking search. Nevertheless, hillclimbing search can be computationally expensive when the length of each climb is long, or when many climbs are required due to the presence of local, but non-global optima. "Hierarchic Hillclimbing" (HHC) is an extension of ordinary "Flat Hillclimbing" that is designed to attack such difficulties. HHC carries out hillclimbing search in a hierarchy of abstraction spaces, starting with the most abstract and proceeding to the most concrete. HHC takes as input a description of the abstraction hierarchy, as well as an evaluation function for each abstraction level. The HHC algorithm has been implemented along with a program to synthesize the required abstraction hierarchies and evaluation functions. The synthesis program and the HHC algorithm have been tested in the domains of uniprocessor scheduling and two dimensional tile packing. Results show HHC to improve in two ways on ordinary hillclimbing without abstraction: HHC requires less computation time to complete a single climb. In addition, when the abstraction hierarchy is chosen with care HHC is more likely to find a solution on a single climb.

Content Areas: Knowledge Compilation, Analytic Learning, Speedup Learning, Constraint-based reasoning.

1 Introduction

Hillclimbing search has been shown to be useful for solving constraint satisfaction problems that are too large to be attacked using backtracking search. For example, hillclimbing methods are shown to be effective in solving large boolean satisfiability problems [Selman *et al.*, 1992], [Gu, 1992], and in solving large scheduling problems [Minton *et al.*, 1992]. Nevertheless, hillclimbing search can be computationally expensive, especially when the length of each climb is long, or when many climbs are required due to the presence of local, but non-global optima. Consider the fact that abstraction techniques have been shown useful for enhancing the performance of backtracking search on constraint satisfaction problems [Ellman, 1993]. In light of this fact, one might ask whether abstraction can also improve the performance of hillclimbing search on constraint satisfaction problems. This paper reports on research into the possibility. In particular it develops a technique called "Hierarchic Hillclimbing" (HHC) which performs hillclimbing search in a hierarchy of abstraction spaces.

The behavior of Hierarchic Hillclimbing can be illustrated by considering the uniprocessor scheduling problem defined in Figure 2. A problem is defined by a set of jobs, a set of time slots, a working time for each job, a deadline for each job, and a precedence relation. Solution of the problem requires finding an assignment of starting times to jobs. The assignment must meet the deadlines, obey the precedence relation and never have two jobs running at once. In this problem, the original "concrete" search space is defined in the following way: A concrete state is simply an assignment of a starting time to each job. An "abstract" search space can be defined in the following way: First the set T of time slots is partitioned into contiguous, disjoint time windows. Each state in the abstract search space is then defined by an assignment of a starting window to each job.

Hierarchic Hillclimbing would begin by randomly assigning a starting window to each job. It would then carry out a hillclimbing search in the abstract space of starting window assignments. Each hillclimbing step would change one job's starting window to an adjacent window. Hillclimbing in the abstract space would terminate when all the problem constraints are met, so far as can be determined knowing only the starting window assignments. The algorithm would then move into the concrete search space. First the algorithm would choose a starting time for each job by randomly selecting a time slot inside the job's assigned starting window. It would then carry out a hillclimbing search in the concrete search space. Each step would change one job's starting time to an adjacent time slot. Hillclimbing in the concrete space would terminate when all the scheduling constraints are satisfied. The behavior of the algorithm is illustrated visually in Figure 1.

A method of automatically constructing Hierarchic Hillclimbing (HHC) problem solvers is presented in this paper. The basic HHC algorithm is presented in Section 2. The algorithm is shown to require two inputs in order to be applied to a particular constraint satisfaction problem: (1) A hierarchy of abstraction spaces; (2) An evaluation function for each space. A method of constructing abstraction hierarchies is presented in Section 3. The method operates by converting the original "goal function", which operates on states in a concrete search space, into an "abstract goal function", which operates on states in an abstract search space. (The abstraction technique is presented in more detail in [Ellman, 1993], along with experimental results obtained from using it to enhance the performance of backtracking search algorithms.) A method of constructing evaluation functions is pre-

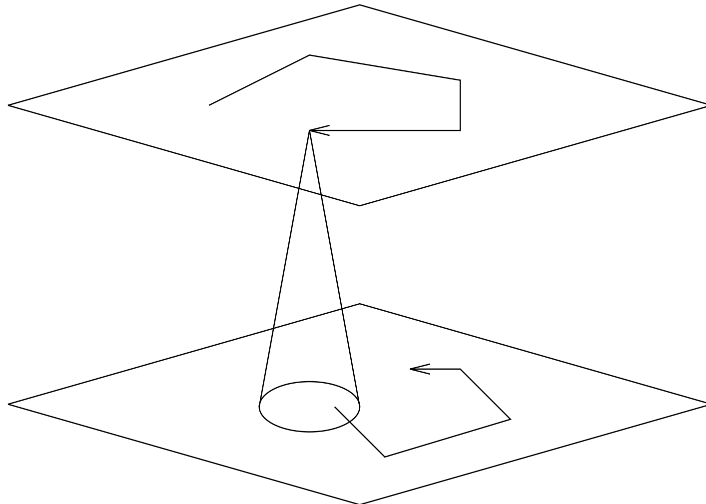

Figure 1: Hierarchic Hillclimbing

sented in Section 4. This method operates by converting abstract or concrete boolean goal functions into numeric functions that operate on abstract or concrete states. The methods of synthesizing abstraction hierarchies and evaluation functions have been implemented and tested in the domains of uniprocessor scheduling and two dimensional tile packing. These domains are defined in Figure 2. Experimental results from testing the HHC algorithm in these domains are presented in Section 5.

2 Hierarchic Hillclimbing

The "Hierarchic Hillclimbing" (HHC) algorithm is outlined in Figure 3. HHC uses a subroutine for ordinary "Flat Hillclimbing" shown in Figure 4. HHC takes as input an ordered list of search spaces, and an ordered list of evaluation functions. The first search space is the most abstract, and the last is the most concrete. HHC carries out an ordinary hillclimbing search in each of the search spaces, starting with the most abstract, and proceeding to the most concrete. At each level, hillclimbing is guided by the corresponding evaluation function. Whenever the hillclimber reaches a local optimum at a given level, one of two things may happen: (1) If the evaluation function indicates that the optimal state is promising, HHC randomly selects a refinement of the state, i.e., a state appearing in the next lower level of the hierarchy. HHC then proceeds to search the next level, starting at the refined state. (2) If the evaluation function indicates that the optimal state is not promising, HHC jumps back up to the highest level, and begins the entire process over again. (HHC could be easily modified to backtrack to intermediate levels, rather than to the highest level.) The algorithm terminates when it finds a solution in the most concrete space. HHC thus operates in a manner that is analogous to well known techniques for hierarchic planning [Sacerdoti, 1974], [Knoblock *et al.*, 1991].

An abstraction hierarchy may be technically defined as a sequence $(S_1, ..., S_n)$ of sets, where S_1 represents the most abstract space in the hierarchy, and S_n represents the most concrete space in the hierarchy. Each set S_i is a partition of the set S_{i+1}. Thus a state $s_i \epsilon S_i$ is a subset of the set S_{i+1}. A state r is said to be a "refinement" of state s provided that

- Tiling Problem:
  - Given: A set of rectangular tiles with specified lengths and widths; A square box.
  - Find: An assignment of positions and orientations to tiles such that each tile lies entirely within the box, and no two tiles overlap.
- Uniprocessor Scheduling Problem: (NP-Hard [Garey and Johnson, 1979])
  - Given: A set J of jobs; A set T of possible starting times; A working time $w(j)$ required to complete each job j; A deadline $d(j)$ by which time each job j must be completed; A precedence relation $p(j, k)$ indicating that job j must complete by the time job k begins.
  - Find: An assignment of starting times to jobs that meets the deadlines, obeys the precedence relation, and has at most one job running at a time.

Figure 2: Test Domains for Hierarchic Hillclimbing

res. The evaluation functions $(E_1, ..., E_N)$ are defined in the following way: Each evaluation function E_i maps the set S_i into the real interval $[0, 1]$. The mapping is required to obey the following rules: (1) For any state $s \epsilon S_i$, if $E_i(s) < 1$, then there exists no way to refine state s zero or more times into a state $t \epsilon S_n$ such that t satisfies the problem constraints. Thus the constraint $E_i(s) = 1$ is a necessary (but not sufficient) condition on refinability of state s into a concrete solution to the problem. (2) For any concrete state $s \epsilon S_n$, $E_n(s) = 1$ if and only if the state s satisfies the problem constraints. Condition (1) explains why the HHC algorithm backtracks when it reaches a locally optimum state s in space S_i for which $E_i(s) < 1$, i.e. no solution can be found by repeatedly refining such a state. Condition (2) explains why HHC terminates successfully when finding a concrete state that evaluates to 1.

"Hierarchic hillclimbing" is expected to perform better than ordinary "Flat Hillclimbing" for two reasons: (1) To begin with, Hierarchic Hillclimbing takes large steps during the beginning of a hillclimbing process (when moving in highly abstract spaces) and takes short steps during the final stages of hillclimbing (when moving in the concrete space). In contrast, Flat Hillclimbing takes short steps during the entire search. We therefore expect the overall path length, summed over all levels, to be shorter for Hierarchic Hillclimbing than for Flat Hillclimbing. The CPU time needed to complete a single climb is therefore likely to be lower in the hierarchic case. (2) In addition, we expect that Hierarchic Hillclimbing will avoid getting stuck at local, but non-global optima more often than Flat Hillclimbing. In particular, whenever the concrete space contains valleys that lie entirely within single abstract states, these valleys will be invisible from the corresponding abstraction space. The Hierarchic Hillclimber can potentially jump over such valleys in a single step, whereas the Flat Hillclimber may get stuck on one side. We therefore expect Hierarchic Hillclimbing to be more likely than Flat Hillclimbing to find a solution on a single climb. (By virtue of its ability to jump over valleys early in the search process, but not later on, Hierarchic Hillclimbing may be said to operate in a manner similar to simulated annealing [Kirkpatrick

Procedure *Hierarchic-Hillclimb-Top(S-List,E-List)*:

1. $S \leftarrow$ *Nil.*
2. *While* $S =$ *Nil do* $S \leftarrow$ *Hierarchic-Hillclimb(S-List,E-List,First(S-List)).*

Procedure *Hierarchic-Hillclimb(S-List,E-List,S)*:

- *S-List* is a list of search spaces.
- *E-List* is a list of evaluation functions.
- *S* is a subset of *First(S-List)*.

  If Empty(S-List) then Return(S) else

  1. $s \leftarrow$ *Random-Member(S).*
  2. $s \leftarrow$ *Hillclimb(First(S-List),First(E-List),s).*
  3. *If* $E(s) = 1$

     then Return(Hierarchic-Hillclimb(Rest(S-List),Rest(E-List),s))
     else Return(Nil).

Figure 3: Hierarchic Hillclimbing

et al., 1983],[Press *et al.*, 1986]. An alternative method of crossing valleys in hillclimbing search involves construction and use of "peak to peak" macro operators [Iba, 1989].)

3 Synthesis of Abstraction Spaces

The tiling and scheduling problems defined in Figure 2 both have the form of *function finding* problems. The specification of each problem includes a domain D (e.g., tiles or jobs) and a range R (e.g., positions or time slots). A solution is defined by a function f that maps the domain D into the range R (e.g., an assignment of positions to tiles or starting times to jobs). Two distinct types of abstraction can be defined for function finding problems [Ellman, 1993]. Abstraction based on *Range Symmetry* produces an abstraction space in which each state is function $\dot{f}$ mapping the domain D into a partition $\hat{R}$ of the original range R. The "window" abstraction for scheduling is an example of this type. Abstraction based on *Domain Symmetry* produces an abstraction space in which each state is a function $\dot{f}$ defined on a partition $\hat{D}$ of the original domain D, and returning multisets of values over the original range R. Although we believe that Hierarchic Hillclimbing methods apply to both *Range Symmetry* and *Domain Symmetry*, they have been tested only in the case of *Range Symmetry*. The following discussion therefore focuses on methods of synthesizing abstraction spaces based on *Range Symmetry*.

Abstractions based on *Range Symmetry* are constructed from the problem specifications of constraint satisfaction problems. The specifications are assumed to have the general form

Procedure *Hillclimb(S,E,s)*:

- S is a search space.
- E is an evaluation function.
- s is a state.

1. *While* ($\exists\ t \in$ *Neighbors(s,S)*) *such that* $E(t) > E(s)$ *do* $s \leftarrow t$
2. Return(s)

Procedure *Neighbors(s,S)*: *Return the set of all states reachable from state s by incrementing or decrementing a single state variable.*

Figure 4: Flat Hillclimbing

shown in Figure 5: (1) A description of the search space in terms an unknown function f; (2) A goal function G that assigns either "True" or "False" to candidate each solution depending on whether it satisfies the problem constraints. The synthesis procedure has two main steps: (1) For each concrete state variable v, representing the value of the unknown function f at a point in the domain D, define an abstract state variable $\dot{v}$, representing the value of an unknown abstract function $\dot{f}$ at the same point. (2) Transform the original "concrete" goal G into an "abstract" goal $\dot{G}$. The abstract goal $\dot{G}$ is constructed by systematically replacing all references to concrete state variables v with references to abstract state variables $\dot{v}$, and by replacing all arithmetic, relational and boolean operations in G with corresponding operations on sets: (Given an arbitrary function $h : A \rightarrow B$, the corresponding set function $\hat{h} : A \cup 2^A \rightarrow 2^B$, is defined so that $\hat{h}(x) = \{h(x)\}$, if $x \epsilon A$, and $\hat{h}(x) = \{h(y) | y \epsilon x\}$ if $x \subseteq A$.) Once this systematic replacement is complete, the resulting expression will return a set of boolean values. The abstract goal $\dot{G}$ is obtained by adding a test for the appearance of "True" in this returned set. The abstract goal so constructed is a *necessary* (but not sufficient) condition on the refinability of abstract states into solutions in the concrete search space. In the current implementation, it is assumed that the original range R is a set of integers and that the abstraction space is defined by partitioning R into intervals. Arithmetic operations are transformed into interval arithmetic operations. Relational operations on integers (mapping integers to booleans) are transformed into relational operations on intervals (mapping intervals to sets of booleans). Once the abstract goal is constructed, it is simplified using a set of equivalence-preserving optimizing transformations. When applied to the scheduling problem specification in Figure 5, this procedure constructs the abstract goal shown in Figure 6.

4 Synthesis of Evaluation Functions

Evaluation functions are used to measure progress toward satisfying problem constraints during hillclimbing search, as shown in the basic "Flat Hillclimbing" algorithm in Figure 4. Our system builds evaluation functions that are analogous to the "min-conflicts" heuristic

- Given:
  - Jobs J of type Symbol.
  - Times T of type Integer.
  - Working-Time $w : J \rightarrow T$
  - Deadline $d : J \rightarrow T$
  - Precedence $p : J \times J \rightarrow \{True, False\}$
- Find: Beginning-Time $b : J \rightarrow T$
- Goal Function:

$$
\begin{array}{lcl}
G(s) & = & G_1(s) \wedge G_2(s) \wedge G_3(s) \\
G_1(s) & = & (\forall j \epsilon J)\ \ b(j,s) + w(j) \leq d(j) \\
G_2(s) & = & (\forall j, j' \epsilon J)\ \ \neg p(j,j') \vee b(j',s) \geq b(j,s) + w(j) \\
G_3(s) & = & \neg\ \ (\exists j, j' \epsilon J)\ \ O(j,j',s) \\
O(j,j',s) & = & b(j,s) + w(j) \geq b(j',s) \wedge b(j,s) \leq b(j',s)
\end{array}
$$

Figure 5: Uniprocessor Scheduling Problem Class Specification

$$
\begin{array}{lcl}
\dot{G}(a) & = & (\forall j, j' \epsilon J)\ \ G_1(j,a) \wedge G_2(j,j',a) \wedge G_3(j,j',a) \\
G_1(j,a) & = & True\ \epsilon\ \{\dot{b}(j,a) \hat{+} \hat{w}(j) \hat{\leq} \hat{d}(j)\} \\
G_2(j,j',a) & = & True\ \epsilon\ \{\neg p(j,j')\} \vee True\ \epsilon\ \{\dot{b}(j',s) \hat{\geq} \dot{b}(j,s) \hat{+} \hat{w}(j)\} \\
G_3(j,j',a) & = & \{j = j'\}\ \vee False\ \epsilon\ \{\dot{b}(j,s) \hat{+} \hat{w}(j) \hat{\geq} \dot{b}(j',s)\} \vee False\ \epsilon\ \{\dot{b}(j,s) \hat{\leq} \dot{b}(j',s)\}
\end{array}
$$

Figure 6: Abstract Uniprocessor Scheduling Goal

discussed in [Minton *et al.*, 1992]. In particular, given a state s, our evaluation functions $E(s)$ identify all the problem constraints and assign each constraint a weight indicating the degree to which the constraint is satisfied. Satisfied constraints are given a value of 1. Unsatisfied constraints are given a value in the interval $[0, 1)$, where a value near 1 indicates the constraint is "close" to being satisfied, and a value near 0 indicates the constraint is "far" from being satisfied. $E(s)$ then returns the sum of weights of all the problem constraints. We conjecture that this approach yields a more sensitive (an effective) evaluation function than simply counting the number of violated constraints. Experiments to test this hypothesis are under way; however, they are not the focus of this paper.

Hierarchic Hillclimbing requires an evaluation function for each level in the abstraction hierarchy. Our system constructs each required evaluation function by transforming the goal function associated with each level in the hierarchy. For example, the evaluator for the concrete scheduling space is constructed from the original scheduling goal shown in Figure 5. Likewise, the evaluator for the abstract scheduling space is constructed from the abstract

scheduling goal shown in Figure 6. The process begins by putting each goal in conjunctive normal form. The constructions are then carried out by a set of rewrite rules that replace all boolean valued functions with numeric valued functions:

- **Relational Operations on Integers:** The functions $> \geq < \leq =$ are transformed into numeric valued functions that return 1 only when the integer arguments satisfy the relation. Otherwise they return a number in the interval $[0, 1)$, which increases as the two integers come closer to satisfying the relation.

- **Relational Operations on Intervals:** The functions $\hat{>} \hat{\geq} \hat{<} \hat{\leq} \hat{=}$ are transformed into numeric valued functions that return 1 only when some pair of integers drawn from the two interval arguments can be found to satisfy the relation. Otherwise they return a number in the interval $[0, 1)$, which increases as the two intervals come closer together.

- **Boolean Operations on Boolean Values:** Conjunction $\wedge$ is transformed into a function that simply computes the numeric average of its arguments. Disjunction $\vee$ is transformed into a numeric function that returns 1 if any sub-expression returns 1, and otherwise averages its arguments. Negation $\neg$ is transformed into a numeric function that returns 1 if its argument is less than 1, and returns 0 otherwise.

The constructed evaluation functions actually do somewhat more than is described above. In particular, each evaluation function returns a triple which has the form (T 1 NIL) (when the abstract or concrete goal is satisfied), and has the form (NIL N *Variable-List*) (when the goal is not satisfied). The first element indicates goal satisfaction or failure. The second element is the actual evaluation. The third element is a list of the state variables that are referenced by failed conjuncts of the goal. The implicated state variables are used in the *Neighbors* function of the flat hillclimber (Figure 4) to avoid constructing and evaluating states that are known a priori to have the same evaluation as the current state.

Observe that the constructed functions do indeed fulfill the requirements on evaluation functions specified in Section 2. Each evaluation function returns a value of 1 whenever the associated goal function returns a value of "True", and returns a value less than 1 whenever the associated goal function returns a value of "False". Since the abstract goal is a necessary condition on refinability of abstract solutions, it will return "False" only when the abstract state is not refinable into a concrete state satisfying the problem constraints. The abstract evaluation function therefore returns a value less than 1 only when the abstract state is not refinable, thus fulfilling the first condition described in Section 2. Since the concrete goal returns "True" only when the concrete state is a solution, the concrete evaluation function returns a value of 1 only when the concrete state is a solution, thus fulfilling the second condition described in Section 2.

5 Experimental Results

A series of experiments was run to compare the performance of Hierarchic Hillclimbing to the performance of ordinary Flat Hillclimbing. In these experiments the Hierarchic Hillclimbing algorithm used abstraction spaces and evaluation functions that were automatically synthesized using the methods described above. The Flat Hillclimbing algorithm used evaluation

functions that were synthesized automatically as well. The hierarchic and flat hillclimbers actually run the same underlying hillclimbing code. They behave differently because they are given different input specifications, describing different numbers of abstraction levels, i.e., two levels for the hierarchic case and one level for the flat case.

The algorithms were tested in both the scheduling and tiling domains described above. (See Figure 2.) Each algorithm was tested on a range of different problem sizes: In the scheduling domain, each problem set included 50 randomly generated problems. The number of time slots was varied across problems sets from a minimum of 50 slots, to a maximum of 100 slots. All scheduling problems included 5 jobs to be scheduled. In the tiling domain, each problem set also included 50 randomly generated problems. The width and length of the square box was varied from a minimum of 50 positions, to a maximum of 100 positions. All tiling problems included 5 tiles to be placed. Each algorithm performed exactly one (hierarchic or flat) climb on each test problem. (The hierarchic hillclimber did one climb on each of the two levels, abstract and concrete. The flat hillclimber did one climb on the concrete level.) For each problem, the system recorded both the total CPU time expended in attacking the problem, as well as an indication of whether the problem was solved successfully. The results of these experiments are shown in Figures 7 and 9 (for scheduling) and Figures 8 and 10 (for tiling).

The graphs in Figures 7 and 8 compare the average CPU time per problem across all problem sizes, for each of the two test domains. CPU time is a better measure than alternatives, like numbers of evaluations or numbers of states generated, for two reasons: First, the computational cost of generating one state or evaluating one state may not be the same in the flat and hierarchic problem solvers. Second, the relative importance of state generation and state evaluation may not be the same in the flat and hierarchic problem solvers. Notice that the hierarchic hillclimber consistently requires less CPU time per problem than the flat hillclimber. A series of single-tailed, paired T-Tests shows that the hierarchic algorithm beats the flat algorithm with significance 98% or greater on each problem size in the scheduling domain. Likewise, in the tiling domain, single-tailed, paired T-Tests show that the hierarchic algorithm beats the flat algorithm with significance 95% or greater on all but the smallest problem size. Thus the hierarchic method is consistently faster, despite the fact that the hierarchic hillclimber performs one abstract and one concrete hillclimb, while the flat hillclimber performs only one hillclimb, at the concrete level. We conjecture that the hierarchic hillclimber has better CPU time because it follows a path (of abstract and concrete states) that is shorter than the path (of concrete states) followed by the flat hillclimber. Although the hierarchic hillclimber performs two climbs, its climb in the concrete space tends to be shorter by more than the length of its climb in the abstract space. This occurs because the hierarchic hillclimber enters the concrete level at a refinement of an abstract solution. This point is more likely to be near the top of a hill than a randomly selected concrete state.

The tables in Figures 9 and 10 compare the fractions of problems solved successfully across all problem sizes, for each of the two test domains. This measure is important because hillclimbing is not guaranteed to find a solution on a single climb, even if one exists, due to the presence of local, but non-global optima in the search space. Thus a hillclimber often needs to perform multiple climbs, from random starting points, in order to have a good chance of finding a solution. If a solution exists, and the hillclimber does repeated climbs until finding one, the required number of climbs will depend on the probability of success on

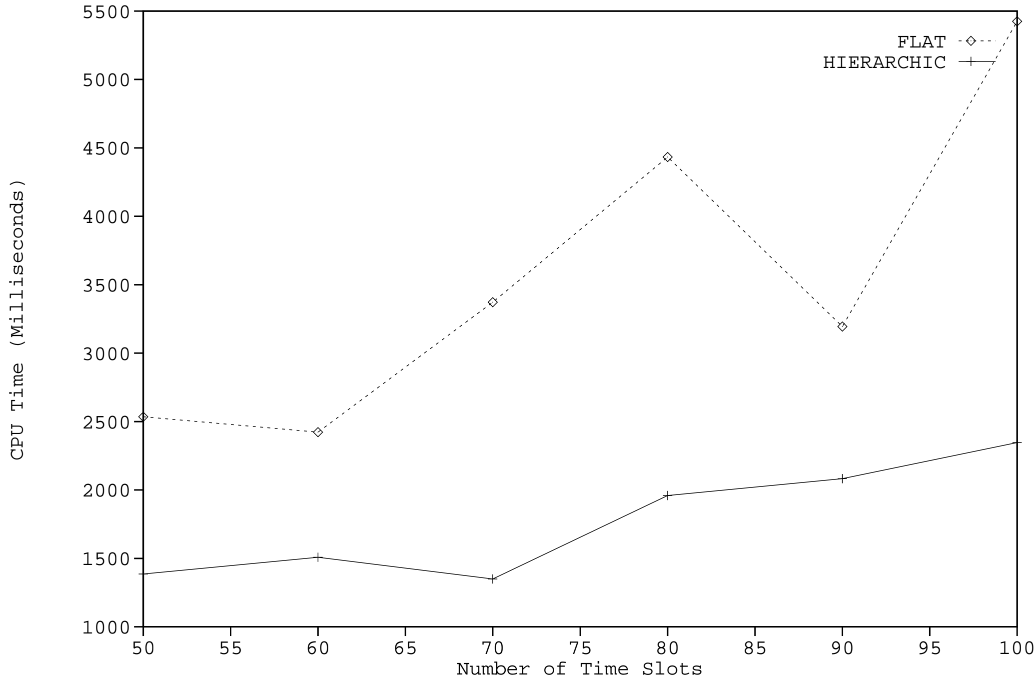

Figure 7: Average CPU Time per Scheduling Problem

a single climb. Thus the relative performance of flat and hierarchic hillclimbing depends in part on the likelihood that each will solve a problem on a single hillclimbing run.

In the scheduling domain, hierarchic hillclimbing solves a greater fraction of the test problems on each of the problem sizes; however, a Chi-Square test shows that only one of these differences for individual problem sizes has significance as high as 95%. Nevertheless, when all scheduling problem sizes are averaged together, a Chi-Square test shows the hierarchic method to solve a greater fraction of the problems, with significance level of greater than 99%. In the tiling domain, the situation is less clear. For most problem sizes the hierarchic method does solve a greater fraction of the problems; however, the differences are small and a Chi-Square test shows none of the differences to have significance as high as 95% in favor of either method. When all tiling problem sizes are averaged together, hierarchic hillclimbing solves a greater fraction of the problems; however, a Chi-Square test shows the difference to have significance less than 90%, i.e., not significant.

We looked carefully at the scheduling and tiling problems in order to interpret these results: We observed the following phenomenon: In the abstract scheduling space, the scheduling windows are large compared to the average job length. As a result, when climbing in the abstract space, the hierarchic hillclimber can reverse the order of two adjacent jobs, in a single step, without incurring the penalty for overlapping jobs. The hierarchic method thus avoids getting stuck at some local, but non-global optima. This explains why hierarchic hillclimbing solves significantly more scheduling problems than flat hillclimbing. In contrast to this, in the abstract tiling space, the position intervals are small in comparison to the tile dimensions. The hillclimber therefore cannot move tiles past each other without incurring

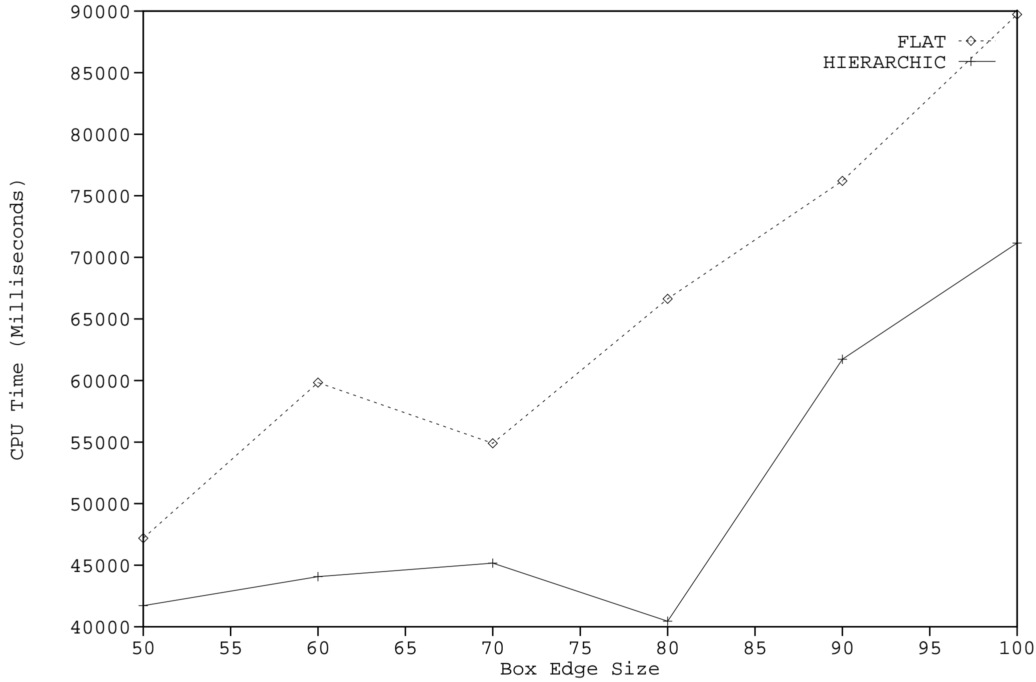

Figure 8: Average CPU Time per Tiling Problem

the overlap penalty, even when moving in the abstract space. This explains why hierarchic hillclimbing does not solve significantly more tiling problems than flat hillclimbing.

Our experiments suggest that hierarchic hillclimbing will sometimes achieve a greater probability of finding a solution; however, the effect will not always be present. We conjecture that that the effect depends critically on two things: (1) The width of valleys in the concrete search space; (2) The diameter of the set of concrete states corresponding to each abstract state. If the diameter is large compared to the width of the valley, then a single step in the abstract space can jump over the valley. A hierarchic hillclimber can thus avoid getting stuck at a local, but non-global optima. If the diameter is small compared to the width of

Number of Time Slots	Fraction Solved (Flat)	Fraction Solved (Hierarchic)
50	0.16	0.26
60	0.1	0.3
70	0.12	0.18
80	0.14	0.2
90	0.18	0.22
100	0.12	0.26
All Problems	0.14	0.24

Figure 9: Fraction of Scheduling Problems Solved

Box Edge Size	Fraction Solved (Flat)	Fraction Solved (Hierarchic)
50	0.08	0.1
60	0.08	0.2
70	0.14	0.14
80	0.04	0.06
90	0.12	0.08
100	0.06	0.02
All Problems	0.09	0.1

Figure 10: Fraction of Tiling Problems Solved

the valley, then a single step in the abstract space may not jump over the valley, and the hierarchic hillclimber will be less likely to avoid getting stuck. If this conjecture is correct, one might enhance the performance of hierarchic hillclimbing by carefully choosing the degree of abstraction, i.e., the size of abstract states and the corresponding size of abstraction spaces. One might also see benefits from use of a multi-level abstraction hierarchy, especially in problems for which the concrete space exhibits valleys with widely varying widths.

6 Summary

Our experiments support the overall conclusion that Hierarchic Hillclimbing (HHC) performs better than Flat Hillclimbing. Hierarchic Hillclimbing takes less time to carry out a single climb through two spaces (abstract and concrete) than flat hillclimbing requires to carry out a single climb through one (concrete) space. HHC achieves this advantage by taking large steps early in the search process, and taking small steps during the final stages of search. In some cases, Hierarchic Hillclimbing is also more likely to solve a constraint satisfaction problem on a single climb. HHC achieves this advantage because abstraction can smooth out the regions surrounding some locally, but non-globally optimal points that would cause ordinary Flat Hillclimbing to get stuck. We therefore conclude that the expected CPU time needed to solve a problem using multiple climbs will be lower when hillclimbing in a hierarchy of abstraction spaces.

7 Acknowledgments

The research reported in this paper is supported by the National Science Foundation under grants IRI-9017121 and IRI-9021607. This research has also benefited from discussions with Haym Hirsh, Christopher Tong and Kerstin Voigt.

References

[Ellman, 1993] T. Ellman. Abstraction via approximate symmetry. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France, August 1993.

[Garey and Johnson, 1979] 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.

[Gu, 1992] J. Gu. Efficient local search for very large scale satisfiability problems. *SIGART Bulliten*, 3(1), 1992.

[Iba, 1989] G. Iba. A heuristic approach to the discovery of macro operators. *Machine Learning*, 3(4):285–318, 1989.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671 – 680, 1983.

[Knoblock *et al.*, 1991] 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.

[Minton *et al.*, 1992] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1):161 – 206, 1992.

[Press *et al.*, 1986] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes.* Cambridge University Press, New York, NY, 1986.

[Sacerdoti, 1974] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115 – 135, 1974.

[Selman *et al.*, 1992] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA, 1992.