USING *A PRIORI* KNOWLEDGE TO CREATE PROBABILISTIC MODELS FOR OPTIMIZATION

Shumeet Baluja[1]
School of Computer Science
Carnegie Mellon University

Abstract

Recent studies have examined the effectiveness of using probabilistic models to guide the sample generation process for searching high dimensional spaces. Although the simplest models, which do not account for parameter interdependencies, often perform well on many problems, they may perform poorly when used on problems that have a high degree of interdependence between parameters. More complex dependency networks that can account for the interactions between parameters are required. However, building these networks may necessitate enormous amounts of sampling. In this paper, we demonstrate how *a priori* knowledge of parameter dependencies, even incomplete knowledge, can be incorporated to efficiently obtain accurate models that account for parameter interdependencies. This is achieved by effectively putting priors on the network structures that are created. These more accurate models yield improved results when used to guide the sample generation process for search and also when used to initialize the starting points of other search algorithms.

1 Introduction

Within the past few years, there has been increased interest in using probabilistic modeling for combinatorial optimization. Unlike hillclimbing methods, which operate by sampling solutions neighboring the current solution, probabilistic methods explicitly maintain statistics about the search space by creating models of the good solutions found so far. These models are sampled to generate the next query points to be evaluated. The high-performing sampled solutions are then used to update the model, and the cycle is continued. Comprehensive survey papers of this literature are available [28][31][39][40][41].

A closely related optimization procedure is the genetic algorithm (GA). By maintaining a population of points, GAs can be viewed as creating *implicit* probabilistic models of the solutions seen in the search. In many standard GAs, new sampling points are generated by applying randomized recombination operators to two or more of the high-performance members of a population [10][15][23]. These recombination operators, such as one-point,

1. Shumeet Baluja is currently with JAMDAT Mobile Inc., Los Angeles, CA.

two-point or uniform crossover, randomly select non-overlapping subsets of two “parent” solutions to place into “children” solution. By using a crossover operator that preserves groups of parameters from parent to children strings, GAs attempt to implicitly capture dependencies between the parameters. The randomization of crossover is necessary because no information about which parameter interdependencies are important is explicitly maintained. Therefore, when combining two very different solutions, numerous crossover operations may be required before a useful child solution is produced.

One of the first steps towards making the GA’s probabilistic model more explicit was the “Bit-Based Simulated Crossover (BSC)” operator [44]. Instead of combining pairs of solutions, population-level statistics were used to generate new solutions. The BSC operator works as follows: for each bit position[2], the number of members that contain a one in that bit position is counted. Each member’s contribution is weighted by its fitness with respect to the target optimization function. The same process is used to count the number of zeros. Instead of using pair-wise crossover operators to generate new solutions, BSC generates new query points by stochastically assigning each bit’s value with the probability of having seen that value in the previous population (the value specified by the weighted count).

The ideas incorporated into Population-Based Incremental Learning (PBIL) [1] were similar to those used in BSC. While BSC used a population of solutions from which the sampling statistics were entirely rederived after each generation, PBIL incrementally adjusts its sampling statistics after each generation. PBIL is similar to a cooperative system of discrete learning automata in which the automata choose their actions independently, but all automata receive a common reinforcement dependent upon all their actions [45]. Unlike most previous studies of learning automata, which have commonly addressed optimization in noisy but very small environments, PBIL was used to explore large deterministic spaces. The algorithm maintains a real-valued probability vector from which solutions are generated. As search progresses, the values in the probability vector are gradually shifted to represent high-evaluation solution vectors.

Note that the probabilistic model created in PBIL and BSC is extremely simple: *there are no inter-parameter dependencies modeled; each bit is generated independently.* Although this simple probabilistic model was used, PBIL was successful when compared to a variety of standard genetic algorithms and hillclimbing algorithms on numerous benchmark and real-world problems [2][3][18]. A more theoretical analysis of PBIL can be found in [16][17][22][25][27]. An analysis of PBIL in the Univariate Marginal Distribution framework is given in [33]. Limitations to the PBIL algorithm were described in [12]: PBIL and BSC may not perform as well as pair-wise operators when tested on problems explicitly designed with a high degree of interdependence between parameters.

More complex models in the form of probabilistic networks were introduced to overcome the limitations of models that assumed each parameter was independent. Although these models provided a more accurate representation of the high evaluation solutions, they also required more samples to be used effectively. To reduce the amount of required data, studies were conducted with networks that modeled only a subset of the possible dependencies

2. Note that in this paper, we will discuss optimization with the solutions represented as binary vectors. However, the use of probabilistic models for optimization has been extended to continuous search spaces, for examples see [7][14][29][42].

[4][5][11].

In this paper, we show how *a priori* knowledge of the problem or of the search space can be used to direct the creation of the probabilistic networks. The interactions of variables in the objective function can be more accurately ascertained from the sampled points if knowledge of the problem is used. This helps to overcome the drawbacks of limited sample sizes by ensuring that the modeled dependencies are reflective of real dependencies in the problem and not merely spurious correlations in the sampled solutions. We demonstrate empirically that by creating more accurate models, we improve the quality of the final solutions found through the search.

In the next section, we review the PBIL algorithm. We also show how a simple probabilistic model can be extended to capture dependencies; this work was originally presented in [4]. Section 3 gives an introduction to how *a priori* knowledge can be incorporated into model creation. Section 4 empirically demonstrates the effectiveness on a set of four problems. Section 4 also shows the effectiveness of incorporating knowledge into the networks that are used as "wrappers" to initialize the starting points of faster search algorithms. Finally, Section 5 closes the paper with conclusions and suggestions for future work.

2 Probabilistic Models

In this section, we review the basic PBIL algorithm and show how it can be extended to incorporate models to capture parameter interdependencies.

2.1 Basic PBIL Framework

PBIL employs a simple probabilistic model to independently track the distributions of the bits in the high evaluation solutions. In each generation, only the samples with the best evaluations contribute to the next generation's population; the remaining members are discarded [1][2]. This is akin to truncation selection in genetic algorithms.

The algorithm works as follows: candidate solutions are generated by sampling a real-valued vector, **P**. **P** specifies the probability of generating a 1 in each bit position. A number of solution vectors are generated by stochastically sampling **P**; each bit is sampled independently. The probability vector is then moved towards the solution vectors for which the evaluation function returns the best values, according to Equation 1. The update rule is similar to those used in unsupervised competitive learning [21].

$$P_{t+1,i} = (1-\alpha) \cdot P_{t,i} + \alpha \cdot BestSolutionVector_i \quad (1)$$

$P_{t,i}$ is the value of the probability vector at time t, for parameter i. $BestSolutionVector_i$ is the value of parameter i in the vector being used to update the probability vector. α is a learning rate parameter that determines how much each new datapoint changes the value of the probability vector. The basic version of the PBIL algorithm is shown in Figure 1. The final result of the PBIL algorithm is the best solution generated throughout the search. Numerous extensions, such as those commonly used with genetic algorithms, are possible.

To visually demonstrate how the PBIL algorithm works, we examine the values in the

```
Initialization.               for i :=1 to LENGTH do P[i] := 0.5;

Generate Samples.             while (NOT termination condition)
                                  for i :=1 to K do
                                      solution_vectors[i] := generate_vector_with_probabilities (P);
                                      evaluations[i] :=Evaluate_Solution (solution_vectors[i]);

Sort Vectors according            best_solution_vectors =
to evaluations.                       sort_solutions_from_best_to_worst (solution_vectors,evaluations);

Update Towards Best               for i := M downto 1 do
vectors.                              for j :=1 to LENGTH do
                                          P[j] := P[j] * (1.0 - α) + best_solution_vectors[i][j]* (α);


Terminate Search - return     Return the best solution generated throughout the entire search.
best ever found
```

PBIL CONSTANTS:
K: # of vectors generated before update of the probability vector (200).
α: the learning rate, how fast to exploit the search performed (0.95).
M: number of vectors in the population that are used to update P (1-2).
LENGTH: # of bits in the solution encoding (problem dependent).

Figure 1: Basic PBIL algorithm for a binary alphabet. Values in parentheses are typical settings for the parameters.

probability vector through multiple generations. Consider the following maximization problem: 1.0/|(366503875925-X)|, where $0 \leq X < 2^{40}$. Note that 366503875925 is represented in binary as a string of 20 pairs of alternating ‘01’. The values in the probability vector over time are shown in Figure 2. Note that the most significant bits are pinned to either 0 or 1 very quickly, while the least significant bits are pinned last. This is because during the early portions of the search, the most significant bits yield more information about high-evaluation regions of the search space than the least significant bits.

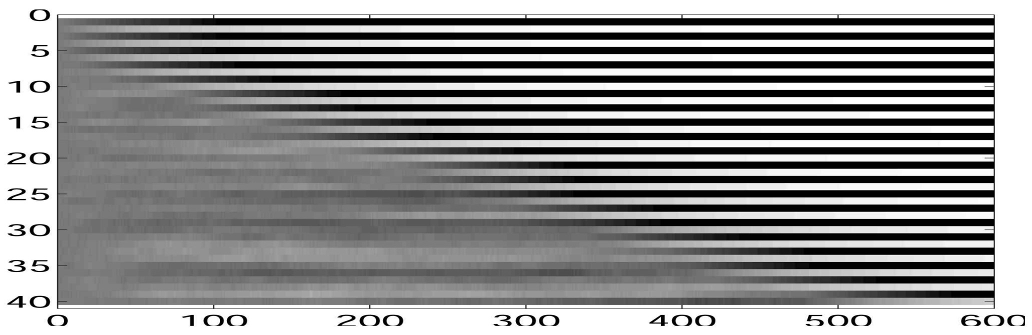


Figure 2: The evolution of the probability vector, ***P***, in the PBIL algorithm. The X-axis is the generation number, the Y-axis is the bit position. White represents a high probability of generating a 1, black represents a high probability of generating a 0. Intermediate grey represents probabilities close to 0.5 - equal chances of generating a 0 or 1. Bit 0 is the most significant, bit 40 the least.

2.2 Modeling Dependencies

In genetic algorithms, the crossover operator is used as an attempt to combine “building blocks” from two different solutions into a new “child” solution. It is clear, however, that neither PBIL nor BSC propagate building blocks in a manner similar to standard GAs, since all parameters are examined independently. Probabilistic models attempt to capture dependencies, or more specifically mutual information, between the parameters to determine which parameters are dependent upon each other. These dependencies are used to generate the new candidate solutions. In the remainder of this section, we will look at how probabilistic models can be automatically created from the sampled points and then used for candidate generation. In Section 3, we examine how *a priori* information about the parameters can be incorporated into the models. The reader is referred to texts by Pearl [38] and Jensen [24] for an introduction to probabilistic modeling and Bayesian networks.

2.2.1. Algorithm Basics

The overall structure of the algorithm is similar to PBIL. After evaluating each member of the current generation, the best members of that population are used to update a probabilistic model from which the next generation's population will be generated.

From the set of solutions evaluated in each generation, the best samples are added into a dataset, termed **S**. Rather than recording the individual members of **S**, our algorithm maintains a sufficient set of statistics in an array ***A***. For models that use pair-wise interactions, this contains a number $\boldsymbol{A}[X_i=a, X_j=b]$ for every pair of variables X_i and X_j and every combination of binary assignments to *a* and *b*. $\boldsymbol{A}[X_i=a, X_j=b]$ is as an estimate of how many recently generated "good" bit strings (from **S**) have bit $X_i=a$ and bit $X_j=b$. To give more weight to recently generated bit-strings, the contributions of bitstrings that were previously added to the dataset are decayed. All $\boldsymbol{A}[X_i=a, X_j=b]$ are initialized to some constant C_{init} before the first iteration of the algorithm; this causes the algorithm's first set of bit-strings to be generated from the uniform distribution. See Figure 3.

```
For all bits i and j and all binary assignments to a and b, initialize $\boldsymbol{A}[X_i=a, X_j=b]$ to $C_{init}$.

Repeat until Termination Condition is Met
    1. Generate probabilistic model based on A. See Figure 4.
    2. Stochastically generate K bit-strings based on the probabilistic model. Evaluate these bit-strings.
    3. Multiply all of the entries in A by a decay factor $\alpha$ between 0 and 1.
    4. Choose the best M of the K bit-strings generated in step 2.
       For each bit-string V of these M, add 1.0 to every $\boldsymbol{A}[X_i=a, X_j=b]$ such that V has $X_i=a$ and $X_j=b$.
```

CONSTANTS (Values used in this study)
C_{init}: Constant used to initialize matrix **A** - Number of examples "seen" at initialization (1000).
K: Number of samples generated in each iteration. This is the population size (200).
M: Number of best samples (from the K generated) that are used to update the statistics (4).
α: How much to decay the effect of older examples (0.99).

Figure 3: Outline for using a probabilistic model. The values in parenthesis are those that will be used in the experiments presented later in this paper.

The values of $\boldsymbol{A}[X_i=a, X_j=b]$ at the beginning of an iteration may be thought of as specifying a prior probability distribution over "good" bit-strings: the ratios of the values within

A$[X_i=a,\ X_j=b]$ specify the distribution, while the magnitudes of these values, multiplied by α, specify an "equivalent sample size" reflecting how confident we are that this prior probability distribution is accurate.

Like PBIL, we only select the top members of the population to contribute to the probabilistic model. Although arbitrarily complex probabilistic models can be used, we use a simple one that is capable of capturing pair-wise dependencies: optimal dependency trees.

2.2.2. Dependency Trees

Given a dataset, **S**, of previously generated good bitstrings, we try to model a probability distribution P(**X**) = $P(X_1, ..., X_n)$ of bit-strings of length **n**, where $X_1, ..., X_n$ are variables corresponding to the values of the bits. We try to learn a simplified model $P'(X_1, ..., X_n)$ of the empirical probability distribution $P(X_1, ..., X_n)$ entailed by the bitstrings in **S.** We restrict our model $P'(X_1, ..., X_n)$ to the following form:

$$P'(X_1 \ldots X_n) = \prod_{i=1}^{n} P\left(X_i \middle| Parent_{X_i}\right) \tag{2}$$

where $Parent_{X_i}$ is X_i's single "parent" variable (the variable on which X_i will be conditioned). We require that there be no cycles in these "parent-of" relationships: formally, there must exist some permutation *m* = *(m*$_1$*, ..., m*$_n$*)* of *(1, ..., n)* such that $(Parent_{X_i} = X_j) \Rightarrow m(i) < m(j)$ for all *i*. (The "root" node, X_R, will not have a parent node; however, this case can be handled with a "dummy" node X_0 such that $P(X_R \mid X_0)$ is by definition equal to $P(X_R)$.) In other words, we restrict P′ to factorizations representable by Bayesian networks in which each node (except X_R) has one parent, *i.e.*, tree-shaped graphs.

A method for finding the optimal model within these restrictions is given in [9]. A complete weighted graph **G** is created in which every variable X_i is represented by a corresponding vertex V_i, and in which the weight W_{ij} for the edge between vertices V_i and V_j is set to the mutual information $I(X_i,X_j)$ between X_i and X_j:

$$I(X_i, X_j) = \sum_{a,b} P(X_i = a, X_j = b) \cdot \log \frac{P(X_i = a, X_j = b)}{P(X_i = a) \cdot P(X_j = b)} \tag{3}$$

The empirical probabilities of the form $P(X_i = a)$ and $P(X_i = a,\ X_j = b)$ are computed directly from **S** for all combinations of *i, j, a,* and *b* (*a* & *b* are binary assignments to X_i & X_j). Once these edge weights are computed, the maximum spanning tree of **G** is calculated, and this tree determines the structure of the network used to model the original probability distribution. Since the edges in **G** are undirected, a decision must be made about the directionality of the dependencies with which to construct P′; however, all such orderings conforming to the restrictions described earlier model identical distributions. Among all trees, this algorithm produces a tree that maximizes:

$$\sum_{i=1}^{n} I(X_{m(i)}, X_{m(p(i))}) \tag{4}$$

which in turn minimizes the Kullback-Leibler divergence, D(P||P′), between P (the true empirical distributions exhibited by **S**) and P′ (the distribution modeled by the network):

$$D(P \parallel P') = \sum_{\boldsymbol{X}} P(\boldsymbol{X}) \log \frac{P(\boldsymbol{X})}{P'(\boldsymbol{X})} \tag{5}$$

As shown in [9], this produces the tree-shaped network that maximizes the likelihood of **S** (this means that of all the tree shaped networks, this is the most likely to have generated **S**). This tree generation algorithm, summarized in Figure 4, runs in time ***O(n²)***, where **n** is the number of bits in the solution encoding.

```
Generate an optimal dependency tree:
  • Set the root to an arbitrary bit $X_{root}$
  • For all other bits $X_i$, set bestMatchingBitInTree[$X_i$] to $X_{root}$.
  • While not all bits have been added to the tree:
     • Of all the bits not yet in the tree, pick bit $X_{add}$ with the maximum
          mutual information I($X_{add}$, bestMatchingBitInTree[$X_{add}$]),
          using A (which contain sufficient statistics for S) to
          estimate the relevant probability distributions.
     • Add $X_{add}$ to tree, with bestMatchingBitInTree[$X_{add}$] as parent.
     • For each bit $X_{out}$ not in the tree,
          if I($X_{out}$, bestMatchingBitInTree[$X_{out}$]) < I($X_{out}$, $X_{add}$).
            then set bestMatchingBitInTree[$X_{out}$]=$X_{add}$.
```

Figure 4: Detailed procedure for generating the dependency tree.

The arcs which remain in the maximum spanning tree represent the dependencies to be modeled. Since it is a tree, each variable will be conditioned on exactly one other variable (its parent). The exception to this is the root of the tree, which is set according to its *un*conditional probabilities. Once we have generated a dependency tree modeling $P(X_1, ..., X_n)$, we use it to generate **K** new bitstrings. Each bitstring is generated in ***O(n)*** time during a depth-first traversal of the tree. Each bitstring is then evaluated. The best **M** of these bitstrings are selected and effectively added to **S** by updating the counts in **A**. Based on the updated **A**, a new dependency tree is created, and the cycle is continued.

2.3 Discussion of Related Models

Another extension to PBIL that captured pair-wise dependencies was termed *Mutual Information Maximization for Input Clustering (MIMIC)* [11]. MIMIC used a greedy search to generate a chain in which each variable is conditioned on the previous variable. The first variable in the chain, X_1, is chosen to be the variable with the lowest unconditional entropy $H(X_1)$. When deciding which subsequent variable X_{i+1} to add to the chain, MIMIC selects the variable with the lowest conditional entropy $H(X_{i+1} \mid X_i)$. While MIMIC was restricted to a greedy heuristic for finding chain-based models, the algorithm described in this paper uses a broader class of models, trees, and finds the optimal model in the class.

Example dependency graphs shown in Figure 5 illustrate the types of probability models learned by PBIL, a dependency-chain algorithm similar to MIMIC, and our dependency tree algorithm. We use Bayesian network notation for our graphs: an arrow from node X_p to node X_c indicates that X_c's probability distribution is conditionally dependent on the value of X_p. These models were learned while optimizing a noisy version of a two-color graph coloring problem (shown in Figure 5a) in which there is a 0.5 probability of adding 1 to the evaluation function for every edge constraint satisfied by the candidate solution. Note that the dependency tree algorithm is able to discover the underlying structure of the graph, in terms of which bits are dependent on each other (as shown in Figure 5D).

The clear next step after modeling pair-wise dependencies is modeling higher-order dependencies. The need for this has been demonstrated in [6]. However, generating models which are capable of representing higher order dependencies may be computationally expensive. The hope is that the expense of generating the models will be offset by the savings obtained by the smaller number of function evaluations that will be required due to the more accurate modeling. A large amount of work has been done exploring different models to use. The Factorized Distribution Algorithm (FDA) [32][33][34][35] uses a fixed model throughout the search, with the model being specified by an expert. The FDA algorithm is designed to work with problems that are decomposable into independent parts. This work has been extended to incorporate learning with low complexity networks and Junction-Trees [36][37]. The Bayesian Optimization Algorithm (BOA) and related work [13][30][39][40] is the closest method to the optimization techniques presented here. The model used in BOA is able to represent arbitrary dependencies. When general Bayesian Networks are used for modeling, the scoring function used to determine the quality of the network plays a vital role in finding accurate networks. The quality of networks can be assessed through a variety of measures. For example, both Minimum Description Length and Bayesian Dirichlet metrics have been explored in [40]. The models that are found by

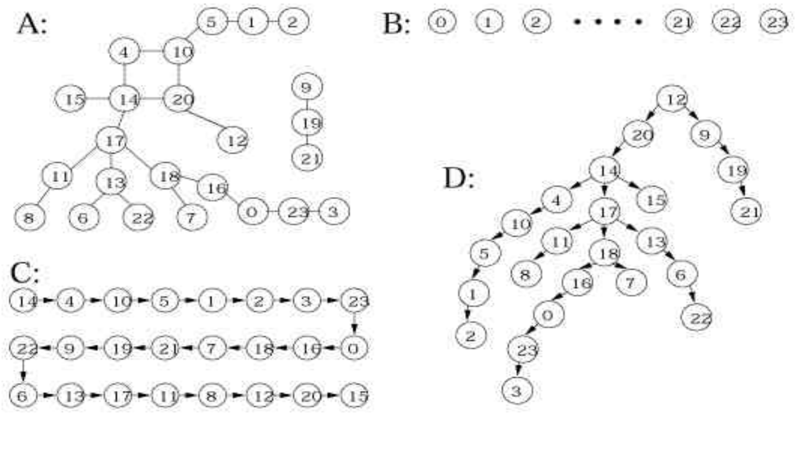


Figure 5: A: The underlying graph in a noisy two-color graph coloring problem. B: the empty dependency graph used by PBIL. C: the graph learned by our implementation of the dependency chain algorithm. D: the graph learned by our dependency tree algorithm.

BOA are similar to those used in FDA; however, BOA is designed to learn the models as the search progresses. Because the models used by BOA are general Bayesian Networks, it is clear that *a priori* information can be incorporated [19][20]. This is the focus of the next section.

3 Incorporating *a priori* Knowledge

In this section, we describe how to incorporate information into the process of learning the probabilistic model. The method is general and can be used in MIMIC [11], COMIT [5], BOA [39] or any other algorithm which has a learning component in the model generation process. Although the *a priori* information that is available for a problem is often high level and specifies complex dependencies, we show how the knowledge can be used even when simple probabilistic models, those that cannot represent high-order dependencies, are employed.

To this point, we have restricted the probabilistic models that we have examined to dependency trees that only model pair-wise interactions. This was done to mitigate the need for a large number of samples that arises when higher-order dependencies are modeled. Nonetheless, in some cases more complex models are required. Additionally, even when simple models such as trees are used, by using *a priori* information to constrain the number of possible trees that are considered, the samples can be more effectively used since they must only select trees from a reduced set. In this paper, we use *a priori* knowledge about the function to be optimized to constrain the arcs that are modeled in the probabilistic models. This technique is applicable to optimization procedures regardless of whether a multiply connected Bayesian network is used or a simple dependency tree is employed.

As an introductory example, the potential for using *a priori* knowledge is clearly demonstrated in problems in which the dependencies are evident, such as graph coloring. Consider the graph coloring problem as shown in Figure 5A. In this simple problem, it is clear that the color of node 5 should be dependent upon the color of node 1 and 10, and that the color of node 10 should be dependent on the colors of nodes 4, 5, and 20. There are several ways to incorporate this information into a probabilistic model. The first is to employ a model that captures more than pair-wise dependencies. For example, if we allowed arbitrary dependencies, we could create models with more than a single parent; thereby mimicing the graph structure shown in Figure 5A. Although this would require maintaining more than pair-wise statistics, only a subset of these higher-order statistics would be required since we could specify the dependencies to be modeled from our knowledge of the underlying graph structure. The second approach is to select the model from a family of low complexity models (such as the set of all trees - as we have described in Section 2.2) but to allow the arcs to be selected only from the subset of those that exist in the graph. Continuing with the same example, the allowed parents for node 5 would be either node 10 or node 1, but not both (since that would violate the tree property).

Throughout the remainder of this paper, we concentrate on the second approach described above: constraining the tree structures that can be created. This approach has the benefit of not requiring exact information of which dependencies must be modeled; although we do not specify the specific parents of each of the nodes, we restrict the possibilities. This method also has benefits as problem sizes increase. Modeling higher order dependencies, even when the structure of the network is known, requires a large number of samples. In the graph coloring task, this problem becomes significant when the connectivity of the graph increases.

One of many ways to implement constraints on the dependency graph is to impose a prior over network structures in which the prior likelihood of a network decreases exponentially with the number of arcs in the network that do not correspond to edges in a pre-specified set. With the optimal dependency trees, such a prior can be simply implemented. We need only subtract a penalty term from the mutual information between any two variables that are not connected by an edge (***E***) in the pre-specified preferred set (***S***) and run the maximum spanning tree algorithm on these modified weights instead. The modified mutual information calculation is shown in Eq [6].

$$I'(X_i, X_j) = \begin{cases} if(E_{i,j} \in S) \\ \quad \sum_{a,b} P(X_i = a, X_j = b) \cdot \log \frac{P(X_i = a, X_j = b)}{P(X_i = a) \cdot P(X_j = b)} \\ if(E_{i,j} \notin S) \\ \quad \left(\sum_{a,b} P(X_i = a, X_j = b) \cdot \log \frac{P(X_i = a, X_j = b)}{P(X_i = a) \cdot P(X_j = b)}\right) - \alpha_{i,j} \end{cases} \tag{6}$$

As shown above, the penalty, α, does not need to be constant, and can vary per dependency arc. The severity of the penalty provides a means to convey confidence in the *a priori* information. The more confident we are that an arc should not be modeled, the larger the penalty can be.

For simplicity, however, we do not use a complex penalty setting. Instead, the penalty term is constant for every arc. In the experiments presented in this paper, a sufficiently severe penalty term was used to ensure that arcs in the pre-specified set were *always* favored over arcs not in the set. This simple penalty procedure was chosen to ensure that the focus of the experiments remain on demonstrating that improvements in the final search result were obtainable by incorporating *a priori* information into the probabilistic models. Nonetheless, we do not suggest that this will work well on all problems; in problems in which the information should be regarded only as a preferred dependency instead of one that must be enforced, a less severe penalty may yield improved results.

4 Empirical Results

This section is divided into two parts. In the first, Section 4.1, two problems sets are examined. The first is a simple numerical optimization problem, termed “Summation Cancellation”, and the second is a set of graph coloring problems. For each of these problems, we show that by incorporating knowledge of parameter dependencies, more accurate models can be built and subsequently, better results obtained.

In Section 4.2, we show how *a priori* knowledge can be incorporated into optimization procedures which use probabilistic modeling techniques to generate the initialization points for specialized/faster search heuristics [4]. In this optimization architecture, the points that are generated from the probabilistic model are used to initialize specialized search heuristics. The best found solutions that are returned at the *completion of the specialized search heuristics* are then inserted into the data set used to generate the next probabilistic model. The probabilistic models effectively "wrap-around" the specialized search heuristics. This contrasts with the approach described to this point where the generated points are fed directly back into the data set for the creation of the next probabilistic model. As will be shown, in both approaches, the need and benefits of incorporating *a priori* information remains the same.

Note that the results in the section are not intended to represent a comparison of different optimization algorithms. For more comprehensive comparisons between optimization methods, such as genetic algorithms, probabilistic optimization methods, and hillclimbing methods, the reader is referred to [2][4][5][12][18]. For the experiments presented in this section, we keep the probabilistic modeling algorithms as simple as possible to concentrate our examination on the effects of incorporating knowledge into the models. We have not included operators such as mutation, local hillclimbing or any of the numerous heuristics that can be used in conjunction with optimization techniques to create a general purpose optimization tool [15][26].

4.1 Proof of Concept

For these tests, the parameters used were exactly as shown in Figure 3. The incorporation of *a priori* knowledge is described with each problem.

4.1.1. Summation Cancellation

In this problem, the parameters (s_1, ..., s_N) in the beginning of the solution string have a large influence on the quality of the solution. The goal is to minimize the magnitudes of cumulative sums of the parameters. Small changes in the first parameters can cause large changes in the evaluation. The evaluation function is defined as the maximization functions shown below:

$$\begin{matrix} -0.16 \le s_i \le 0.15 \\ i = 1 \ldots N \end{matrix} \qquad y_1 = s_1 \qquad \begin{matrix} y_i = s_i + y_{i-1} \\ i = 2 \ldots N \end{matrix} \qquad C = \frac{1}{100000} \qquad f(y_1, \ldots, y_N) = \frac{1.0}{C + \left|\sum_{1}^{N} |y_i|\right|}$$

The solution was encoded in binary and each parameter was assigned 5 bits. There were a total of 50 parameters; therefore, the search space size was 2^{250}.

If the tree-based model is used without any *a priori* knowledge, there are (250*249)/2 dependencies to model. The only dependencies that are not modeled are those from a bit to itself. In terms of edges that may be added to the preferred set, these dependencies are represented by 250*249 edges. There are twice as many edges that may be in the preferred set since we can introduce priors that limit the selection of trees to those that specify the dependencies used for sample generation of variable *a* on variable *b*, but not vice-versa.

As can be seen by the function evaluation, each of the 50 parameters is directly dependent on the parameter that *precedes* it. Therefore, a straightforward method of incorporating knowledge is to ensure that each parameter can only be conditionally dependent on those that come before it. [3]

We can further examine the effects of incorporating *a priori* knowledge by varying the number of preceding parameters that the sample generation process can be based upon. We examine the effects of letting a variable be conditionally dependent on 100% of the previous parameters, as well as smaller percentages. The final experiment we perform is to examine the effect of allowing dependencies on only the single preceding parameter. To better illustrate the allowed dependencies, they are shown graphically in Figure 6.

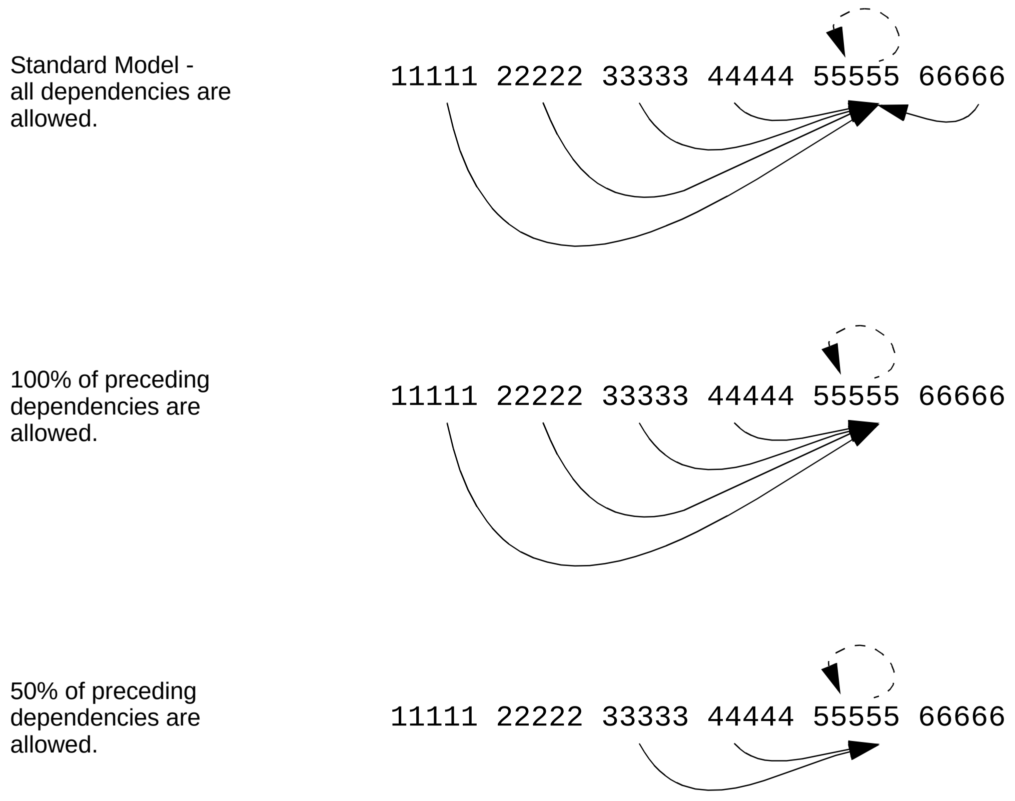

Standard Model - all dependencies are allowed.
11111 22222 33333 44444 55555 66666
100% of preceding dependencies are allowed.
11111 22222 33333 44444 55555 66666
50% of preceding dependencies are allowed.
11111 22222 33333 44444 55555 66666

Figure 6: The arcs show the sets of dependencies allowed for the bits in parameter 5. Each parameter is represented by 5 bits (e.g. 22222). Each arrow represents potential dependencies between any of the bits in the parameters pointed to. No single bit can be dependent upon itself, although a bit within a parameter can be dependent on another bit in the same parameter. Each solid line represents 25 edges, each dashed line represents 20 edges.

3. Note that because each of the 50 parameters is actually represented with 5 bits, we allow the dependency tree to have bits in a single parameter be dependent on another bit in the same parameter. This is represented graphically in Figure 6, where the arrow above parameter 5 points to itself.

The results of the experiments are shown in Figure 7; there were 20 trials conducted per dependency setting. From Figure 7, note that the trials conducted with no *a priori* information (full dependencies allowed) achieved an average evaluation of 0.543. Models in which only the preceding dependencies were allowed achieved scores of approximately 0.660-0.667. Using a standard *t*-test, the differences in results between using "Full Dependencies" and all of the "Preceding Dependencies" experiments are statistically significantly different to the p=0.01 level. There was no significant differences between allowing 100% and 20% of preceding parameters to be modeled. When only the single preceding parameter was allowed, the results jumped to 0.696. The differences between the "Preceding 100%" and "Single Preceding" trials are significant to the p=0.02 level. All of these significance tests were repeated using a non-parametric equivalent to the standard *t*-test, the Mann-Whitney test; the same results were obtained.

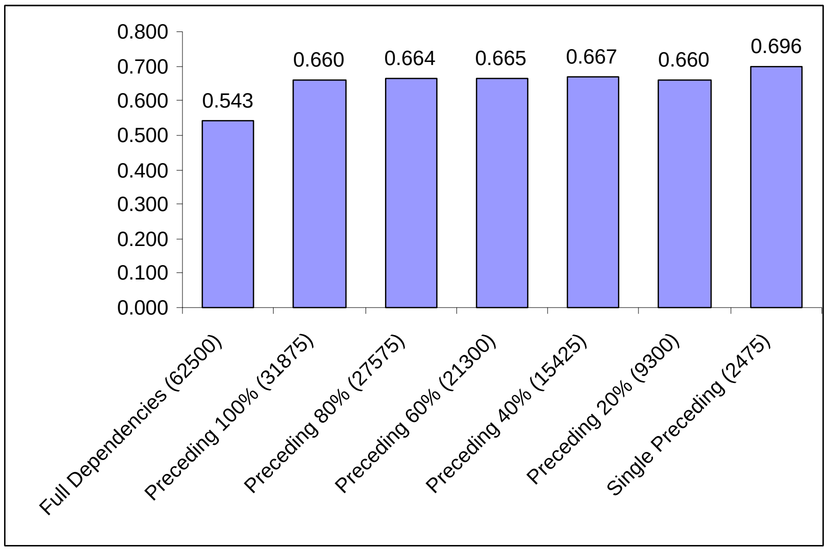


Figure 7: Results for the Summation Cancellation problems. Larger numbers are better. Note that using any *a priori* knowledge (through using dependencies only with preceding parameters) statistically significantly improves the results over not incorporating *a priori* knowledge. The number in parentheses is the number of dependencies that *do not* have a penalty applied to them and are therefore in the preferred candidates set for edges in the dependency tree.

4.1.2. Graph Coloring

In each of these problems, there are 750 nodes in a partially connected graph to be assigned one of two colors. The goal is to assign each of the connected nodes opposite colors. In these problems, the graphs are not planar, and a solution in which all of the constraints are met is not guaranteed to exist. The evaluation, to be maximized, is the number of constraints (connected nodes that have the opposite color) that are met.

To examine the effects of problem complexity, three sets of problems are examined. In the first set, each node is connected, on average, to 4 other nodes. In the second set, each node is connected to 10 nodes. In the last set, the average connectivity is 20 nodes.

The results are shown in Table I. For each set of problems, we examine three cases. When the amount of information used is 0.0, no penalties are given during the mutual information calculation. For the experiments labeled "100% *a priori* information", when the mutual information is calculated for two nodes which are not connected by an edge in the problem (in the pre-selected set of edges), a penalty is applied. For the experiments labeled "50% *a priori* information", rather than applying a penalty to all of the edges not in the graph, a penalty is applied to a randomly selected 50% of them. This effectively incorporates less information into the optimization process because it lessens the restrictions on the probabilistic models which can be created. This case is included to determine whether providing hints to the network is beneficial, even if perfect information is unavailable.

Table I: Results for the Graph Coloring Problems

	Mean Result	Standard Deviation
Graphs with 4 Connections		
No *a priori* Information	1140	14.0
50% *a priori* Information	1160	9.1
100% *a priori* Information	1214	9.8
Graphs with 10 Connections		
No *a priori* Information	2397	17.6
50% *a priori* Information	2406	39.6
100% *a priori* Information	2485	19.4
Graphs with 20 Connections		
No *a priori* Information	4449	14.7
50% *a priori* Information	4416	28.3
100% *a priori* Information	4549	14.9

In every case, the results with the "100% *a priori*" information are statistically significantly different than the "no-*a priori*" knowledge results to the p=0.01 confidence level. The differences between the "50% *a priori*" results and the "no-*a priori*" information results are not always significantly different, as can be seen by the relatively large standard deviations in the "50% *a priori*" results. The results are statistically different to the p=0.01 confidence level between the "100% *a priori*" results and the "50% *a priori*" results. The use of *a priori* information can significantly improve performance on these problems. As suspected, the less specific the information, the less improvement is seen, on average.

It is important to note that in these experiments the connectivity of the graphs increased in each of the trials. In the cases in which the connectivity was on average 20 connections, the predefined set of edges that were allowed to be in the dependency graph included on average 20 parents for each node. Because the dependency model was a tree, from this set, only a *single* node was chosen to be the parent. Note that we did not need to *a priori* specify the exact parent of each node; just by limiting the set of possible parents we improved the performance of the search.

4.2 Using *a priori* Knowledge with "Wrapper" approaches to Optimization

If we are trying to solve a well studied optimization problem, for example Satisfiability, Bin Packing, Jobshop, VLSI Layout, etc., there already exist a wide variety of specialized stochastic heuristics to address these problems. We would like to be able to use the probabilistic modeling techniques in conjunction with these specialized procedures.

Instead of using the pure probabilistic modeling techniques as described earlier, we employ the probabilistic models to generate points with which to initialize search with the specialized heuristics. For example, once we make several runs with the specialized heuristic, we can model the better points found during these runs with the probabilistic models. These probabilistic models can then be used to generate new initialization points. In this manner, the probabilistic models "wrap-around" the specialized search heuristics[4]. The probabilistic models are created based upon the good solutions that have been found in the previous specialized-search runs. Similar to the manner in which *a priori* information was incorporated into the probabilistic models in the Section 4.1, we can incorporate any *a priori* information in the models created in this approach. The resulting algorithm is shown in Figure 8.

Note that the wrapper approach may also be employed for computational benefits, irrespective of whether a *specialized* search algorithm is used. For example, as described in [5], faster search algorithms such as PBIL, hillclimbing, etc. can be used to provide computational benefits over the "standard" probabilistic modeling-based optimization techniques described in Section 2. In comparison to the standard approach, when using the wrapper approach, the probabilistic model is created much less frequently. This provides enormous benefits in terms of speed since model creation is a computationally intensive procedure (even for the trees, it is $\boldsymbol{O(n^2)}$ where $\boldsymbol{n}$ is the number of bits in the parameter encoding, for more complex networks the expense can be much greater [8]). Instead of creating the model in every generation, the model is created only between runs of the faster search heuristics.

In this section, we demonstrate the wrapper framework with a simple random mutation stochastic hillclimbing (RMSH) as the faster search technique. The hillclimber is *next-ascent*: whenever a new solution is found with a better or equal evaluation, the move is accepted. The probabilistic model is used to combine the best solutions found by the hillclimbing runs. The hillclimbing runs are restarted whenever $\boldsymbol{M}$ evaluations are encoun-

4. Note that sometimes the specialized search heuristics use specialized solution encodings. In these cases, we can convert between this representation and a low-cardinality representation which is more suitable for the probabilistic modeling techniques described here (for example, this is the case for common search heuristics used with the TSP, Jobshop Scheduling, Satisfiability, Binpacking, Knapsack, etc. problems).

```
• Initialize dataset S with random solutions from uniform distribution

• While termination condition is not met:
  • Create a probabilistic network (in this study a tree) T that models S.
        Use priors on the network structure to incorporate a priori information.
  • Use T to stochastically generate K solutions. Evaluate these K new solutions.
  • Start specialized heuristic run initialized with the single best of the K solutions.
  • Replace up to R bitstrings in S with unique, better bitstrings found during the
        specialized-heuristic's run just executed.

USER DEFINED CONSTANTS (values used in this study)
|S|: Constant size of the dataset S (1000).
K: Number of solutions to generate from the probabilistic model (100).
R: max number of bitstrings to replace in S with better strings from a
   single specialized-search run (100).
```

Figure 8: Combining Probabilistic Modeling with Specialized Heuristics.

tered without improving on the current state. Three algorithms are compared:

1. **RMSH-Model-Based-Restart:** this initializes the hillclimbing runs by sampling the probabilistic model that has been created by modeling the best solutions found in previous RMSH runs. Prior to starting each RMSH run, 100 samples are generated from the model, and the best one is used to start the next RMSH run.

2. **RMSH-apriori-Model-Based-Restart:** this initializes the hillclimbing runs by sampling the probabilistic model that has been created by modeling the best solutions found in previous RMSH runs. The probabilistic model also uses the *a priori* information that is available. Prior to starting each RMSH run, 100 samples are generated from the model, and the best one is used to start the next RMSH run.

3. **RMSH-Random-Restart:** this initializes the hillclimbing runs by randomly generating 100 samples and using the best one to start the next hillclimbing run. [5]

Rather than arbitrarily setting ***M***, which could introduce confounding biases into the results, ***M*** was set automatically. ***M*** was chosen by exploring 10 different settings for each algorithm on each problem and selecting the best for each. The settings ranged from (1 * Solution_Encoding_Length) to (10 * Solution_Encoding_Length).

5. Note that in this algorithm 100 random samples are generated instead of a single random point to make this algorithm as parallel in structure to the other two algorithms that are compared.

4.2.1. Constrained Optimization

This problem is composed of ***N*** variables and ***C*** constraints on each variable. Each variable can take on only a small set of values, ***V***. Each constraint specifies the relationship between two variables, ***a*** and ***b,*** i.e. whether ***a*** should be greater than ***b*** or ***b*** should be greater than ***a***. The objective is to maximize the number of constraints that are met.

This problem is formulated as a minimization problem. For each constraint that is not met, a penalty is applied. The sum of the penalties is the evaluation. The penalty applied for each constraint connecting nodes ***a*** and ***b*** is $|a - b|$. The penalty is only applied for constraints that are not satisfied; if the constraint is satisfied it does not contribute to the evaluation. This evaluation gives more information to the search algorithm than only giving only a binary value specifying whether the constraint has been satisfied. In the representation used, the value of each parameter is represented in binary. Each parameter is represented with $\log_2 V$ bits. The total length of the encoding is $N\log_2 V$ bits.

A priori knowledge of the problem was incorporated by examining the graph of the constraints. In this graph, each variable was represented as a vertex and each constraint was a edge that connected two of the vertices. In the case where no *a priori* knowledge was used in the model creation process, every parameter could be dependent on every other parameter; therefore all $(N\log_2 V)^2$ dependencies were modeled (all dependencies were in the preferred set). To incorporate knowledge, we limited the dependencies of each parameter to those parameters that could be reached in ***L*** steps by traversing edges in the graph. We expect the maximum effect on parameters to come from other parameters that are close (in terms of edges in the graph). For small ***L***, this dramatically reduces the number of dependencies that were placed in the preferred set when the average connectivity of the graph is low. For simplicity, we kept ***L*** constant, ***L=1***.

Three problems sets are attempted. For each problem set, we examine 20 problem instances which are randomly generated given the problem parameters shown in Table II. The generated problems may not have constraints that are all simultaneously satisfiable. Rather than presenting the average score of the results we give the ranks of each of the algorithms, as well as the average rank. The algorithm with the least summed penalties has the highest rank, and is therefore the best. Ranks are given to avoid misrepresentations that may arise from averaging the widely differing scores possible with each problem instantiation. For ties, each tied algorithm is given the better of the tie scores. The results are shown in Table II.

In the vast majority of the trials, the **RMSH-apriori-Model-Based-Restart** algorithm performed best. The significance of difference in the ranks was measured by the non-parametric Wilcoxon Matched Pairs Signed-Ranks test. The differences in all the results are statistically significant to p=0.01 level. Note that the number of edges in the preferred set in the **RMSH-apriori-Model-Based-Restart** algorithm were *less than 3%* of the total.

The edges that were members of the preferred set in the above tests were asymmetric between vertices. The edge that was added depended on on how the edge was specified in the problem definition. For example, if a constraint existed that ***a < b,*** then the dependency between the bits representing the value of ***a*** on the bits representing the value of ***b*** were added to the preferred set. Had the edge been specified as ***b > a***, the dependency between the bits representing the value of ***b*** on the bits representing the value of ***a*** would have been

Table II: Numerical Constraint Results: Ranks of RMSH Algorithms on 3 Problem Sets (Lower rank is better)

Problem Set Parameters	Search Heuristic	Average # dependencies in preferred set.	# of trials in which rank = 1	# of trials in which rank = 2	# of trials in which rank = 3	**average rank**
N = 200 Variables |V| = 16 Values C = 5 Constraints / Variable (1000 Total Constraints)	**RMSH-*apriori***	18,208	19	1	0	**1.05**
	RMSH-Model	640,000	1	18	1	**2.00**
	RMSH-random	n/a	0	1	19	**2.95**
Encoding Length = 200 * $\log_2 16$ = 800 bits.						
N = 200 Variables |V| = 64 Values C = 5 Constraints / Variable (1000 Total Constraints)	**RMSH-*apriori***	41,528	18	2	0	**1.10**
	RMSH-Model	1,440,000	2	18	0	**1.90**
	RMSH-random	n/a	0	0	20	**3.00**
Encoding Length = 200 * $\log_2 64$ = 1200 bits.						
N = 400 Variables |V| = 16 Values C = 10 Constraints / Variable (4000 Total Constraints)	**RMSH-*apriori***	68,010	19	1	0	**1.05**
	RMSH-Model	2,560,000	1	19	0	**1.95**
	RMSH-random	n/a	0	0	20	**3.00**
Encoding Length = 400 * $\log_2 16$ = 1600 bits.						

added to the preferred set. For completeness, preliminary experiments were conducted with adding both sets of dependencies. With both sets of dependencies, performance suffered over using only an asymmetric set, although the performance remained improved over using no *a priori* information. Although exploring the detailed use of *a priori* information on this particular problem is beyond the scope of this paper, this underscores the importance of using *a priori* information that works in conjunction with the underlying search technique employed and also emphasizes the sensitivity of using *a priori* information. This presents an interesting avenue for future research.

4.2.2. Traveling Salesman Problem

The encoding used in this study requires a bit string of size $\mathbf{N}\log_2\mathbf{N}$ bits, where N is the number of cities in the problem. Each city is assigned a substring of length $\log_2\mathbf{N}$ bits; the value of these bits determines the order in which the city is visited. See [43] for details. Three problem were attempted: a 100 city problem, and two 150 city problems. The encoding length for the first problem was 700 bits. For the second two problems, the encoding length was 1200 bits.

For the **RMSH-apriori-Model-Based-Restart** trials, *a priori* information was used as follows. In many TSP problems, we may suspect that it is reasonable to assume that the order in which a city appears in a tour will be most dependent upon the nearest ***C%*** of the cities (where $C < 100$). Therefore, dependencies should only be modeled between these cities, and not ones that are further away. For the *a priori* knowledge, we simply put high priors on network structures that have dependency arcs between bits that represent cities that are close. In this way, the allowed dependencies are those that are between cities that are close. Note that this does not require us to know exactly which cities will be used for modeling the dependencies; instead, we are able to provide a set of cities that we think are likely good candidates for modeling.

As in the previous problems, for simplicity, we use sufficiently severe penalties for arcs that we wish to exclude from being in the tree that any arc in the selected set is guaranteed to be selected before those not in the set. In the future, this problem may be a good candidate to examine the effects of more graded penalties - for example, by making the magnitude of the penalty in the arc proportional to the distance of the cities.

The results are shown in Figure 9. In all three of the problems, there was a dramatic decrease in tour size when a probabilistic model was employed; the differences between the searches that did not employ a probabilistic model and those that did were significant to the p=0.01 level. The setting of ***C*** at 60% provided, on average, improved performance in all three problems over using models without *a priori* information. However, the differences were determined not to be statistically significant to the p=0.01 level over using models with no *a priori* information.

Only in problem 3 are the differences between the searches that use *a priori* information and those that do not significant to the p=0.05 level (for ***C***=40%). There is no statistical difference between the different values for ***C*** where ***C*** is set to 20%, 40%, 60% or 80%. It is suspected that the effectiveness of the setting of ***C*** depends on the clustering and layout of the cities; it will be interesting in the future to examine how to account for clustering in setting the ***C*** parameter. As suggested above, it may be possible to avoid this problem altogether by using proportional penalties, or by using other heuristics as the basis of the *a priori* information.

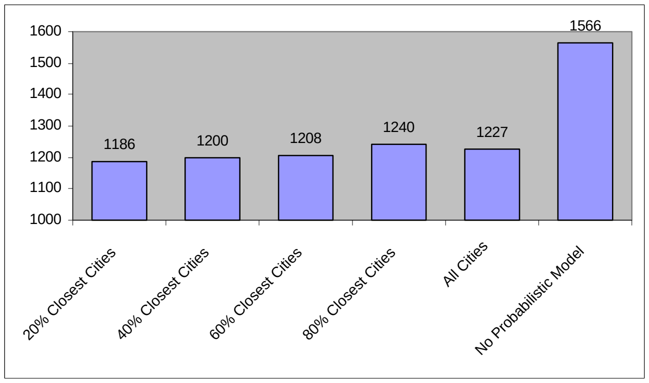
The use of a probabilistic model to select restart points on the TSP problem leads to significant improvements in solution quality in comparison to using simple random restarts. This has also been empirically shown in a variety of other problems [5]. The incorporation of the *a priori* knowledge used here improves the results in some cases and has no effect on others. This is discussed in Section 4.3.

4.3 Discussion of Empirical Results

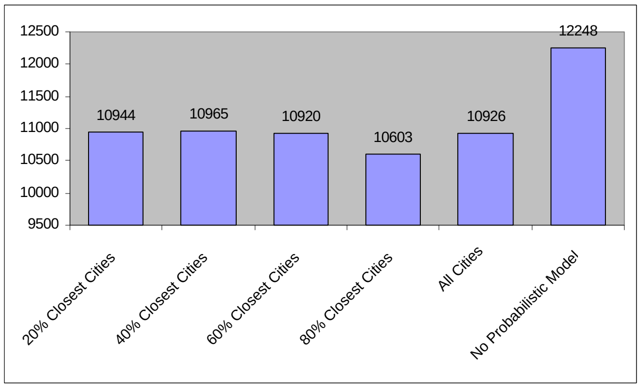
In almost every instance of problems examined, we have demonstrated that it is possible to achieve benefits in the final results obtained through the use of *a priori* knowledge to create the probabilistic models. With respect to using the “wrapper” approach, an important point to note is that the methodology of restarting faster search algorithms with the probabilistic models is applicable to more than RMSH heuristics; any search algorithm can be used within the “wrapper”, such as PBIL, TABU search, standard genetic algorithms or specialized search heuristics.

A necessary step for using the methods presented in this paper is determining what *a priori* information is suitable. In this paper, we selected problems to demonstrate the techniques; all of the problems had straight-forward knowledge that could be incorporated. However, in other problems, it may be difficult for a non-expert to generate appropriate knowledge (*i.e.* bin packing, knapsack, etc.). There are also sets of problems in which it may be impossible to *a priori* narrow down the set of dependencies. For example, consider solving a set of linear equations in which each variable is used in each equation; without more information, it is impossible to tell which parameters are most likely to be dependent on each other. Because of the simplicity of the probabilistic model used, dependency-trees, there also exist problems that the model will not be able to accurately represent

Problem #1:
100 City



Problem #2:
150 City



Problem #3:
150 City

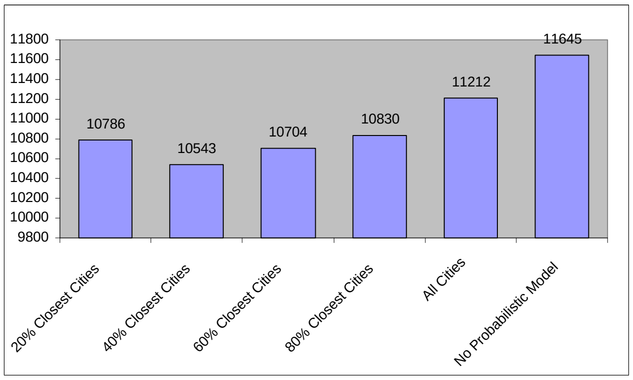


Figure 9: Average tour length. Results of incorporating *a priori* knowledge into the TSP problem. Three problem instantiations are shown. Each bar represents the average tour size found over all trials with the specified setting. Note that when no probabilistic model is used, the results are substantially degraded.

despite the model's incorporation of *a priori* knowledge. For example, in the numerical constraint problems discussed in Section 4.2, when the number of dependencies per node became large, the limitation of one parent per node in the dependency tree will prevent the tree from learning a sufficient number of dependencies to impact the final result. This can be the case regardless of whether or not the set of possible parents was limited to connected nodes through the use of *a priori* knowledge. In these cases, optimization procedures that model higher order dependencies will be useful [34][35][39][40].

The purpose of this paper is to demonstrate the potential for *a priori* knowledge and give examples of how the knowledge may be incorporated. To empirically demonstrate the potential, the problem sizes were chosen to be large enough so that incorporating knowledge had an effect. When the search spaces were too small or too easy, *a priori* information had no effect, beneficial or detrimental, on the search results. In these cases, the dependencies found by the tree even without *a priori* information performed as well as with hints provided by experts. Only as the problems became either complex or large did the *a priori* information become useful. It is also important to recognize the possibility of incorrect hints being detrimental to search. If not enough dependencies are allowed, or if the wrong ones are emphasized, the models created may not be able to overcome the hints. This paper has presented a method for incorporating *a priori* information into models such as trees; however, finding the *a priori* information that is most useful is a necessary step that must be conducted for each individual type of problem that is examined.

5 Conclusions & Future Work

This paper has demonstrated the effectiveness of incorporating *a priori* knowledge into the probabilistic models that are used to guide search. The knowledge was used to direct the arcs that were included in the optimal dependency trees. We have also demonstrated that the knowledge that is included does not need to be exact. In all of the experiments conducted, we limited the set of edges that could be included in the tree; however, the exact tree was never specified - and was automatically generated during the search.

Two broad optimization methods were discussed in this paper. A "standard" approach to using probabilistic models was explored first. In this model, the probabilistic model is sampled and the best of the samples is immediately introduced into the population of points from which the next probabilistic models is generated; the cycle is then continued. In the second method, "the wrapper" approach, after the probabilistic models are sampled, the best samples are used to initialize faster or specialized search algorithms. Upon completion of the specialized search algorithm, the best result obtained is put into the population of points from which the next model is generated. From this model, the next set of points are generated and the cycle is continued. In this paper, we have demonstrated that it is possible to incorporate information into either of these methods; the need for accurate models remains the same.

Although we have not attempted to propose a complete optimization "system" in this paper, there is a vast amount of literature available on heuristics that can be used in conjunction with the algorithms proposed here. Extensions to the algorithms proposed here will include such heuristics as elitist selection, mutation operators, adaptive operator probabilities, and domain-dependent operators. Future research should also examine the effects of using non-uniform penalty settings. The magnitude of the penalty can be used as a means to convey the confidence in the *a priori* information including whether the information is mandatory or a suggestion. Another direction for future research is to examine the convergence properties of probabilistic optimization techniques with *a priori* knowledge. Convergence studies have been conducted with optimization with fixed networks, which may be viewed as an extreme form of the knowledge incorporated in this paper [46].

In previous papers, we have shown that the performance of optimization algorithms consistently improves as the accuracy of their statistical models increases. In [4] we showed that trees generally performed better than chains, and chains generally performed better than models which assumed all variables were independent, such as those used in PBIL. The accuracy of the models can be improved through either using more complex models or by ensuring that the models that are created are more representative of the structure of the underlying search space. Unfortunately, when we move toward models in which variables can have more than one parent variable, the problem of finding an optimal network with which to model a set of data becomes NP-complete [8]. The methods presented in this paper provide a means to reduce the set of probabilistic models that must be considered – whether pair-wise or higher order dependencies are included. The incorporation of *a priori* information improves the accuracy of the models that are created given a limited number of samples. As shown, improved accuracy in the models leads to improved search results.

Acknowledgments

The author would like to gratefully acknowledge the contributions of Dr. Scott Davies in the studies that formed the foundation upon which this work was conducted. Both he and Dr. Kaari Flagstad Baluja provided very helpful feedback in proofing this paper.

References

[1] Baluja, S. (1994), "Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning," Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA.

[2] Baluja, S. (1995), "An Empirical Comparison of Seven Iterative and Evolutionary Heuristics for Static Function Optimization" Technical Report CMU-CS-95-193, Carnegie Mellon University, Pittsburgh, PA.

[3] Baluja, S. and Caruana, R. (1995), "Removing the Genetics from the Standard Genetic Algorithm," *Proceedings of the Twelfth International Conference on Machine Learning, 1995*, Prieditis, A., Russel, S. (Eds.), Morgan Kaufmann Publishers. San Mateo, CA. pp. 38-46.

[4] Baluja, S. and Davies, S. (1997), "Using Optimal Dependency Trees for Combinatorial Optimization: Learning the Structure of the Search Space," *Proceedings of the Fourteenth International Conference on Machine Learning, 1997,* Fisher, D.H. (Ed.), Morgan Kaufmann

Publishers. San Mateo, CA. pp. 30-38.

[5] Baluja, S. and Davies, S. (1998), "Fast Probabilistic Modeling for Combinatorial Optimization," *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98),* AAAI Press, pp 469-476.

[6] Bosman, P. and Thierens, D. (1999), "Linkage Information Processing in Distribution Estimation Algorithms," *Proceedings of the Genetic and Evolutionary Computation Conference 1999*, Banzhaf, W., Daida, J, Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M. (Eds.), Morgan Kaufmann Publishers, pp 60-70.

[7] Bosman, P. and Thierens, D. (2000), "Continuous Iterated Density Estimation Evolutionary Algorithms within the IDEA Framework", Tehnical Report UU-CS-2000-15, Utrecht University.

[8] Chickering, D., Geiger, D., and Heckerman, D. (1995), "Learning Bayesian Networks: Search Methods and Experimental Results," *Preliminary Papers of the Fifth International Workshop on Artificial Intelligence and Statistics*, pp. 112-128.

[9] Chow C. and Liu, C. (1968), Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462-467.

[10] De Jong, K. (1975), Analysis of Behavior of a Class of Genetic Adaptive Systems. Ph.D. Thesis. University of Michigan, Ann Arbor, MI.

[11] De Bonet, J., Isbell, C., and Viola, P. (1997), "MIMIC: Finding Optima by Estimating Probability Densities," *Advances in Neural Information Processing Systems 9*, Mozer, M.C., Jordan, M.I., and Petsche, T. (Eds). The MIT Press, pp 424-431.

[12] Eshelman, L.J., Mathias, and K.E., Schaffer, J.D. (1996), "Convergence Controlled Variation", in *Proc. Foundations of Genetic Algorithms 4*. Morgan Kaufmann Publishers, San Mateo, CA., pp. 203-224.

[13] Etxeberria, R. and Larrañaga P. (1999), "Global Optimization using Bayesian networks". *Proceedings of the Second Symposium on Artificial Intelligence*. pp. 332-339

[14] Gallagher, M., Fream, M., and Downs, T. (1999), "Real-Valued Evolutionary Optimization Using a Flexible Probability Density Estimator," *Proceedings of the Genetic and Evolutionary Computation Conference 1999*, Banzhaf, W., Daida, J, Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M. (Eds.), Morgan Kaufmann Publishers, pp 840-846.

[15] Goldberg, D.E. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA. Addison-Wesley.

[16] González, C., Lozano, J.A., and Larrañaga, P. (2000), "Analyzing the PBIL Algorithm by Means of Discrete Dynamical Systems", *Complex Systems* 12(4), 465-479.

[17] González, C., Lozano, J.A., and Larrañaga, P. (2001), "The Convergence Behavior of the

PBIL Algorithm: A Preliminary Approach", *Proceedings of the Fifth International Conference on Artificial Neural Networks and Genetic Algorithms*, Springer, pp.228-231.

[18] Greene, J.R. (1996), "Population-Based Incremental Learning as a Simple Versatile Tool for Engineering Optimization," *Proceedings of the First International Conf. on EC and Applications*, pp. 258-269.

[19] Heckerman, D. (1996), "A Tutorial on Learning with Bayesian Networks", Technical Report MSR-TR-95-06. Microsoft.

[20] Heckerman, D., Geiger, D., and Chickering, D. (1995), "Learning Bayesian Networks: The Combination of Knowledge and Statistical Data," *Machine Learning* 20:197-243.

[21] Hertz, J., Krogh A., and Palmer R.G. (1991), *Introduction to the Theory of Neural Computing.* Addison-Wesley, Reading, MA.

[22] Hohfeld, M. and Rudolph, G. (1997), "Towards a Theory of Population-Based Incremental Learning," *Proceedings of the 4th IEEE Conference on Evolutionary Computation*. Piscataway, NJ: IEEE Press 1997, pp. 1-5.

[23] Holland, J.H. (1975), *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan Press.

[24] Jensen, F.V. (2001), *Bayesian Networks and Decision Graphs*. Springer-Verlag, 2001.

[25] Juels, A. (1996), *Topics in Black-box Combinatorial Optimization*. Ph.D. Thesis, University of California - Berkeley.

[26] Keane, A (2000), *The Options Design Exploration System: Reference Manual and Users Guide*, available from: http://www.soton.ac.uk/~ajk/options/welcome.html.

[27] Kvasnica, V., Pelikan, M, and Pospical, J. (1996), "Hill Climbing with Learning (An Abstraction of Genetic Algorithm). *Neural Network World (Czech Republic)*, 6(5): 773-796.

[28] Larrañaga, P., Etxeberria, R., Lozano, A., and Peña, J. (1999), Optimization by learning and simulation of Bayesian and Gaussian networks. Technical Report EHU-KZAA-IK-4/99, Dept. of Computer Science and Artificial Intelligence, University of the Basque Country.

[29] Larrañaga, P., Etxeberria, R., Lozano, A., and Peña, J. (2000), "Optimization in continuous domains by learning and simulation of Gaussian networks", *Optimization By Building and Using Probabilistic Models Workshop in the GECCO-2000 Conference*. pp. 201-204.

[30] Larrañaga, P., Etxeberria, R., Lozano. J.A., and Peña, J.M. (2000), "Combinatorial Optimization by Learning and Simulation of Bayesian Networks", *Proceedings of the Conference in Uncertainty in Artificial Intelligence*, Morgan Kauffman Publishers, pp 343-352.

[31] Larrañaga, P. and Lozano, J., (2001), *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, Kluwer Academic Publishers.

[32] Mahnig, T. and Muhlenbein, H. (2001), "Mathematical Analysis of Optimization Methods Using Search Distributions," *Optimization By Building and Using Probabilistic Models Workshop in the GECCO-2000 Conference*. pp. 205-208.

[33] Muhlenbein, H. (1997), "The Equation for Response to Selection and Its Use for Frediction", *Evolutionary Computation*, 5(3), 303-346.

[34] Muhlenbein, H. and Mahnig, T. (1999), "Convergence Theory and Applications of the Factorized Distribution Algorithm" *Journal of Computing and Information Technology*, 7, pp. 19-32.

[35] Muhlenbein, H., Mahnig, T. and Rodriguez, A.O. (1999), "Schemata, Distributions and Graphical Models in Evolutionary Optimization" *Journal of Heuristics*, 5, pp 215-247.

[36] Ochoa, A., Muehlenbein, H., and Soto, M. (2000), "Factorized Distribution Algorithms using Bayesian Networks of Bounded Complexity," *Optimization By Building and Using Probabilistic Models Workshop in the GECCO-2000 Conference*. pp. 212-215.

[37] Ochoa A., Soto M., Santana R., Madera J. C., and Jorge N. (1999), "The Factorized Distribution Algorithm and The Junction Tree: A Learning Perspective" *Proceedings of the Second Symposium on Artificial Intelligence*, pp. 368-377.

[38] Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

[39] Pelikan, M., Goldberg, D.E., Lobo, F. (2002), "A Survey of Optimization by Building and Using Probabilistic Models" *Computational Optimization and Applications*, 21 (1): 5-20.

[40] Pelikan, M., Sastry, K., and Goldberg, D.E. (2001), "Evolutionary Algorithms + Graphical Models = Scalable Black-Box Optimization," Technical Report 2001029, Illinois Genetic Algorithm Laboratory, University of Illinois at Urbana-Champaign.

[41] Pelikan, M., Goldberg, D.E., and Tsutsui, S. (2001), "Combining the Strengths of the Bayesian Optimization Algorithm and Adaptive Evolution Strategies", Technical Report 2001023, Illinois Genetic Algorithm Laboratory, University of Illinois at Urbana-Champaign.

[42] Sebag, M., and Ducoulombier, A. (1998), "Extending Population-Based Incremental Learning to Continuous Search Spaces", *Parallel Problem Solving from Nature V*, pp. 418-427.

[43] Syswerda, G. (1989), "Uniform Crossover in Genetic Algorithms," *Proceedings of the Third International Conference on Genetic Algorithms*, Schaffer, D. (Ed.), Morgan Kaufmann Publishers, San Mateo, CA, pp. 2-9.

[44] Syswerda, G. (1993), "Simulated Crossover in Genetic Algorithms," *Foundations of Genetic Algorithms 2,* Whitley, D.L. (Ed.), Morgan Kaufmann Publishers, San Mateo, CA. pp. 239-255.

[45] Thathachar, M, and Sastry, P.S. (1987), "Learning Optimal Discriminant Functions Through a

Cooperative Game of Automata," *IEEE Transactions on Systems, Man, and Cybernetics*, 17 (1): 73-85.

[46] Zhang Q. and Muehlenbein H. (1999), "On Global Convergence of FDA with Proportionate Selection," *Proceedings of the Second Symposium on Artificial Intelligence.* pp. 340-343.