The official version of this paper has been published by
American Association of Artificial Intelligence (http://www.aaai.org).
Paper appears in AAAI-98.

Fast Probabilistic Modeling for Combinatorial Optimization

Shumeet Baluja
baluja@cs.cmu.edu
Justsystem Pittsburgh Research Center &
Carnegie Mellon University

Scott Davies
scottd@cs.cmu.edu
School of Computer Science
Carnegie Mellon University

Abstract

Probabilistic models have recently been utilized for the optimization of large combinatorial search problems. However, complex probabilistic models that attempt to capture inter-parameter dependencies can have prohibitive computational costs. The algorithm presented in this paper, termed COMIT, provides a method for using probabilistic models in conjunction with fast search techniques. We show how COMIT can be used with two very different fast search algorithms: hillclimbing and Population-based incremental learning (PBIL). The resulting algorithms maintain many of the benefits of probabilistic modeling, with far less computational expense. Extensive empirical results are provided; COMIT has been successfully applied to jobshop scheduling, traveling salesman, and knapsack problems. This paper also presents a review of probabilistic modeling for combinatorial optimization.

1 Background

Within the past few years, there have been several novel methods proposed for probabilistic modeling for combinatorial optimization. Unlike methods such as hillclimbing, which progress 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 sampled solutions are then used to update the model, and the cycle is continued.

By maintaining a population of points, genetic algorithms (GAs) can be viewed as creating *implicit* probabilistic models of the solutions seen in the search. GAs attempt to implicitly capture dependencies between parameters and the solution quality by maintaining a population of solutions. Samples are generated by applying randomized recombination operators to high-performance members of the population [Goldberg, 1989][Holland, 1975][De Jong, 1975]. Unlike the models explored in this paper, however, no explicit information is kept about which groups of parameters contribute to the quality of candidate solutions. One of the first steps towards making the GA's model more explicit was the "Bit-Based Simulated Crossover (BSC)" operator [Syswerda,1993]. 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[1], the number of members which 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 traditional crossover operators to generate new solutions, BSC generates new query points by stochastically assigning each bit's value by the probability of having seen that value in the previous population (the value specified by the weighted count) [Syswerda, 1993].

BSC used a population of solutions from which the sampling statistics were entirely rederived after each generation. In contrast, Population-based incremental learning (PBIL) incrementally adjusts its sampling statistics after each generation [Baluja, 1995]. Rather than being based on population-genetics, PBIL is very 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 [Thathachar & Sastry, 1987]. 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. This algorithm will be described in detail in Section 4.

Note that the probabilistic model created in PBIL is extremely simple. *There are no inter-parameter dependencies modeled; each bit is examined independently.* Although this simple probabilistic model was used, PBIL was very successful when compared to a variety of standard genetic algorithm and hillclimbing algorithms on

1. Note that in this paper, we will discuss combinatorial optimization with the solutions represented as binary vectors. However, all of the results can be trivially extended to higher cardinality alphabets.

numerous benchmark and real-world problems [Baluja, 1997][Greene, 1996]. A more theoretical analysis of PBIL can be found in [Juels, 1997][Kvasnicka *et al.*, 1995][Hohfeld & Rudolph, 1997].

The most immediate way in which the PBIL algorithm can be improved is to create mechanisms that capture inter-parameter dependencies. One of the first extensions to PBIL along these lines was termed *Mutual Information Maximization for Input Clustering (MIMIC)* [De Bonet *et al.*, 1997]. MIMIC captured a heuristically chosen set of the pairwise dependencies between the solution parameters. MIMIC maintained the top N% of all previously generated solutions, from which it calculated pair-wise conditional probabilities. MIMIC used a greedy search to generate a chain in which each variable was conditioned on the previous variable. The first variable in the chain, X_1, was 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 selected the variable with the lowest conditional entropy $H(X_{i+1} \mid X_i)$. As with PBIL, after creating the full chain, it randomly generated more samples from the distribution specified by this chain. The entire process was then repeated.

In [Baluja & Davies, 1997a], MIMIC's probabilistic model was extended to a larger class of dependency graphs: trees in which each variable is conditioned on at most one parent. As shown in [Chow and Liu, 1968], a simple algorithm can be employed to select the *optimal* tree-shaped network for a maximum-likelihood model of the data. In experimental comparisons, MIMIC's chain-based probabilistic models typically worked significantly better than PBIL's simpler models. The tree-based graphs typically worked significantly better than MIMIC's chains. Thus, using more accurate probabilistic models increased the probability of generating new candidate solutions in promising regions of the search space. Tree-Based graphs are the basis of the probabilistic models used in this study, and will be returned to in Section 2.

An extension of pair-wise dependency modeling is arbitrary dependency modeling. Bayesian networks [Pearl, 1988] are a popular method for efficiently representing dependencies and independencies in probability distributions. Bayesian networks are directed acyclic graphs in which each variable is represented by a vertex, and dependencies between variables are encoded as edges. As in the tree-based algorithm, the networks model probability distributions of the form shown in Equation (1).

$$P_B(X_1,\ldots,X_n) = \prod_{i=1}^{n} P_B\left(X_i \middle| \Pi_{X_i}\right) \qquad (1)$$

where Π_{X_i} is the set of X_i's parents in B and n is the number of nodes. The tree-shaped networks described in the previous paragraph are a special case of Bayesian networks in which each node in the graph has at most one parent. PBIL may be thought of as employing a degenerate Bayesian network in which the graph has no edges. Unfortunately, when we move toward models in which variables can have more than one parent, the problem of finding an optimal network with which to model a set of data becomes NP-complete [Chickering, *et al.*, 1995]. However, search heuristics have been developed for automatically learning Bayesian networks from data (for example [Heckerman, *et al.*, 1995]). A common approach is to perform hill-climbing over network structures, starting with a relatively simple network. This approach was used for combinatorial optimization in [Baluja & Davies, 1997b]. The empirical results with Bayesian networks showed a noticeable improvement over tree-based optimization in some problems that exhibit complicated dependencies. However, this benefit is achieved through significantly more computational effort. In other problems in which only a few dependencies must be modeled, the tree-based model performed as well as the Bayesian network.

Thus far, all of the approaches that model dependencies have been used for optimizing relatively small problems (with search spaces smaller than 2^{256}). Extending these models to large problems is challenging because of the severe computational expense of modeling the dependencies between a large number of variables. On the other hand, when using less computationally expensive search algorithms, it is a common procedure to restart them with random initialization points in the hope that they will find better local optima. After performing several such restarts, there is information to be gained by analyzing the various local optima found in multiple search runs and looking for features they have in common.

One learning-based approach to selecting good starting points for hillclimbing was presented in [Boyan & Moore, 1997]. The algorithm attempts to map a set of user-supplied features of the state space to a single value. This value represents the quality of solutions that were found by hill-climbing runs passing through states with similar features. The algorithm then uses this "value function" to select promising starting points for future hillclimbing runs.

In this paper, we utilize probabilistic methods to model the solutions obtained by faster search algorithms. This contrasts with the approach of [Boyan and Moore, 1997] since an explicit probabilistic model is used to model only the top-performing solutions gathered through all of the fast searches. Further, the model is based directly on the parameters of the solution encoding; *no high-level attributes of the problem are provided to the algorithm.* These models

are then sampled to intelligently select new starting points for further searches. The resulting algorithm, termed COMIT (***C***ombining ***O***ptimizers with ***M***utual ***I***nformation ***T***rees), gains most of the benefits of modeling the dependencies in the search space at a significantly reduced computational cost.

In the next section, we describe the COMIT algorithm, and give details of how the probabilistic model is created. In Section 3, we illustrates the use of COMIT with search techniques, such as hillclimbing, that maintain only a single solution from which new solutions are generated. Section 4 demonstrates how COMIT can be integrated with algorithms that themselves model probability distributions of possible solutions, such as PBIL. Extensive empirical results are provided in both sections. Finally, Section 5 presents conclusions and directions for future research.

2 The COMIT Algorithm

The basic premise of the COMIT algorithm is that probabilistic models can be used to intelligently select starting points for fast search algorithms such as hillclimbing or PBIL. The model is created based upon good solutions gathered from all previous searches. The general procedure is shown in Figure 1.

Specifically, once we have chosen a model $P'(X_1...X_n)$ of the set, **S**, of previously found good solutions, P' is used to stochastically generate a number of candidate solutions. The best of these newly generated solutions are used to initialize the fast-search algorithm. Once the fast-search has terminated, up to MAX_INFLUENCE solutions in **S** are replaced by better solutions generated during the run; the size of **S** is kept constant. The MAX_INFLUENCE parameter is important, and must be set by considering the size of **S**. If MAX_INFLUENCE is set too high, too much weight may be given to a single fast-search run, resulting in premature convergence. If it is too low, then previously found good solutions may not carry enough weight; this causes the algorithm to put too much emphasis on random exploration. The process is repeated until a termination condition is met. The next section describes in detail how the probabilistic model of the good solutions is generated.

2.1 Modeling Dependencies in the COMIT algorithm

Suppose we have a set of good solutions, **S**, found from previous fast-search runs. We wish to discover what inter-parameter dependencies are exhibited by the bit strings in **S**, and use this information to generate good starting points for future hillclimbing runs. To do this, we try to model a probability distribution $P(\mathbf{X}) = P(X_1, ..., X_n)$ over bitstrings 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.** As in [Baluja & Davies, 1997a], we restrict our model $P'(X_1, ..., X_n)$ to the following form:

- Initialize dataset **S** with random solutions from uniform distribution
- While termination condition is not met:
  - Create probabilistic model of **S**.
  - Use model to stochastically generate **K** solutions. Evaluate new solutions.
  - Execute fast-search procedure, initialized with the best solutions generated from the model.
  - Replace up to **MAX_INFLUENCE** bitstrings in **S** with better bitstrings found during the fast-search run just executed.

USER DEFINED CONSTANTS (values used in this study)

|**S**|: Constant size of the dataset **S (1000)**

MAX_INFLUENCE: max number of bitstrings to replace in **S** with better strings from a single fast-search run (100).

Figure 1: Overview of the COMIT algorithm.

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

where Π_{X_i} is X_i's single "parent" variable. 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 $(\Pi_{X_i} = X_j) \Rightarrow m(i) < m(j)$ 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. As described earlier, more complete Bayesian Networks could be used; however, trees were used because of computational limitations.

A method for finding the optimal model within these restrictions is given in [Chow and Liu, 1968]. 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:

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

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 deter-

mines 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 produces a tree that maximizes:

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

this 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):

$$D(P \,\|\, P') = \sum_{\boldsymbol{X}} P(\boldsymbol{X}) \log \frac{P(\boldsymbol{X})}{P'(\boldsymbol{X})} \quad (5)$$

As shown in [Chow & Liu, 1968], this produces the tree-shaped network that maximizes the likelihood of **S**, under the assumption that the members of **S** were generated independently and identically distributed. Our optimization algorithm violates this assumption since many highly correlated members are added to the dataset from any single fast-search run; however, the use of the MAX_INFLUENCE parameter limits this correlation.

This tree generation algorithm, summarized in Figure 2, runs in time $O(|\mathbf{S}|*n^2)$, where **|S|** is the size of **S** and **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 bestMatchInTree[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 $I(X_{add}$, bestMatchInTree[X_{add}]), using ***S*** to estimate the relevant probability distributions.
  - Add X_{add} to tree, with bestMatchInTree[X_{add}] as parent.
  - For each bit X_{out} not in the tree, if $I(X_{out}$, bestMatchInTree[X_{out}]) < $I(X_{out}, X_{add})$. then set bestMatchInTree[X_{out}]=X_{add}.

Figure 2: Procedure for generating the dependency tree.

3 Using COMIT with Hillclimbing

This section illustrates the use of COMIT with search techniques, such as hillclimbing, that maintain a single solution from which new solutions are generated. The algorithm uses a tree-shaped probabilistic network, **T**, to model a set, **S,** of previously found good solutions. **T** is then sampled to generate **K** new solutions, of which the highest-evaluation solution is used as the starting point of a hillclimbing run. Up to **MAX_INFLUENCE** solutions in **S** are replaced with better solutions from this run, and the process is repeated. The algorithm is summarized in Figure 3.

- Initialize dataset **S** with random solutions from uniform distribution
- While termination condition is not met:
  - Create a tree-shaped probabilistic network **T** that models **S**.
  - Use **T** to stochastically generate **K** solutions. Evaluate these **K** new solutions.
  - Start hillclimbing run initialized with the single best of the **K** solutions.
  - Replace up to **MAX_INFLUENCE** bitstrings in **S** with better bitstrings found during the hillclimbing run just executed.

USER DEFINED CONSTANTS (values used in this study)

|S|: Constant size of the dataset **S. (1000)**

MAX_INFLUENCE: max number of bitstrings to replace in **S** with better strings from a single fast-search run (100).

Figure 3: Overview of the COMIT algorithm with hillclimbing.

3.1 Algorithm Details

Hillclimbing (HC): The baseline search technique is next-ascent stochastic hillclimbing. The hillclimbing algorithm used has three notable properties. First, it allows moves to solutions with higher or equal evaluation; this is extremely important for hillclimbing to work well in many complicated spaces, since this allows it to explore plateaus. Second, before restarting, up to **PATIENCE** evaluations are allowed that are worse than the best evaluation seen so far in the run. Evaluations which are equal to the best evaluation seen so far are not counted towards the **PATIENCE** count. This parameter has a large impact on the effectiveness of hillclimbing in large search spaces. Therefore, for each problem, multiple settings were tried for this parameter. The range of values was (1*|**X**|) to (10*|**X**|), where |**X**| is the length of the solution encoding. The results with the best setting of the **PATIENCE** parameter are reported. Third, the hillclimbing algorithm used is a next-ascent hillclimber; as soon as a better solution is found, it is accepted. This contrasts with steepest-ascent hillclimbing, which searches all possible single-bit changes and accepts the one with the largest improvement. On the problems explored here, steepest ascent hillclimbing did not work as well.

COMIT: We experiment with two versions of the COMIT algorithm: one with **K** set to 100 (termed COMIT-100), which samples the tree 100 times before selecting the best point; and one with K=1000 (termed COMIT-1000). These evaluations are counted against the total allowed.

Augmented Hillclimbing (AHC): The fact that COMIT-***K*** examines **K** points before choosing one to use for hillclimbing is a possible confounding factor in determining how effective COMIT is in comparison to HC. To ensure that it is not simply the process of selecting these **K** before hillclimbing that gives performance gains, we augment hillclimbing as follows. Before the beginning of each run, AHC-***K*** examines **K** *randomly* chosen points from which it selects the best one as the starting position. (The difference

between this and COMIT is that COMIT samples **K** points from the dependency tree). Two versions of AHC are examined: AHC-100, and AHC-1000.

Note that all of the parameters of all of the algorithms were tuned on the 100-city TSP problem, and were held constant for all runs and all problems.

3.2 Results

For each algorithm on each problem, we try multiple settings of the **PATIENCE** parameter. The setting of the **PATIENCE** parameter that gives the best result is reported here. The results reported are the average of at least 25 runs of the entire algorithm. Each algorithm is given 200,000 function evaluations on each problem. Because of space limitations, the full description of the problems cannot be given here. However, brief descriptions are given in Appendix A. The results are shown in Table I. In the first line of each cell in the table, the numerical results are presented. In the second line, the rank (1[best]..5[worst]) of each algorithm is given. Also given for HC, AHC-100 and AHC-1000 is whether the difference between the results achieved is significantly different from that of COMIT-100 and COMIT-1000, respectively. The significance is measured by the Mann-Whitney test (a non-parametric equivalent to the *t*-test) at the 95% confidence interval.

Table I: COMIT with Hillclimbing

	Size of Problem in Bits (Goal: MAX or MIN)	HC (≠C100, ≠C1000) (95% Conf. Interval)	AHC-100 (≠C100, ≠C1000) (95% Conf. Interval)	AHC-1000 (≠C100, ≠C1000) (95% Conf. Interval)	COMIT-100	COMIT-1000
Knapsack 512 elem.	512 (MAX)	3238 5 (Y,Y)	3377 3 (Y,Y)	3335 4 (Y,Y)	6684 1	6259 2
Knapsack 900 elem.	900 (MAX)	3403 5 (Y,Y)	3418 4 (Y,Y)	3488 3 (Y,Y)	7733 1	7182 2
Knapsack 1200 elem.	1200 (MAX)	5226 5 (Y,Y)	5270 4 (Y,Y)	5280 3 (Y,Y)	13052 1	12829 2
Jobshop-enc 1 10x10	500 (MIN)	998 5 (Y,Y)	988 4 (Y,Y)	982 3 (N,Y)	978 2	970 1
Jobshop-enc 2 10x10	700 (MIN)	965 5 (Y,Y)	961 4 (Y,Y)	957 3 (N,N)	954 2	953 1
Jobshop-enc 2 20x5	700 (MIN)	1207 5 (Y,Y)	1201 4 (Y,Y)	1199 3 (N,N)	1196 2	1196 1
Binpack (10^{-3}) 8 bins, 168 el.	504 (MIN)	1.70 5 (N,Y)	1.58 3 (N,N)	1.62 4 (N,N)	1.56 2	1.45 1
Binpack (10^{-2}) 16 bins, 200 el.	800 (MIN)	1.54 5 (Y,Y)	1.50 4 (Y,Y)	1.38 3 (Y,Y)	1.11 1	1.24 2
Summation Cancellation	675 (MIN)	64 5 (Y,Y)	61 4 (Y,Y)	59 3 (Y,Y)	54 2	52 1
TSP 100 city	700 (MIN)	1629 5 (Y,Y)	1599 4 (Y,Y)	1573 3 (Y,Y)	1335 1	1336 2
TSP 200 city	1600 (MIN)	15119 3 (N,N)	15286 5 (N,N)	15100 1 (N,N)	15189 4	15117 2
TSP 150 city	1200 (MIN)	11451 5 (Y,Y)	11247 3 (Y,Y)	11290 4 (Y,Y)	9812 2	9077 1

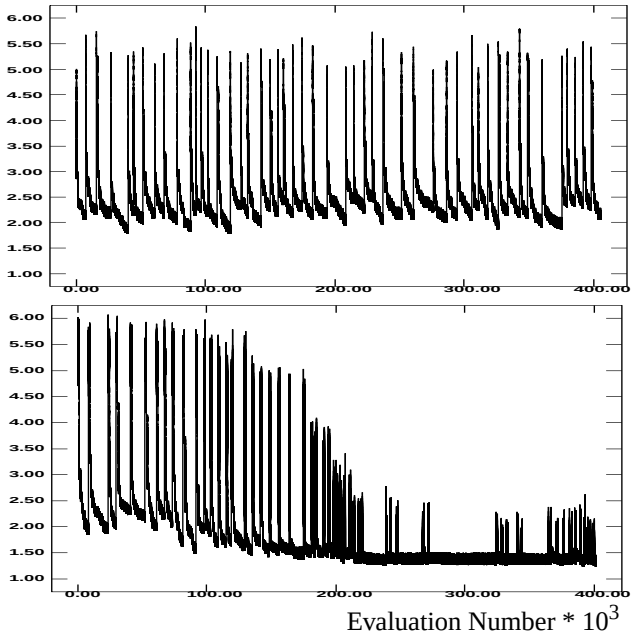

Figure 4: These graphs show the values of every evaluation performed by the HC (top) and COMIT (bottom) algorithms for the TSP domain. The object is to minimize the tour length. Note that these runs are extended to 400,000 evaluations.

In almost every problem examined, COMIT significantly improves the performance over hillclimbing. Only in one problem (200-city TSP) did one of the COMIT runs (COMIT-100) not perform as well as HC. However, the difference in performance was not statistically significant.

To provide some intuition about how the COMIT algorithm progresses, Figure 4 shows the values of each evaluation performed by the HC and COMIT-1000 algorithm in the TSP domain. There are four features that should be noticed. First, the spikes in the evaluations correspond to the beginning of hillclimbing runs. In the COMIT graph, the spikes also represent the **K** samples generated by sampling the tree. Second, for the COMIT algorithm, the random initial samples in the dataset **S** were entirely removed by evaluation #90,000 (this approximately corresponds to the number of evaluations used in the first 10 hillclimbing runs; each run contributed 100 samples to the dataset, and the size of **S** is 1000). Third, the magnitude of the spikes in the COMIT plot gradually decreases; this corresponds to the COMIT algorithm learning to seed the hillclimbing runs with high-quality solutions. Fourth, and most importantly, the final solutions found at each hillclimbing run have improved over standard hillclimbing, even before the HC runs are started at noticeably better solutions. This indicates that by using the interparameter dependency models to generate starting points, the hillclimbing runs are started in basins of the search space that lead to high-evaluation solutions.

4 Using COMIT with PBIL

The previous section demonstrated that relatively complex probabilistic models can be used in conjunction with search techniques, such as hillclimbing, that maintain only a single solution from which new solutions are generated. In this section, we examine how to integrate COMIT with other algorithms that themselves model probability distributions of possible solutions. As described in Section 1, PBIL only maintains unconditional probabilities; no inter-parameter dependencies are modeled. A vector, **P**, specifies the probability of generating a 1 in each bit position. Initially, all values in **P** are set to 0.5. A number of solution vectors are generated by stochastically sampling **P**; each bit is sampled independently of all the others. The probability vector is then moved towards the generated solution vector with the highest evaluation according to Equation 6. The update rule is similar to the updates used in unsupervised competitive learning [Hertz, *et al.* 1991].

$$ProbabilityVector_{t+1,i} = (1-\alpha) \cdot ProbabilityVector_{t,i} + \alpha \cdot BestSolutionVector_i \quad (6)$$

$ProbabilityVector_{t,i}$ 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 and its parameters are shown in Figure 5. The final result of the PBIL algorithm is the best solution generated throughout the search. The version used here extended this as suggested in [Baluja, 1997]: first, a mutation operator was added. This randomly selects positions in the probability vector to alter. The motivation for using a mutation operator is to preserve diversity in the generated solutions; therefore, the mutation operator always moves the probabilities to higher-entropy states (closer to 0.5). Second, instead of only moving towards the best solution, the probability vector was also moved away from the worst solution in the positions in which the best and worst solutions differed.

In the experiments conducted with COMIT in this section, three algorithms were compared. The first algorithm, which we use as a baseline, is PBIL. The parameters are set as follows: α=.15, M=1, N=50, MUT_PROB = 0.02, MUT_SHIFT= 0.05; these parameters are also used for the second and third algorithms. The second algorithm tested is COMIT-PBIL. COMIT-PBIL is identical to the version of COMIT used for hillclimbing except for the following differences: PBIL is used instead of hillclimbing; each PBIL run is terminated after 5000 evaluations without improvement. At the beginning of a PBIL run, each entry of the probability vector, **P**, is initialized to the average value of

```
****** Initialize Probability Vector ******
for i :=1 to LENGTH do P[i] := 0.5;

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

    best_solution_vectors =
       sort_solutions_best_to_worst (solution_vectors,evaluations);

    **** Update Probability Vector towards best solutions****
    for i := M downto 1 do
       for j :=1 to LENGTH do
             P[j] := P[j] * (1.0 - α) + best_solution_vectors[i][j]* (α);

    **** Mutation - Always move towards 0.5 ****
    for i := 1 to LENGTH do
       if (random (0,1) < MUT_PROB) then
             if (P[i] > 0.5) P[i] := P[i]*(1.0 - MUT_SHIFT);
             else P[i] := P[i]*(1.0 - MUT_SHIFT) + MUT_SHIFT

Return the best solution generated throughout the entire search.

PBIL CONSTANTS:
N: # of vectors generated before update of the probability vector.
α: the learning rate, how fast to exploit the search performed.
M: number of vectors in the population that are used to update P.
LENGTH: # of bits in the solution encoding (problem dependent).
MUT_PROB: probability of "mutating" each bit position.
MUT_SHIFT: amount a mutation alters the value in the bit position
```

Figure 5: Basic PBIL algorithm for a binary alphabet.

that position in the top 10% of the solutions generated by sampling the tree-based model 1000 times. As described in Section 2, the tree-based model is created from the data-set **S**, which contains the good solutions returned from multiple PBIL searches. As before, the size of **S** is 1000 and **MAX_INFLUENCE** is 100. The algorithm is shown in Figure 6. The third algorithm tested is PBIL with restarts; after 5,000 evaluations are conducted with no improvement, the algorithm is restarted. This test is to ensure that it is the probabilistic modeling (of COMIT) that is responsible for the increase in performance, not simply the fact that PBIL is being restarted. The parameters for all of the algorithms were tuned on the 100-city TSP problem, and no parameter changes were made for any other problem.

- Initialize dataset **S** with random solutions from uniform distribution
- While termination condition is not met:
  - Create a tree-shaped probabilistic network **T** that models **S**.
  - Use **T** to stochastically generate **K** solutions. Evaluate new solutions.
  - Select top **C**% of the **K** generated solutions.
  - Initialize PBIL's **P** vector to unconditional probabilities in selected solutions.
  - Execute PBIL run. Replace up to **MAX_INFLUENCE** bitstrings in **S** with better bitstrings found during the PBIL run just executed.

USER DEFINED CONSTANTS (values used in this study)

|**S**|: Constant size of the dataset **S. (1000)**

MAX_INFLUENCE: max number of bitstrings to replace in **S** with better strings from a single fast-search run (100).

C: Percentage of **K** generated solutions used to initialize **P**. **(10)**

Figure 6: Overview of the COMIT algorithm for PBIL.

The results are shown in Table II. Each experiment was given 600,000 evaluations. The results reported are the average of at least 50 experiments per algorithm, per problem. In the first line of each cell in the table, the numerical results are presented. In the second line, the rank (1[best]..3[worst]) of each algorithm is given. Also given for PBIL with no-restarts and random restarts is whether the difference between the results is significantly different from those of COMIT-PBIL.

Table II: COMIT with PBIL

	Size of Problems in Bits (Goal: Maximize or Minimize)	PBIL - No Restart (≠ to COMIT at 95% conf)	PBIL - Restart (≠ to COMIT at 95% conf)	COMIT-PBIL
Knapsack 512 elem.	512 (MAX)	7823 2 (Y)	7765 3 (Y)	7872 1
Knapsack 900 elem.	900 (MAX)	9601 2 (Y)	9402 3 (Y)	10134 1
Knapsack 1200 elem.	1200 (MAX)	14668 2 (Y)	13768 3 (Y)	18014 1
Jobshop-enc 1 10x10	500 (MIN)	982 3 (Y)	963 2 (N)	961 1
Jobshop-enc 2 10x10	700 (MIN)	959 3 (Y)	943 2 (Y)	940 1
Jobshop-enc 2 20x5	700 (MIN)	1188 3 (Y)	1176 2 (Y)	1170 1
Binpack (10^{-3}) 8 bins, 168 el.	504 (MIN)	1.4 1 (Y)	2.8 2 (N)	2.8 3
Binpack (10^{-2}) 16 bins, 200 el.	800 (MIN)	8.3 1 (Y)	9.6 2 (N)	10.0 3
Summation Cancellation	675 (MIN)	25 1 (Y)	33 2 (N)	33 3
TSP 100 city	700 (MIN)	1331 2 (Y)	1518 3 (Y)	1021 1
TSP 200 city	1600 (MIN)	17148 2 (Y)	21858 3 (Y)	14870 1
TSP 150 city	1200 (MIN)	11035 2 (Y)	13698 3 (Y)	9078 1

In summary, in the majority of the problems examined, the COMIT-PBIL approach was significantly better than PBIL, both with and without restarts. This is encouraging, since PBIL has itself often outperformed standard genetic algorithms and hillclimbing techniques on similar problems [Baluja, 1997]. Two exceptions to this are the binpacking problem and the summation cancellation problem. In both of these problems, COMIT's choice of restarting points did not hurt performance; however, not restarting PBIL at all led to better performance.

5 Conclusions & Future Work

We have shown that probabilistic models can be used to combine the information gathered from runs of fast search algorithms. By using a model of the interparameter dependencies in previously found good solutions, new starting points for the fast search algorithm are chosen. In most of the problems examined, this has led to the discovery of significantly better final solutions.

One can imagine that COMIT lies in the middle of a continuous spectrum, with the ends representing never using a probabilistic model (as is done with hillclimbing) and always using a probabilistic model (as is done in [Baluja & Davies, 1997]). In the Tree-based algorithm described in [Baluja & Davies, 1997a], all of the candidate generation is done by sampling the model, and the high-evaluation points are added directly back into **S**. The empirical results with COMIT have demonstrated that even infrequent use of the probabilistic model is advantageous. An immediate direction for future research is to determine if there is any performance loss incurred by using the complex probabilistic models only for the initialization of searches when compared to always using the probabilistic model (which is extremely computationally expensive). In the preliminary tests conducted, the COMIT algorithm was often able to perform as well as the Tree-Based algorithm. Quantifying the difference in performance and computational expense is currently being researched.

Because different search algorithms have different sampling procedures, each may explore different regions of the solution space. By allowing all of the search algorithms to contribute solutions to the dataset from which dependencies are modeled, COMIT provides a simple method for combining multiple different search algorithms.

Since the probabilistic model is updated infrequently by COMIT, it may be feasible to replace the dependency-tree model with more sophisticated but computationally expensive models, such as general Bayesian networks. It will be interesting to determine if automatically learning networks with hidden variables [Friedman, 1997] would improve optimization performance. It will also be interesting to examine whether it is possible to explicitly use the fact that the samples modeled are *not* independent, as is assumed by the probabilistic models used here.

We illustrated how to incorporate COMIT with PBIL and hillclimbing. Although not reported here, we have used COMIT to restart genetic algorithm (GA) based searches. This combination improved the performance over the GA alone. However, neither the GA nor COMIT-GA were typically able to perform as well as the hillclimbing and PBIL approaches presented here. COMIT requires no change to be used with other search algorithms such as simulated annealing or TABU search [Glover, 1989]. In addition to these general search techniques, it can also be used with randomized search techniques designed to address specific problems, such as WALKSAT [Selman *et. al*, 1996].

References

Baluja, S. (1997) "Genetic Algorithms and Explicit Search Statistics," *Advances in Neural Information Processing Systems* 9, 1996. Mozer, M.C., Jordan, M.I., & Petsche, T. (Eds). MIT Press.

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.

Baluja, S. & Davies, S. (1997a) "Using Optimal Dependency-Trees for Combinatorial Optimization: Learning the Structure of the Search Space", *Proc. 1997 International Conference on Machine Learning*. pp 30-38.

Baluja, S. & Davies, S. (1997b) "Probabilistic Modeling for Combinatorial Optimization", *Pre-Print*.

Boyan, J. & Moore, A. (1997) "Using Prediction to Improve Global Optimization", to appear in *Sixth Intl. Wkshp. on AI & Statistics*.

Chickering, D., Geiger, D., and Heckerman, D. (1995) "Learning Bayesian networks: Search methods and experimental results," *Proc.of Fifth Conference on Artificial Intelligence and Statistics*

Chou. C. and Liu, C. (1968) Approximating discrete probability distributions with dependence trees. *IEEE Trans. on Info. Theory*, 14:462-467.

De Bonet, J., Isbell, C., and Viola, P. (1997) "MIMIC: Finding Optima by Estimating Probability Densities," *Advances in Neural Information Processing Systems*, 1996. Mozer, M.C., Jordan, M.I, & Petsche, T. (Eds).

Fang, H.L., Ross, P. & Corne, D. "A Promising GA Approach to Job-Shop Scheduling, Rescheduling and Open-Shop Scheduling Problems". In *Proc. Int. Conf. on GAs-95*. S. Forrest, (ed). Morgan Kaufmann.

Friedman (1997) "Learning Belief Networks in the Presence of Missing Values and Hidden Variables," *Proc. 1997 International Conference on Machine Learning*. pp 125-133.

Glover, F. (1989) "Tabu-Search - Part I", *ORSA Journal on Computing* 1:190-206.

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

Heckerman, D., Geiger, D., and Chickering, D. (1995) "Learning Bayesian networks: The combination of knowledge and statistical data," *Machine Learning* 20:197-243.

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

Hohfeld, M. & Rudolph, G. (1997) "Towards a Theory of Population-Based Incremental Learning", *International Conference on Evolutionary Computation*. pp. 1-5.

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

Kvasnica, V., Pelikan, M, Pospical, J. "Hill Climbing with Learning (An Abstraction of Genetic Algorithm). *In Proceedings of the First International Conference on Genetic Algorithms (MENDEL, '95)*. pp. 65-73.

Muth & Thompson (1963) *Industrial Scheduling* Prentice Hall International. Englewood Cliffs, NJ.

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

Selman, B. Kautz, H. & Cohen, B. (1996) "Local Search Strategies for Satisfiability Testing", in Johnson & Trick (eds) Cliques, Coloring and Satisfiability, DIMANCS VOlume 26 521-532.

Syswerda, G. (1989) "Uniform Crossover in Genetic Algorithms," *Int. Conf. on Genetic Algorithms 3*. 2-9.

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

Thathachar, M, & Sastry, P.S. (1987) "Learning Optimal Discriminant Functions Through a Cooperative Game of Automata", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 17, No. 1.

APPENDIX A: Problem Descriptions

Due to space limitations, only brief descriptions of the problems are given here. Details can be found in the referenced reports.

1. Traveling Salesman Problems (TSP)

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 [Syswerda, 1989] for details. Three problem were attempted: 100, 200 and 150 city.

2. Jobshop Scheduling Problems

Two standard test problems are attempted, a 10-job, 10-machine problem and a 20-job, 5-machine problem. A description of the problems can be found in [Muth & Thompson, 1963]. The first problem is encoded in two ways. The first encoding is commonly used with genetic algorithms; see [Fang *et. al*, 1993].

The second encoding [Baluja, 1995] is very similar to the encoding used in the Traveling Salesman Problem. The drawback of this encoding is that it uses more bits than the previous one. Nonetheless, empirically, it revealed improved results. Each job is assigned M entries of size $\log_2(J*M)$ bits. The total length of the encoding is $J*M*\log_2(J*M)$. The value of each entry (of length $\log_2(J*M)$) determines the order in which the jobs are scheduled. The job that contains the smallest valued entry is scheduled first, etc. The order in which the machines are selected for each job depends upon the ordering required by the problem specification.

3. Knapsack Problem

In this problem, there is a bin of limited capacity, and **M** elements of varying sizes and values. The goal is to select the elements that yield the greatest summed value without exceeding the capacity of the bin. A penalty is given to solutions that exceed the maximum capacity; the encoding was taken from [Baluja, 1995]. Three versions of the problem were attempted with 512, 900, and 1200 elements.

4. Bin Packing/Equal Piles

In this problem there are **N** bins of varying capacities and **M** elements of varying sizes. The goal is to pack the bins with elements as tightly as possible, so that the size of the bins closely matches the total size of the elements assigned to the bins. The solution is encoded in a bit string of length $\mathbf{M}*\log_2\mathbf{N}$. Each element is assigned to a bin (which is encoded in $\log_2\mathbf{N}$ bits). These problems were generated so that there was an assignment of elements which matched the capacities of the bins exactly. Two version of this problem are explored, the first with 8 bins and 168 elements, and the second with 16 bins and 200 elements.

5. Summation Cancellation

In this problem, there is very strong parameter interdependence. The parameters 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. The problem had 75 parameters, and each parameter was represented with 9 bits, encoded in standard base-2, with the values uniformly spaced between -2.56 and +2.56. It is set as a maximization problem by using the reciprocal of the function.

$$-2.56 \le s_i \le 2.56 \qquad i = 1 \ldots N$$

$$y_i = s_i + y_{i-1} \qquad i = 2 \ldots N \qquad y_1 = s_1$$

$$f = \frac{1.0}{C + \left| \sum_1^N |y_i| \right|} \qquad C = \frac{1}{100000}$$