Heuristic-Biased Stochastic Sampling

John L. Bresina

Recom Technologies
IC Division, Mail Stop: 269-2
NASA Ames Research Center
Moffett Field, CA 94035-1000 USA
E-mail: `bresina@ptolemy.arc.nasa.gov`
URL: `http://ic-www.arc.nasa.gov/ic/projects/xfr/`

Abstract

This paper presents a search technique for scheduling problems, called *Heuristic-Biased Stochastic Sampling (HBSS).* The underlying assumption behind the HBSS approach is that strictly adhering to a search heuristic often does not yield the best solution and, therefore, exploration off the heuristic path can prove fruitful. Within the HBSS approach, the balance between heuristic adherence and exploration can be controlled according to the confidence one has in the heuristic. By varying this balance, encoded as a *bias function*, the HBSS approach encompasses a family of search algorithms of which greedy search and completely random search are extreme members. We present empirical results from an application of HBSS to the real-world problem of observation scheduling. These results show that with the proper bias function, it can be easy to outperform greedy search.

Introducing HBSS

This paper presents a search technique, called *Heuristic-Biased Stochastic Sampling (HBSS)*, that was designed to solve scheduling problems and other constrained optimization problems. The motivating idea behind the HBSS approach is, briefly, that heuristics encode advice which is generally useful, but should be taken with "a grain of salt". Deriving heuristics that are both accurate and computationally inexpensive is a difficult endeavor for most problems. This is especially true when not just any solution is acceptable and the heuristic is further required to find a high quality solution. Furthermore, the larger the class of problem instances, the more difficult it is for a search heuristic to perform consistently well for each instance. The performance quality of search techniques (e.g., greedy search) can be too dependent on the accuracy of the search heuristic employed. The underlying assumption behind the HBSS approach is that strictly adhering to a search heuristic often does not yield the best solution and, therefore, that exploration off the heuristic path can prove fruitful. Within HBSS, the balance between heuristic adherence and exploration can be controlled according to the confidence one has in the heuristic. By varying this balance, the HBSS approach encompasses a family of search algorithms of which greedy search and completely random search are extreme members.

Heuristic-biased stochastic sampling can be viewed as a generalization of *iterative sampling* (Langley 1992; Minton, Bresina, & Drummond 1994). Iterative sampling operates in a *refinement* search space formulated as a n-ary tree in which each internal node corresponds to a *partial* solution, and the leaf nodes either correspond to a complete solution or a failure. A solution is generated by starting at the root node and incrementally selecting a trajectory in the tree by making a random choice at each decision point. For search trees which do not have finite depth, a depth bound is used. This process is repeated some number of times, always restarting from the root node. No memory is kept of the trajectories selected on past iterations; i.e., the search space is sampled with replacement and, hence, the technique is nonsystematic (redundant).

The overall control structure of HBSS is the same as for iterative sampling; the difference lies in how a choice is made at each decision point. In contrast to iterative sampling's random exploration of the search tree, HBSS makes use of a given search heuristic to focus its exploration. The degree of this "heuristic focusing" is determined by a given *bias function.* The selection of the bias function typically reflects the confidence one has in the heuristic's accuracy – the higher the confidence, the stronger the bias. When employing a weaker bias, there is a greater degree of exploration, yielding a greater variety in the solutions found; in contrast, when employing a stronger bias, fewer unique solutions are produced.

The HBSS algorithm is given in Figures 1 & 2; the `HBSS` routine invokes the `HBSS-search` routine to perform a sampling iteration resulting in either a failure or a solution. At each decision point within `HBSS-search`, the alternative choices are sorted according to the given heuristic, and each alternative is assigned a rank based on this sort. We assume that ranks are positive integers; i.e., the top rank is 1. The bias function is then used to assign a non-negative weight to each choice based on its rank. These weights are then normalized by dividing each one by the sum of the weights.

```
HBSS(N, objective, state0, heuristic,
     bias, bound)
  for j = 1 to N {
    result :=
      HBSS-search(0, state0, heuristic,
                  bias, bound)
    unless result = failure {
      score := objective(result)
      when score is better than best_score {
        best_score := score
        best_solution := result }
    }
  }
 return (best_solution, best_score)
end
```

Figure 1: Heuristic-biased stochastic sampling alg.

The normalized weight for an alternative represents its probability of being selected; an alternative is selected according to these probabilities by a weighted random process. On each iteration, the current best schedule and best score are determined. When HBSS is employed in schedule generation, the best schedule and best score are initialized based on the results of a greedy search. However, this initialization step is omitted from Figure 1 because it is not used in the experiments reported in order to better compare the performance of the different bias functions.

The HBSS algorithm can perform iterative sampling by defining the bias function to give equal weight to each alternative, for example, $\{\text{bias}(r) = 1\}$, where r is the rank. At the other extreme, the following bias function causes the algorithm to perform greedy search: $\{\text{bias}(1) = 1;\ \text{for } r > 1,\ \text{bias}(r) = 0\}$. The following are example points in the bias spectrum.

- *logarithmic*: $\text{bias}(r) = \log^{-1}(r+1)$
- *linear*: $\text{bias}(r) = 1/r$
- *polynomial(n)*: $\text{bias}(r) = r^{-n}$
- *exponential*: $\text{bias}(r) = e^{-r}$

Figure 3 shows an example of how six bias functions assign selection probabilities, where "poly(n)" is a polynomial of degree n; the random bias (as in iterative sampling) is also shown. In the example, we assume that there are thirty alternatives to select among; the figure lists the selection probabilities for the top five ranked choices and the probability of selecting one of the remaining 25 choices. This example shows that the 4^{th} degree polynomial is the strongest bias function, in the sense that it follows the heuristic's recommendation more often than the other bias functions. In terms of bias strength, the exponential function is between the 2^{nd} and 3^{rd} degree polynomials.

```
HBSS-search(depth, state, heuristic,
            bias, bound)
  if solved?(state)
  then { return solution(state) }
  else-if state is a leaf or depth > bound
  then { return failure }
  else {
    for each child of state {
      score[child] := heuristic(child) }
    sort child nodes based on scores
    total_weight := 0
    for each child {
      rank[child] := sort position
      weight[child] := bias(rank[child])
      total_weight :=
          total_weight + weight[child] }
    for each child of state {
      probability[child] :=
          weight[child] / total_weight }
    child := weighted random selection
             based on probability
    depth := depth + 1
    HBSS-search(depth, child, heuristic,
                bias, bound)
  }
end
```

Figure 2: HBSS subroutine that performs one iteration

Observation Scheduling Problem

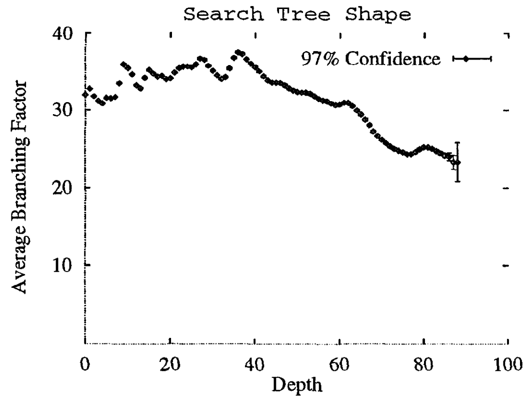
Before describing the experiments, we first briefly describe the observation scheduling problem; for further details see Bresina, *et al.* (1994) and Drummond, *et al.* (1995). The input to the scheduler is a set of observation requests submitted by one or more astronomers (for details on the request specification language, see Boyd, *et al.*, 1993). Each observation request specifies an *observation program* as well as constraints and preferences regarding when it can be executed. An observation program is a sequence of telescope movement and instrument commands.

A program is *enabled* on a given night if all of its constraints are satisfied. The primary constraint is that each request can be executed only within a specific time interval. On a given night, the observation program can only be executed during the intersection of the program's interval and that night's interval of darkness. Enablement is also affected by the moon – each program includes a constraint regarding whether the moon must be up, down, or either. Furthermore, even those programs that are enabled when the moon is up cannot be executed when its observation targets are too close to the moon. We refer to the time interval (for a given night) during which a program can begin execution as its *enablement interval.*

An example of a preference is the relative priority that an astronomer assigns to each observation request.

	Random	Log	Linear	Poly(2)	Poly(3)	Poly(4)	Exp
1	0.033	0.109	0.250	0.620	0.832	0.924	0.632
2	0.033	0.069	0.125	0.155	0.104	0.058	0.233
3	0.033	0.055	0.083	0.069	0.031	0.011	0.086
4	0.033	0.047	0.063	0.039	0.013	0.004	0.031
5	0.033	0.042	0.050	0.025	0.007	0.001	0.012
6–30	0.833	0.678	0.429	0.092	0.013	0.002	0.006

Figure 3: Table showing, for each bias function (and random), the selection probabilities assigned to the top five ranked choices and the probability of selecting one of the remaining 25 choices.



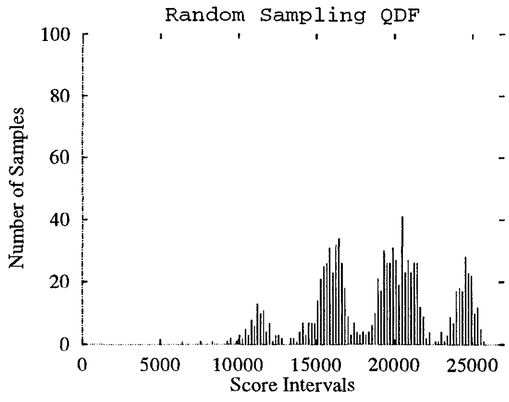


Figure 4: Shape and expected schedule quality of search tree.

On many nights, not all of the possible requests can be executed; these relative priorities help determine which subset to execute on a given night. Each observation program typically takes around five minutes to execute, and between 60 and 140 programs can be executed during a night. We refer to a night as *overloaded* if the sum of each enabled observation program's duration is greater than the night's duration; if the enabled duration sum is less than the night's duration, then we refer to the night as *underloaded*.

The scheduler's task is to find a sequence of observation programs that achieves a good score according to a given objective function. The primary objective is to execute as many of the programs as possible. A schedule is penalized for those enabled programs that are missed (not in the schedule); the penalty is greater for higher priority programs. The secondary objectives are to maximize data quality by minimizing airmass and to minimize telescope slewing. In the experiments reported here, the objective function used to evaluate schedules is defined as: `100` $\times$ `missed penalty` $+$ `10` $\times$ `average airmass` $+$ `average slew`. The weights are set such that the airmass attribute only distinguishes between schedules which have the same missed penalty score, and the slew attribute only distinguishes between schedules which are equivalent with respect to the other two attributes. The highest priority is 1 and the lowest priority is 10; the *missed penalty* score is $\sum[11 - \text{priority}(O)]^2$, where the summation is over the missed observation programs, O. The other two attribute scores are computed from the local hour angle of the observations; their definitions are not important for this paper. For this objective function, lower scores are better.

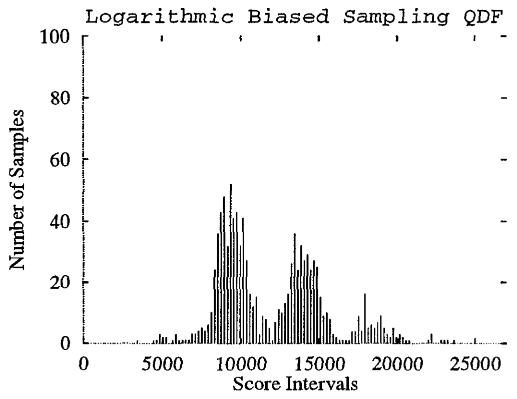
depth	μ:	84.85	σ:	1.35
size	μ:	2.9×10^{131}	σ:	5.7×10^{132}
	min:	3.4×10^{121}	max:	1.7×10^{134}

Figure 5: Depth and size of search tree.

The search space is formulated as a chronologically organized tree where the root node of the tree contains the world model state at dusk. The alternative arcs out of a given node represent the enabled observation programs. An arc connecting two nodes represents the simulated execution of the observation program, which (mainly) consists of incrementing the clock time by the estimated mean duration of the program. No observations can be done after dawn, so the search tree has finite depth. Note that each node corresponds to a unique feasible schedule prefix and, hence, each leaf node corresponds to a unique feasible complete schedule; there are no failure nodes.

HBSS Experiments

In this section, we illustrate the heuristic-biased stochastic sampling approach by presenting a compar-



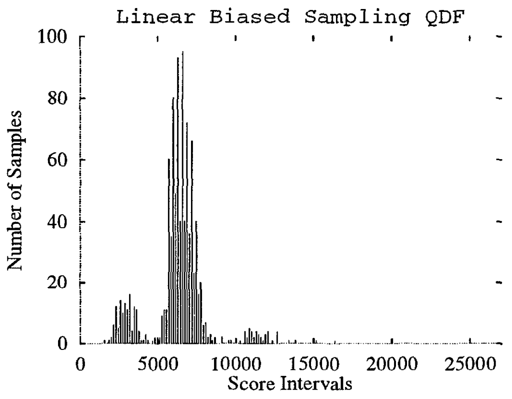


Figure 6: Quality Density Functions for logarithmic bias and linear bias.

ative analysis of six bias functions: logarithmic, linear, exponential, and polynomial of degree 2, 3, and 4. The nature of a scheduling problem in this domain depends on the observation programs and on the particular night (expressed as Julian Date, or JD). Our analysis is based on the (roughly) 150 observation programs that, for the past several years, have been executing on an automatic telescope at Fairborn Observatory (see Henry 1995). These observation programs are grouped into four priority classes with priorities of 4, 5, 9, and 10. The analysis is based on the problem instance corresponding to Julian Date 2449930, which is July 31, 1995. On JD 2449930 the moon is down most of the night so the fainter stars can be observed; this tends to overload telescope. During this night, 99 of the programs are enabled, with a duration sum of 554.67 minutes. There are 505.19 minutes of observation time, resulting in 49.48 minutes of overloading.

Before comparing the impact of the different bias functions, we first statistically characterize the scheduling problem. The shape and size statistics for JD 2449930 are given in Figures 4 and 5. Note that the number of enabled programs (i.e., average branching factor) remains high all night, indicating overloading. These statistics are based on 1000 trials of iterative sampling and the size estimates were computed using the method of Knuth (1975). With a search size of $\mathcal{O}(10^{131})$, iterative sampling has a minuscule chance of finding a schedule better than the greedy solution. The schedules have approximately equal length, as indicated by the low standard deviation (σ) of the depth, and the branching factor within each depth is nearly constant, as indicated by the error bars representing the 97% confidence interval.[1] These two results imply that the possible schedules will be sampled almost uniformly by the iterative sampling process.

[1]Except for the rightmost points, the error bars are too small to distinguish in the figure; the larger error at the deepest points is mainly the result of a reduced sample size because of schedule length variance.

Also included in Figure 4 is the *Quality Density Function (QDF)* (Bresina, Drummond, & Swanson 1995) for the search tree. A QDF is a statistical estimate of the expected density of schedules within different quality ranges and is based on the objective function scores obtained via iterative sampling. Recall that for the objective function (defined in the previous section), smaller scores indicate better schedules. To construct the QDF, 1000 randomly generated schedules were scored by the objective function and the scores were then quantized into 100 intervals of equal size. In the QDF (Figure 4), an impulse line at the midpoint of each interval indicates the number of schedules that scored within that interval. The mean of the scores is 18841.17 and the standard deviation is 3896.61. The QDF gives an indication of expected schedule quality under *uninformed* sampling and, hence, serves as a baseline against which informed sampling with different bias functions can be compared.

We next define the search heuristic. The heuristic is employed to help determine which enabled observation program to select at each decision point; the program with the lowest heuristic score is the most preferred. The heuristic is a multi-attribute function based on a set of observation selection rules (Boyd *et al.* 1993) used as a dispatch policy in automatic telescope controllers. There are four dispatch rules which are applied in this sequence, and each rule only breaks ties that remain from the application of preceding rules; the last rule is used to arbitrarily break any remaining ties. Our heuristic function has three attributes, corresponding to the first three dispatch rules, and is defined as: $w_1\times$ `priority` $+w_2\times$ `run count` $+w_3\times$ `enablement time`, where *run count* is the number of times the program occurs in the current schedule prefix and *enablement time* is the time currently remaining in the program's enablement interval.

For the experiments reported here, the attribute weights used in the search heuristic are: $w_1 = 0.4$, $w_2 = 0.4$, $w_3 = 0.3$. These attribute weights were the

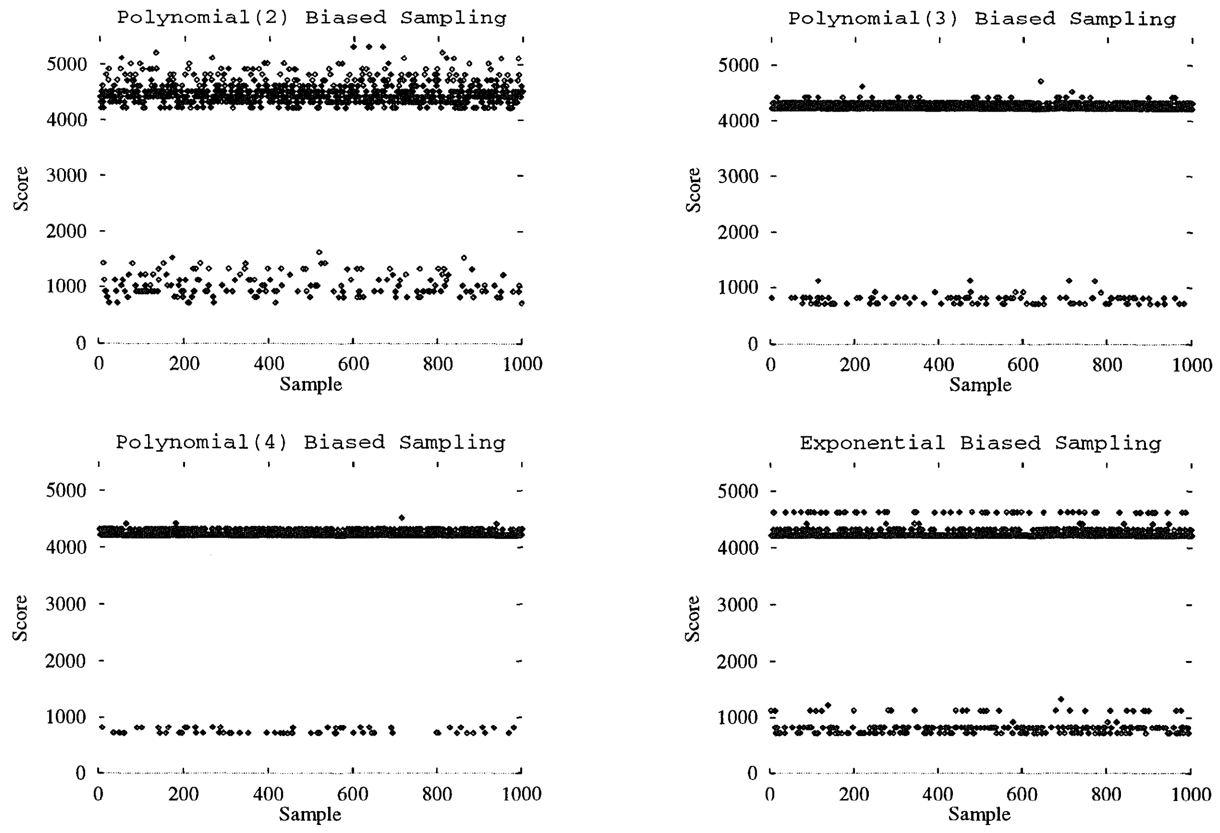


Figure 7: Scores obtained by HBSS with different bias functions.

result of performance tuning by a colleague; what is important to note about these weights is that they make the heuristic behave like the dispatch rules, due to the ranges of values for the different attributes. In fact, for JD 2449930, greedy search with these weights and the dispatch policy generate identical schedules that have an objective score of 4220.35.

We next report the results of the first part of the comparative analysis of the six bias functions. For each bias function, using the same heuristic function, 1000 iterations of the **HBSS-search** algorithm (Figure 2) were performed. Each resulting schedule was scored by the objective function (defined in the previous section). Figure 6 shows the quality density functions for logarithmic and linear bias functions. Compared to the random QDF in Figure 4, these plots show that as bias strength is increased, two changes tend to occur in the QDFs: the expected quality mass shifts left to a range of better scores close to the greedy score and this mass becomes less dispersed. However, this does not mean that the stronger the bias is, the better. A stronger bias tends to produce solutions more similar to the greedy search solution; hence, in order to significantly outperform greedy search, the right degree of exploration must be used.

The scores for the three polynomial bias functions and the exponential bias function tend to cluster in only a few score intervals; hence, instead of plotting their QDFs, in Figure 7 each sample's score is plotted. The two primary clusters (in Figure 7) result from whether or not the heuristic was followed on the first scheduling decision; this is further explained below.

The second part of the comparative analysis evaluates the performance of HBSS in terms of the best schedule's score after a given number of samples. For each of the six bias functions, we collected data regarding the best score after a sample size of 1, 5, 10, 15, and 20. Given a bias function and a sample size, a trial consisted of running HBSS and collecting the best objective score. For each pair of bias and sample size, 100 trials were run and the mean value of the best scores was computed.[2]

Figure 8 shows a plot of the mean values for the six bias functions; the score obtained by greedy search is plotted as a reference line. As can be seen in Figure 8, it does not take many samples to find a better sched-

[2]Note that the number of times HBSS-search is invoked by HBSS is 100 × sample size. However, regardless of sample size, the mean value is based on 100 scores; although, sample size does affect the standard deviation.

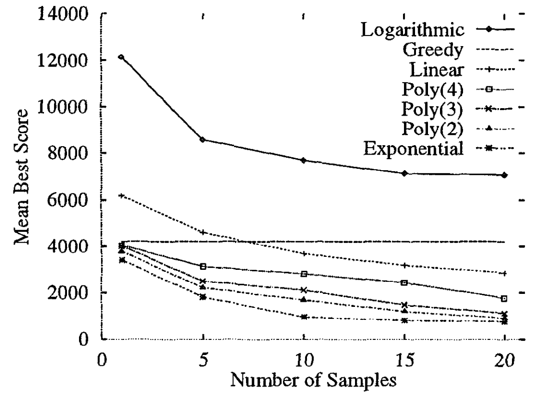


Figure 8: Mean value (averaged over 100 trials) of best score after a given number of samples (1, 5, 10, 15, & 20) for five bias functions, plus the greedy score.

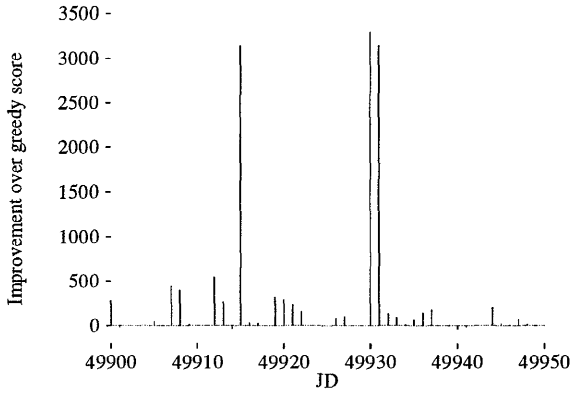


Figure 9: Mean value (averaged over 100 trials) of best score after 10 samples, with exponential bias, compared to greedy search.

ule than found by greedy search – except when the bias is very weak (e.g., logarithmic). The figure also shows that the exponential and 2^{nd} degree polynomial bias functions outperform the two weaker and the two stronger bias functions.

Why is it so easy to outdo greedy search, as well as the dispatch rules, on this problem? The dispatch rules have been employed by telescope controllers for more than six years with overall good performance. However, for the scheduling problem of JD 2449930, there is one scheduling decision where the normally good dispatch policy does not yield the best choice. This decision has a significant impact on schedule quality; it is what differentiates between the two score clusters shown in the plots of Figure 7.

On JD 2449930, there is a priority 5 observation program that can only start execution during the first five minutes of the night (after then, it sinks below the horizon). The search heuristic and the dispatch rules do not select this program to execute first because there is also a priority 4 program enabled at the beginning of the night. Since that program takes over six minutes to execute, the lower priority program can not be executed afterwards. However, the enablement interval of the higher priority program is over an hour long; hence, its execution can be postponed. When HBSS follows the heuristic on the first selection, the schedules found score in the higher (worse) score cluster; whereas those schedules that begin with the second ranked program instead, score in the lower score cluster (Figure 7). Although it is generally a good idea to prefer higher priority programs to execute, this is an example of where it is not the best choice and where deviation from the heuristic can improve schedule quality.

The experiments reported above focus on a single day; to lend support to the claim that HBSS is generally useful in this domain, we measured the improvement over greedy search achievable with 10 samples of HBSS using the exponential bias over 50 days. For each day, the average best 10-sample score was computed by running 100 trials (of 10 samples each). The results, shown in Figure 9, indicate that even with only 10 samples, usually some improvement can be made, and occasionally a substantial improvement can be made.

Related Research

We now discuss research related to our work with heuristic-biased stochastic sampling. As an illustration of the generality of the HBSS algorithm, we explain how the related search techniques can be expressed within the HBSS framework.

In Harvey's dissertation on nonsystematic backtrack search (Harvey 1995, section 4.8.1), he briefly mentions some stochastic sampling experiments in a more restricted context than considered here. The experiments involve satisfiability problems formulated as binary search trees. The bias is specified as a constant representing the probability of following the heuristic choice; hence, a bias of 0.5 yields an iterative sampling search and a bias of 1.0 yields a greedy search. In terms of the HBSS formalism, this is expressed as: $\{\text{bias}(1) = c\ ;\ \text{bias}(2) = 1 - c\}$, where c is the user-selected constant. Counter to his initial intuitions, Harvey found that these two extremes performed better than an intermediate bias of 0.875. However, when he added "bounded backtracking", then the intermediate bias outperformed the two extremes. The explanation of these two contrasting results was suggested as a topic for further investigation. In his dissertation, Harvey also presents *limited discrepancy search*, which systematically enumerates paths in a (binary) search tree, such that the paths are ordered by the number of times the heuristic choice was not followed. The first algorithm that performed this type of exploration was *left-first search* (Basin & Walsh 1993), which was used to find minimal difference unifications.

Related to heuristic-biased stochastic sampling is the class of *local search* algorithms, especially those that use stochastic search moves. Local search approaches are formulated as *repair* (or modification) search; that is, the search space is formulated with nodes representing complete solutions[3] and edges representing small *local* solution modifications (i.e., neighboring solutions are similar). Search (typically) starts from a randomly generated solution; at each node, one of the neighboring solutions is selected. This process continues until some stopping condition is satisfied. Often this local search process is repeated with new random starting solutions and the best solution is returned.

A given evaluation function is used to score the neighboring solutions; conventionally, lower scores represent better solutions. If all neighbors are worse than the current solution, then the search is at a *local minimum*, which is only rarely the global minimum. One approach is to simply restart the search from a new randomly generated solution whenever a local minimum is reached. Other approaches involve the introduction of randomness or noise into neighbor selection in order to escape local minima. Stochastic local search is a growing area of research and we mention only some example algorithms to convey the relationship between this algorithm class and the HBSS technique.

Though HBSS was not designed to operate in a repair search space, the algorithm (Figures 1 & 2) can still be applied with only minor modification. For a local search context, the depth bound in HBSS performs the function of the stopping condition. In refinement search, the initial state (`state0`) is the same for each iteration of HBSS; however, in repair search, it would correspond to a new randomly generated solution. The comparison of a neighbor and the current solution is often a key aspect of local search. One way to incorporate this comparison into HBSS is to include, as a parameter to the bias function, the rank of the current solution with respect to its neighbors.

Within local search, an analog to iterative sampling is *random walk* and both the *min-conflicts* local search strategy (Minton *et al.* 1990) and GSAT (Selman, Levesque, & Mitchell 1992) can be viewed as analogs to greedy search (with random restarts).

GSAT simply ignores local minima – it always selects the neighbor with the lowest score (even if greater than the current score) and restarts after a given maximum number of moves. Selman and colleagues have developed various extensions to the basic GSAT algorithm, some which combine aspects of random walk and GSAT's greedy search, e.g., *mixed random walk strategy* (Selman & Kautz 1993). This strategy can be expressed within HBSS as: $\{\text{bias}(1) = 1 - p\ ;$ for $r > 1$, $\text{bias}(r) = p/(n-1)\}$, where p is a user-specified constant and n is the number of choices.[4]

In *simulated annealing* (Kirkpatrick, Gelatt, & Vecchi 1983) noise is introduced as follows. A random neighbor is repeatedly chosen until one is selected (accepted) by the following decision process. Let ΔQ be a measure of the quality difference between the neighbor solution and the current solution; hence, if $\Delta Q > 0$, then the neighbor solution is worse. If $\Delta Q \leq 0$, then select the neighbor; otherwise, with a probability of $e^{-\Delta Q/T}$, select it anyway. T is the *temperature* parameter, which can be held constant, but more typically, starts out high and is reduced according to some "cooling schedule". To produce simulated annealing within the HBSS framework, the bias function needs the rank of the current solution (as discussed above). Let r_c be the current solution's rank (w.r.t. to its neighbors) and define the bias as follows. $\{\text{bias}(r_c, r) = 1.0$, if $r \leq r_c\ ;\ \text{bias}(r_c, r) = e^{(r_c - r)/T}$, if $r > r_c\}$.

Though quite different in operation from HBSS, the *Bayesian Problem Solver* (Hansson & Mayer 1989) shares the underlying assumption that there is uncertainty in the search heuristic. Within the BPS approach, heuristic search is viewed as inference from uncertain evidence. Heuristic uncertainty is explicitly modeled based on results of previous search and a Bayesian inference procedure is used to control search.

Concluding Remarks

In this paper, we introduced the *Heuristic-Biased Stochastic Sampling* approach for constrained optimization problems and, to illustrate its application, we presented a comparative analysis of six different bias functions using the real-world problem of observation scheduling. As with most advice, the key is knowing when to follow it and when to make an exception. The HBSS approach addresses this issue by performing "informed stochastic sampling". The HBSS algorithm encompasses a wide spectrum of search techniques that incorporate some mixture of heuristic search and stochastic sampling.

The reported experiments showed that, for the example problem instance, HBSS was able to outperform greedy search within a small number of samples. Of the six bias functions examined, the best performers were the exponential and the 2nd degree polynomial; the two weaker bias functions (logarithmic and linear) veered too far from the heuristic's advice and the two stronger bias functions heeded the advice too often. The experiments reported here are only a beginning; other empirical investigations have been carried out and more are planned for the future. We have collected data on depth-sensitive bias functions, e.g., one that uses an exponential bias until some depth cutoff and then uses a greedy bias to complete the search; but further work is required in this area. We have performed some experiments with other attribute weights in the heuristic,

[3]In some cases, infeasible solutions (i.e., ones that violate constraints) are also included, e.g., as in the *min-conflicts strategy* of (Minton *et al.* 1990).

[4]GSAT solves satisfiability problems, and this encoding assumes that only those neighbors that modify a variable in an unsatisfied clause are considered for selection. If this restriction is dropped, then the algorithm becomes a *mixed random noise strategy* (Selman, Kautz, & Cohen 1994).

and intend to further investigate the relationship between heuristic accuracy and bias strength with regard to HBSS performance.

We are currently implementing a refinement of the HBSS algorithm that takes into consideration the discriminatory power of the given search heuristic. Instead of assigning a unique rank to each alternative choice, the alternatives are grouped into equivalence classes and all members of a class are assigned the same rank. An example of this approach is to specify a heuristic discrimination threshold and to assign the top rank to all alternatives that score within the threshold of the best score; this process would then be repeated using the best score of the remaining alternatives. We are investigating this as well as other methods for deriving the equivalence classes.

Although heuristic-biased stochastic sampling does not obviate the knowledge engineering task of devising a good search heuristic, it does reduce the negative impact of the heuristic's inaccuracies and allows the domain expert's confidence in that heuristic to be expressed and used in the search. When constructing a search heuristic, one tries to achieve the proper balance between heuristic accuracy and the cost of evaluating the heuristic; the HBSS approach adds another parameter to this cost-benefit equation. It can be more cost-effective to use a heuristic that is less accurate, but less expensive to evaluate, as long as the extra computation time spent sampling is less than the time saved in heuristic evaluation. When constructing a search heuristic for a varied problem class, one is sometimes forced to choose the option of performing very well on some of the problems at the expense of performing very poorly on others or to choose the option of mediocre performance on all the problems. In such cases, the HBSS approach can offer a better option than either choice; with the proper degree of exploration, high quality performance can be achieved on many more problems with a fixed heuristic and bias function. Another interesting avenue for future research is *adaptive* bias functions. That is, during the sampling process, a current estimate of heuristic accuracy is computed and used to automatically adjust the strength of the bias.

Acknowledgments: Thanks much to my observation scheduling colleagues: M. Drummond, K. Swanson, W. Edgington, E. Drascher, and G. Henry. Thanks also to the readers of previous drafts: P. Cheeseman, W. Edgington, L. Getoor, O. Hansson, R. Levinson, R. Morris, B. Pell, K. Swanson, and S. Wolfe.

References

Basin, D.A., & Walsh, T. 1993. Difference Unification. *Proc. of IJCAI*, Chambéry, France. Morgan Kaufmann Publishers.

Boyd, L., Epand D., Bresina J., Drummond M., Swanson K., Crawford D., Genet D., Genet R., Henry G., McCook G., Neely W., Schmidtke P., Smith D., & Trublood M. 1993. Automatic Telescope Instruction Set 1993. In *I.A.P.P.P. Comm.*, No. 52. Oswalt (ed).

Bresina, J., Drummond, M., & Swanson, K. 1995. Expected Solution Quality. *Proc. of IJCAI*, Montreal, Canada. Morgan Kaufmann Publishers.

Bresina J., Drummond M., Swanson K., & Edgington W. 1994. Automated Management and Scheduling of Remote Automatic Telescopes. In *Optical Astronomy from the Earth and Moon*, ASP Conference Series, Vol. 55. D.M. Pyper & R.J. Angione (eds.).

Drummond, M., Bresina, J., Edgington, W., Swanson, K., Henry, G., & Drascher, E. 1995. Flexible Scheduling of Automatic Telescopes over the Internet. *Robotic Telescopes: Current Capabilities, Present Developments, and Future Prospects for Automated Astronomy*. G.W. Henry & J.A. Eaton (eds). Astronomical Society of the Pacific, Provo, UT.

Hansson, O. & Mayer, A. 1989. Heuristic search as evidential reasoning. *Proc. of the* 5^{th} *Workshop on Uncertainty in A.I.*, Windsor, Ontario.

Harvey, W.D. 1995. Nonsystematic backtrack search. Ph.D. dissertation, Comp. Sci. Dept., Stanford Univ.

Henry, G.W., 1995. ATIS dispatch scheduling of robotic telescopes. *Robotic Telescopes: Current Capabilities, Present Developments, and Future Prospects for Automated Astronomy*, ASP Conf. Ser. No. 79. G.W. Henry & J.A. Eaton (eds).

Kirkpatrick, S., Gelatt, C.D., & Vecchi, M.P. 1983. Optimization by simulated annealing. *Science*, **220**.

Knuth, D.E. 1975. Estimating the Efficiency of Backtrack Programs. *Mathematics of Computation*, **29**.

Langley, P. 1992. Systematic and Non-Systematic Search. *Proc. of AIPS*, College Park, MD. Morgan Kaufmann Publishers.

Minton S., Bresina J., & Drummond M. 1994. Total-Order and Partial-Order Planning: A Comparative Analysis. *Journal of AI Research*, **2**. AI Access Foundation & Morgan Kaufmann Publishers.

Minton, S., Johnston, M., Philips, A., & Laird, P. 1990. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. *Proc. of AAAI*, Boston. AAAI Press / MIT Press.

Selman, B., Levesque, H., & Mitchell, D. 1992. A new method for solving hard satisfiability problems. *Proc. of AAAI*, San Jose, CA. AAAI Press / MIT Press.

Selman, B. & Kautz, H. 1993. Domain-independent extensions to GSAT: solving large structured satisfiability problems. *Proc. of IJCAI*, Chambéry, France. Morgan Kaufmann Publishers.

Selman, B., Kautz, H., & Cohen, B. 1994. Noise strategies for improving local search. *Proc. of AAAI*, Seattle, WA. AAAI Press / MIT Press.