Guiding Constructive Search with Statistical Instance-Based Learning

Orestis Telelis Panagiotis Stamatopoulos
Department of Informatics and Telecommunications
University of Athens
157 84 Athens, Greece
{telelis,takis}@di.uoa.gr

Abstract

Several real world applications involve solving combinatorial optimization problems. Commonly, existing heuristic approaches are designed to address specific difficulties of the underlying problem and are applicable only within its framework. We suspect, however, that search spaces of combinatorial problems are rich in intuitive statistical and numerical information, which could be exploited heuristically in a generic manner, towards achievement of optimized solutions. Our work presents such a heuristic methodology, which can be adequately configured for several types of optimization problems. Experimental results are discussed, concerning two widely used problem models, namely the Set Partitioning and the Knapsack problems. It is shown that, by gathering statistical information upon previously found solutions to the problems, the heuristic is able to incrementally adapt its behaviour and reach high quality solutions, exceeding the ones obtained by commonly used greedy heuristics.

1. Introduction

An important issue to notice in combinatorial optimization (CO) problems that emerge from real world applications is that they exhibit some inherent structural and statistical properties. These properties constitute observable common knowledge for the humans that are in charge of solving the problem. The human experience transforms into heuristic tools for obtaining a satisfactory solution. In most heuristic algorithms, important knowledge concerning a particular problem is embedded in an abstracted and more generic form, so that it can be applied in multiple instances of the same problem model. This abstraction, however, is an obstacle in recognizing specific numerical and structural properties of the particular instance being solved.

The application of machine learning towards achievement of optimized solutions is a relatively recent aspect. However, there have been some remarkable, as well as pioneering, works dealing with it. We discuss here some of them by reporting briefly their point of view.

A reinforcement learning approach has been described by Boyan and Moore for learning evaluation functions of startup search states for a local search algorithm [5].

Reinforcement learning methods [8] have gained great attention because of their state-reward policy, which seems to fit well in the search state paradigm imposed by problem solving. Another related work addresses a job-shop scheduling problem through application of reinforcement learning [16]. Accumulation of scheduling control knowledge through reinforcements has also been exploited for obtaining repaired schedules [12]. Applications of machine learning for deciding the best search policy on a problem, as well as for configuring specific problem solving methods constitute an alternative research direction [6,14]. Analytical learning techniques have also been used for symbolic heuristic induction [10].

Four main research directions of statistical machine learning application to combinatorial optimization are surveyed in [4]. The approach presented in our paper shares a common part with the *search space understanding* direction because it gathers statistical information relative to properties of the search space during the solving process. It also lies in part within the *evaluation function learning* discipline, since a *Kernel Regression* (*KR*) scheme is employed for the approximation of an evaluation function, which shares its optimum with the objective function of the problem.

Within the framework of evaluation function learning, three main directions of application to combinatorial optimization have appeared in recent literature, according to [13]. One is to learn an evaluation function for guiding local search [5,1] to obtain improved solutions over a single instance of some CO problem. The second, mostly ambitious and interesting approach, focuses on learning evaluation functions generically applicable across several instances of the same CO problem [13,1,16]. Finally a third approach [3], within which our work falls, is about guiding the direct construction of solutions to the problem. In particular, the approach mentioned in [3], is about learning a global policy for guiding constructive search through reinforcement learning methods, while we investigate the dynamics of a local approximation scheme for learning evaluation functions.

To our knowledge, machine learning techniques have been mostly integrated in local search procedures. In this paper, we present a heuristic function which employs *KR* and is designed to cooperate with *solution constructive search methods* for global optimization. This aspect is of particular interest in solving CO problems, since constructive search methods are able to preserve the validity of a problem's constraints during the search. This is not the case with local search procedures. They often visit invalid search states, and thus it is harder to even find a feasible solution.

In the following, the proposed methodology is presented initially at a higher level and then, the machine learning based heuristic algorithm is described in more detail. The application of the approach on two specific problems, namely the knapsack problem and the set partitioning problem, is discussed next. Comments on the experimental results are made and, finally, the concluding remarks are presented and further work is briefly described. A preliminary version of our work has appeared in [15].

2. Framework Overview

Constructive search is the kind of solving procedure exploited in this work. By the term constructive search, we designate the construction of a solution to the CO problem by assigning a value to each decision variable in turn, and thus by searching

a tree of variable assignments. This search policy comes in contrast with the various local search techniques, such as hill climbing and simulated annealing, which alter an already known complete assignment of all decision variables of the problem at each step, in order to obtain a new one.

As far as optimality is concerned, the main interest in tree search is associated with selecting dynamically the most promising combination of variable and value to assign, in order to proceed towards a near-optimal solution (in terms of the problem's objective function value). Several problem-specific heuristics exist, which provide an upper (or lower) bound to the expected quality of the solution that probably lies beneath the current state of the search. Other heuristic policies calculate (based on incomplete information) an expected quality of the solution under construction. However, incompleteness of available information during the intermediate stages of construction is the curse of heuristics. Their estimations often prove to be inaccurate, thus misleading the search towards suboptimum search space regions.

In this paper, we propose a heuristic methodology for CO problems, which embeds simple machine learning methods for recognition and utilization of specific numerical problem properties. On every search state of the solving path, alternative choices are evaluated by a *KR* scheme. The evaluation of each choice is an estimation of the expected objective function value, under the assumption that the solving path extends in favour of the respective choice. Solutions previously found by the proposed methodology are utilized as training examples, for evaluation of *partial* solutions on each search state. The core assumption of our work is that good solutions lie somewhere *nearby* the best ones already found. Thus, we actually explore the search space by visiting neighbouring areas of known solutions, hoping to find better ones. The notion of neighbourhood is a rather geometric one as will be shortly apparent, and should not be confused with its corresponding definition in local search literature.

The huge search spaces of the CO problems faced in this work are not expected to supply consistent training sets. Therefore, the widely known *Leave One Out Cross Validation* (*LOOCV*) test adjusts the *KR* approximator's parameters with respect to the underlying training set, so that a minimal estimation error is achieved.

3. The Search Schema

The algorithmic schema is iterative. An overview of the approach is presented in Fig. 1. If I is a problem instance to be solved, the first thing to do is to use some simple heuristic method to obtain initial solutions S. A simple method might be a *Depth First Search* (DFS), guided by a common heuristic that intuitively fits the problem. This step stops after a limited time interval, which suffices for obtaining some initial solutions. The set S is then used for the production of an initial training set $\mathcal{E}$ for the machine learning algorithm employed by the heuristic.

The first step of the iterative process shown in Fig. 1 is a preprocessing procedure, which adapts the *KR* approximator to the training set, in order to achieve higher prediction accuracy with respect to the underlying training set $\mathcal{E}$. The problem instance is then solved by some constructive search algorithm, guided heuristically by the *KR* supported heuristic. The search stops when some criterion, such as a time limit, is met. The training set $\mathcal{E}$ is augmented with information extracted from newly found solutions

```
Find some solutions S to I using a "simple" method
Produce training set E(S)
Iterate until end criterion met
   Adapt KR approximator to E(S) using LOOCV
   Obtain new solutions S' with KR-Heuristic
   S ← S ∪ S'
   Produce training set E(S)
Return Best Solution
```

Figure 1: The overall algorithmic schema

and the process is repeated. The number of iterations is subject to experimentation. It is important to note that there is no constraint dictating that solutions obtained in each iteration should be better than those found in the previous iterations. The absence of such a constraint contributes to the enrichment of the training set with feasible solutions of varying qualities, which contribute to a more detailed picture of the search space.

The algorithm presented in Fig. 1 should terminate if in any iteration is unable to produce at least one new (not in $\mathcal{S}$) solution. Indeed, the predictions of the *KR* approximator depend solely on the training set $\mathcal{E}(\mathcal{S})$. Once the search within the loop fails to augment $\mathcal{S}$ with new solutions (case $\mathcal{S}' \subseteq \mathcal{S}$), the heuristic's suggestions remain unchanged for the next iteration. Then the search has to stop, as the algorithm is unable to proceed further. This case should be incorporated within the end criterion mentioned in Fig. 1. This weakness can be easily overcome through randomized restart of each iteration. However, our preference in constructive search remains justified, due to its ability of handling complicated constraints and, thus, being suitable for real world applications, which also offer a great potential of statistical information.

A simple data-flow representation of the algorithm appears in Fig. 2 and presents the main parts of our implementation. The dotted line encloses the iteratively interacting parts.

4. Integration of Instance-based Learning

In this section, various aspects of the heuristic algorithm are discussed concerning the instance-based learning method, the representation of training examples and the dynamic *KR* approximator selection.

4.1. *Kernel Regression on Nearest Neighbours*

The machine learning methodology exploited within the proposed framework belongs to the family of memory-based or instance-based methods [11]. Memory-based learning methods explicitly store all training examples they are shown. Only at prediction time do they perform non-trivial amounts of computation, which is their main disadvantage. We use the *Kernel Regression* (*KR*) method for approximating the value of a real function. The *KR* method is also known as *Locally Weighted Averaging*. There is a generic scheme for *KR*, which might be configured in many different ways as is the case for *locally weighted* learning methodologies [2].

The exact configuration of *KR* used in this work follows. The *KR* algorithm is

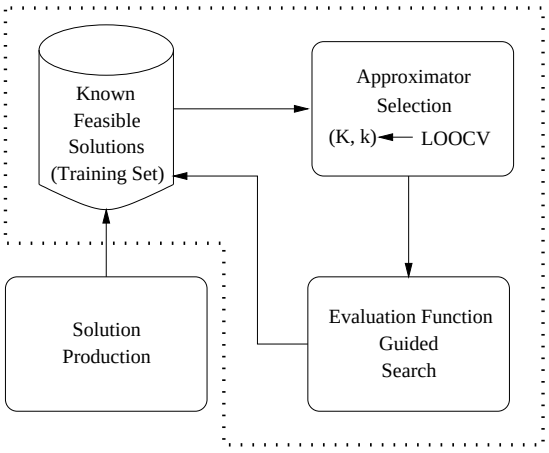

Figure 2: Parts of the implemented system and data flows between them

used here for real function value estimation. The function being approximated can be considered as: $f : \Re^n \mapsto \Re$.

The training set for the *KR* algorithm contains pairs of input vectors to f and their corresponding f-outputs. Thus, if $\vec{x} \in \Re^n$ is an input vector to the function, the respective training example contained in the training set will be the pair $\langle \vec{x}, f(\vec{x}) \rangle$.

Let $\vec{x_q}$ be a query to the *KR* algorithm. The prediction $\hat{f}(\vec{x_q})$ is calculated by the algorithm as follows:

$$\hat{f}(\vec{x_q}) = \frac{\sum_{i=1}^{k} K(d(\vec{x_i}, \vec{x_q}))f(\vec{x_i})}{\sum_{i=1}^{k} K(d(\vec{x_i}, \vec{x_q}))} \quad (1)$$

In formula (1), $d(\vec{x_i}, \vec{x_q})$ denotes the Euclidean distance of the query vector $\vec{x_q}$ from the i-th training example vector $\vec{x_i}$. The k parameter stands for the number of training examples nearest to $\vec{x_q}$, that contribute their known f-values to the prediction $\hat{f}(\vec{x_q})$.

The function $K : \Re \mapsto \Re$ is the *kernel* function, which assigns a smaller weight to the contribution of $f(\vec{x_i})$ to the sum, as much as greater is the distance of $\vec{x_i}$ from the query vector $\vec{x_q}$. Thus, the contribution of less significant values, i.e. values that correspond to more distant vectors, is punished. The kernel function to be used at each iteration of the search schema is determined dynamically by *LOOCV*, from a repertoire of available kernel functions. Dynamic selection of kernel function is a part of the *KR* approximator's adjustment, and will be discussed in a following paragraph in more detail. When the algorithm is presented with a query, all attributes are scaled down to the $[0, 1]$ range. This normalization helps avoiding the domination of large-ranged attributes in computations.

4.2. *Representation of Training Examples*

An important issue for the applicability of *KR* is the implicit definition of the f function, mentioned in the previous paragraph, whose value is going to be estimated. The function input consists of vectors in the Euclidean space $\Re^n$ which describe feasible solutions to the CO problem. The function value for each of these vectors is the value of the objective function of the problem for the corresponding solution.

Each training example for the *KR* approximation scheme is a pair of a solution descriptive vector $\vec{F}$, known as the *features vector*, and the respective objective function value $obj(\vec{F})$. Thus, the training set $\mathcal{E}$ can be defined as

$$\mathcal{E} = \{\langle \vec{F}, obj(\vec{F})\rangle \mid \vec{F}\text{: extracted from a solution}\} \ .$$

The features (i.e. the dimension values) of the features vectors are real arithmetic values that correspond to specific properties of the solution to the optimization problem. Each feature should be an aggregate function on the assignments of the problem variables. As in every step of the search a decision variable is selected to be assigned a value, the features of each vector should be calculated upon the solution path. The only limitations on the features that might belong in a features vector are imposed by the problem structure. As discussed later, a solution can be described through statistical information that is considered to be characteristic of the solution's quality in terms of the objective function value.

4.3. *Selection of* KR *Approximator*

The *Leave One Out Cross Validation* (*LOOCV*) test appears to be quite appropriate for adapting a *KR* approximator to the training set of each iteration in Fig. 1. A discussion on cross validation tests can be found in [9]. Within our study we have limited the selection of a proper *KR* approximator to the selection of a k-value and a kernel function K from a set of available kernel functions. A brief overview of commonly used kernel functions can be found in [2].

Each different pair of k-value (number of nearest neighbours contributing to the estimation) and kernel function yields a different *KR* approximator. For each candidate approximator, each training example is estimated, as if it was a novel example, using as training set the remaining training examples. The distance of this estimation from the actual target value is a measure of the error in prediction. The selected approximator is the one that yields the lowest average prediction error over all training examples.

To illustrate, assume that the set $\mathcal{K} = \{K_i \mid i = 1, \ldots, n\}$ contains the available kernel functions. If $\mathcal{E}$ is the available training set, then the test is going to examine each training example $e \equiv \langle \vec{f}, v\rangle \in \mathcal{E}$, using as training set $\mathcal{E} - \{e\}$. Thus, the k parameter receives a value in $\{1, \ldots, |\mathcal{E}| - 1\}$. For every parameter pair (K_i, k), the resulting approximator is tested over all training examples in $\mathcal{E}$. The approximator, that is the pair (K_i, k), which minimizes the average error $err_{(K_i,k)}$ over all training examples is selected as the most appropriate with respect to the underlying training set $\mathcal{E}$. Figure 3 describes the testing procedure. Although the procedure might seem time consuming because of the triple for-loop, a more efficient implementation than the one depicted in the figure is possible. In fact, the quantity $KR(K_i, k, \mathcal{E} - \{e\}, \vec{f})$

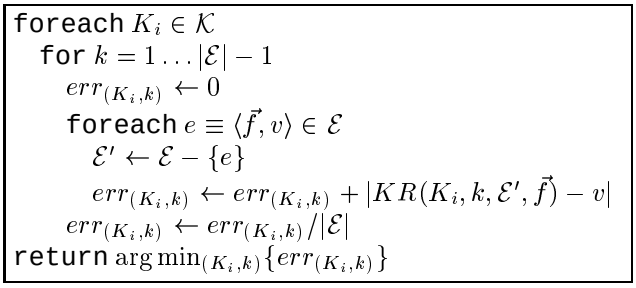

Figure 3: Leave-One-Out Cross Validation for *KR* approximator selection

can be calculated incrementally for increasing values of k, while keeping the rest of its arguments constant. This can be easily verified from the formula (1). Except from that, as will be clarified from our system's experimental configuration, the cardinalities $|\mathcal{K}|$ and $|\mathcal{E}|$ remain within acceptable bounds, so that the test procedure does not burden the system's time requirements.

As the proposed methodology is supposed to solve problems that enclose properties in a rather statistical than precisely defined manner, training data collected during the solving process are expected to be inconsistent. Search spaces of CO problems are extremely large and different solutions to a CO problem might belong to different neighbourhoods of the problem's search space. The features used for the description of the solutions are chosen empirically as representative of the problem's a priori known statistical properties. However, the selected features might prove to be insufficient for the discrimination of certain solutions. It is expected that some solutions might belong to different neighbourhoods, whereas their discrimination in terms of distance of features vectors might be inaccurate. Such inconsistency of the training data is handled via the dynamic approximator selection.

4.4. *The Heuristic Function*

During the construction of a potential solution, the system simultaneously constructs a path of variable assignments towards the bottom of the search tree. On each node of the tree visited, decisions must be taken, so that the next step down the tree is the most promising for the solution quality among all available choices. We describe the heuristic function which guides the search by exploiting the previous experience acquired by the system.

Let $\mathcal{P}$ be the so far constructed path during the search. This is a *partial* path. Each search step consists of two choices: the selection of an unassigned decision variable of the problem and the determination of a value to be assigned to it. Let $\mathcal{A}$ be the set of all possible (i.e. feasibility preserving) such assignments and, consequently, the set of all possible ways to augment the partial path $\mathcal{P}$. The heuristic function should dictate an assignment from $\mathcal{A}$ as the next step for the extension of $\mathcal{P}$.

As already discussed, the training examples for *KR* are features vectors calculated upon feasible solutions of the problem, i.e. upon complete paths. However, even a partial path can be used to calculate such a vector, if the unassigned decision variables

```
getBestAssignment(E, P, A)
  For each assignment α ≡ ⟨x_j = v⟩ ∈ A
    P̂ ← P ∪ {α}
    Calculate F⃗_P̂
    val_α ← KR(E, F⃗_P̂)
  Choose β ∈ A such that val_β is optimum
  return β
```

Figure 4: The *KR* supported heuristic function

are ignored. If statistical or aggregate information is used to describe the partial path extension by using a features vector, then it is reasonable to prefer extensions whose features present similarity to these of the best known solutions contained in the training set. In this way, portions of the search space that have previously produced good solutions are explored further. Let $\hat{\mathcal{P}}$ be the partial path resulting after augmenting $\mathcal{P}$ with a choice from $\mathcal{A}$. The features vector $\vec{F}_{\hat{\mathcal{P}}}$ is calculated upon $\hat{\mathcal{P}}$. The *KR* approximation scheme is requested to produce an objective function value estimation for $\vec{F}_{\hat{\mathcal{P}}}$. The extension of $\mathcal{P}$ which yields the optimum estimation is preferred over all other choices. An overview is presented in Fig. 4.

5. Application on Two Problems

The heuristic methodology was tested on two well known CO problems, namely the *knapsack* and the *set partitioning* problems. These are described below.

Knapsack. Given the n-dimensional vectors: profits $\vec{P} = \langle p_1, \ldots, p_n \rangle$ with $p_j \in \mathcal{Z}^+$, weights $\vec{W} = \langle w_1, \ldots, w_n \rangle$ with $w_j \in \mathcal{Z}^+$, $\vec{X} = \langle x_1, \ldots, x_n \rangle$ a vector of binary decision variables and some capacity $C \in \mathcal{Z}^+$,

$$\text{maximize } Z = \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to } \sum_{j=1}^{n} w_j x_j \leq C \ .$$

Set Partitioning (SP). Given a $m \times n$ binary matrix $A = \{a_{ij}\}$, a n-dimensional cost vector $\vec{C} = \langle c_1, \ldots, c_n \rangle$ with $c_j \in \mathcal{Z}^+$ and the n-dimensional vector $\vec{X} = \langle x_1, \ldots, x_n \rangle$ of binary decision variables, we want to

$$\text{minimize } Z = \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to } \sum_{j=1}^{n} a_{ij} x_j = 1, \quad i = 1 \ldots m \ .$$

5.1. *Generation of Problem Instances*

An important assumption mentioned in the very beginning of this paper is that the CO problems faced have some a priori known properties. For the previous problems, instances were generated that have such properties. In [14], a generation method for knapsack instances is briefly discussed. The $\vec{W}$ vector mentioned previously is determined from a normal distribution, while the fraction p_j/w_j is also calculated from a normal distribution. Each p_j element of the $\vec{P}$ vector is computed as $p_j = w_j \cdot p_j/w_j$. The knapsack instances created in this way tend to associate greater p_j values to greater w_j values. These problems have been shown to be more difficult [14]. Intuitively one can understand the difficulty that emerges as follows: any greedy heuristic trying to collect as many of the most profitable items within the knapsack is punished by their increased weight. This rough monotonicity correspondence between weight and profit is also a realistic matter, corresponding to real life experience.

For the SP problem, instances were generated in a similar way as for the knapsack problem. The number of 1s contained in the j column of $A_{m\times n}$ and the ratio of the column cost c_j to the number of 1s were determined by two different normal distributions, while the c_j quantity was computed as a dependent variable. Thus, columns containing more 1s, tend to have higher cost values c_j. Because the 1s in each column are decided via a uniform distribution, the mentioned property is slightly depressed by the satisfiability of the problem's constraints. Greedy thinking in this case, would impose construction of solutions by including columns of $A_{m\times n}$ which cover as much rows as possible. This practice tends to punish our choices by incurring higher costs.

For the SP and knapsack instances, optimum solutions were planted in a simple manner.

5.2. *Selection of Features*

The selection of features that assemble the features vectors is mostly important for the accuracy of the predictions of the *KR* scheme. Features that encapsulate some information relevant to the objective function's value are preferred, since they are expected to provide a quantitative partition of the search space into regions with expected objective function value. For each of the aforementioned problems, their features are presented, which were selected intuitively.

Knapsack. Three features constitute the features vector for the knapsack problem:

1. The mean value of the fraction p_j/w_j for js such that $x_j = 1$ in the solution.
2. The mean value of the profits p_j which participate in the objective function value, i.e. for js such that $x_j = 1$.
3. The weighted average of profits that participate in the objective function value. The contribution of each p_j profit to the average is weighted by the inverse corresponding w_j value.

Set Partitioning. For the SP problem, the features vector constituted of two features:

1. The mean value of the number of 1s contained in the columns of the binary matrix A that participate in a solution.
2. The mean value of the costs of the columns of matrix A that constitute the solution.

The j-th column of the matrix A is part of a solution if $x_j = 1$.

6. Experimental Results

Experiments were carried out on 10 instances for each problem. The instances were of varying sizes.

6.1. *Experimental Configuration*

For each problem, a heuristic solving method was chosen among the most commonly used. The chosen method provided the best results for the corresponding problem, within the experimentation time interval, and was employed for the construction of the initial training set. The results obtained by the proposed methodology were compared to those provided by the common method in the same time. Only solutions found by the common method within the first 10 minutes were considered for the construction of the initial training set. The common methods were applied for 4200 seconds. The overall running time of the iterative part of our methodology was arranged to last for 3600 seconds, so that summed to the initial training set construction time equals to 4200 seconds.

The system was provided a set of kernel functions to choose from, in order to configure a *KR* approximator, which would yield a minimum expected prediction error during the *LOOCV* test. We let the system choose among the following kernel functions: $K_a(d) = 1/d^a$ and $E_a(d) = 1/e^{d^a}$, for $a = 1, 2, 3$. The forms K_a and E_a lie among the most commonly used [2]. Particularly for the K_a form, rare $d = 0$ cases were handled by assigning the query vector an estimation equal to the known value for the corresponding nearest (zero distance) neighbour. Values of a greater than 3 were not considered, since they would yield very low kernel values, and thus, the *KR* estimation would be dominated by the contribution of one unique nearest neighbour.

The training set expands from one iteration to the next. In order to avoid cases where the *LOOCV* test would decide large values for k, and thus, slowing down the *KR* approximator, the size of the training set was kept stable to, at most, 25 training examples. At the end of each iteration, the training examples that represent solutions of worse qualities are removed.

The measurement of success for the experiments presented in this section is defined as the percentage of improvement achieved by our methodology towards the optimal solution, in comparison with the performance of the common method. Thus if c_o is the best solution found by the common method, ml_o is the best solution obtained by our methodology and opt is the optimal solution, performance is measured as

$$\alpha = 100 \times \frac{c_o - ml_o}{c_o - opt} \ .$$

Instance		Performance			
n	μ	c_o	ml_o	opt	α
3000	15	53848	54856	57499	27.6
4000	20	73954	77976	78693	84.8
4000	40	140669	143079	151415	22.4
4000	51	179656	186341	189211	69.9
4000	23	82385	83882	88849	23.1
4000	30	107621	109785	113112	39.4
6000	30	108744	111043	116577	29.3
6000	20	70956	71750	76909	13.3
8000	15	56049	57100	60031	26.4
8000	23	90257	91352	96367	17.9

Table 1: Experimental results for the knapsack problem

6.2. *Knapsack*

For the knapsack problem, the simple heuristic policy of *"try the most profitable choice first"* on a DFS proved to be quite successful in obtaining soon some high quality solutions.

Twelve iterations, of 5 minutes each, were performed on each problem instance after the construction of the initial training set. *Limited Discrepancy Search* (*LDS*) [7] was used as constructive procedure for the proposed method.* The quality of solutions which were obtained for all instances significantly exceeded the quality of solutions found by the employed common heuristic method. Table 1 summarizes the results. The characteristics of each problem are depicted, namely the n parameter and an estimation μ of the average number of "items" that fit in the knapsack, calculated as the ratio of the capacity C to the mean value of the weights in vector $\vec{W}$.

6.3. *Set Partitioning*

The common method that provided the best results for the SP problem was the heuristic policy *"try the column with the minimum cost first"* with DFS. The proposed methodology performed significantly better on all instances. Table 2 depicts the characteristics of the SP instances and the performance of our methodology. The dimensions of the $A_{m \times n}$ binary matrix are shown as the major characteristics for an SP instance.

Six iterations, of 10 minutes each, were performed on each SP instance. Remarkably better solutions were found by the proposed methodology using *LDS*, than those obtained by the common heuristic policy.

6.4. *Behaviour of the Methodology*

The experimentations on the SP and the knapsack problems gave a view of the behaviour of the methodology during the solving process. Figure 5 gives a low level view of a solving path followed by the heuristic function of Fig. 4 for a knapsack in-

*LDS was also tested with common heuristics on both problem instances but did not outperform DFS.

Instance		Performance			
n	m	c_o	ml_o	opt	α
4000	15	1175	1063	898	40.4
4000	18	1474	1401	1113	20.2
4000	20	900	481	452	93.5
5000	18	15698	12563	10669	62.3
5000	25	993	910	544	18.5
5000	30	2720	2538	1734	18.4
6000	20	754	580	407	50.1
6000	25	1148	893	593	45.9
6000	30	1133	1088	779	12.7
7000	25	1128	905	569	39.8

Table 2: Experimental results for the SP problem

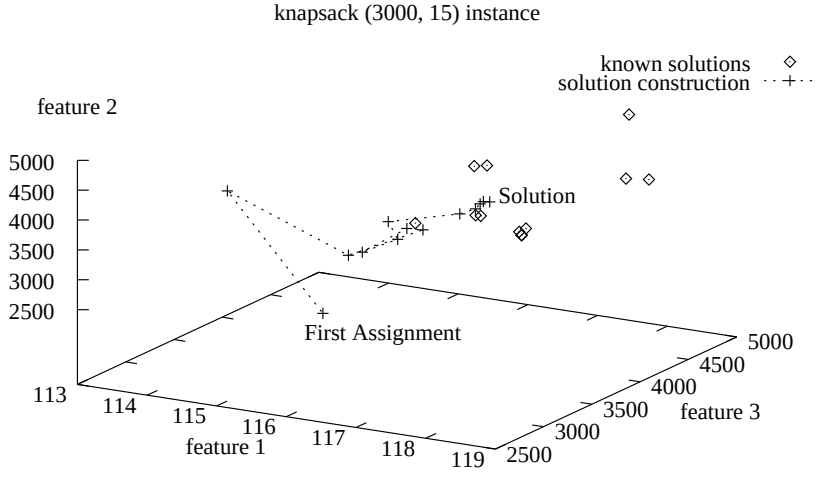

Figure 5: Solution construction path for a knapsack instance

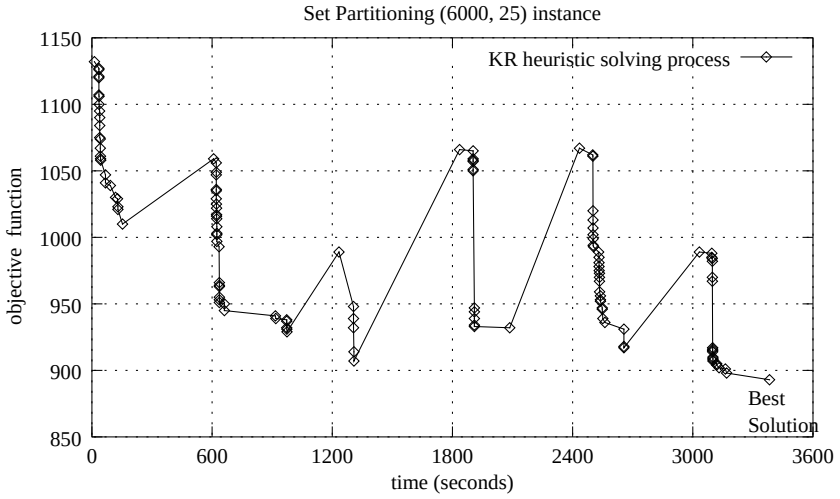

Figure 6: Solving process for an SP instance

stance, whereas the overall solving performance of the proposed iterative methodology is depicted in Figures 6, 7, 8 and 9 for two set partitioning and two knapsack instances respectively.

Figure 5 depicts the features vectors (as independent points in the diagram) for known solutions to the $\langle n = 3000, \mu = 15 \rangle$ knapsack instance that belong to the training set. It also demonstrates the trajectory of a solution construction using the given training set. Every cross point in the trajectory represents a features vector calculated upon a partial path, after a new assignment to some decision variable is performed. At every search step, the heuristic function chooses the assignment which brings the features vector of the extended path closer to specific features vectors of the best known solutions. As is clearly visible from the diagram, although the trajectory in the features space is somewhat awkward, it leads to the construction of a complete solution, with statistical properties (as indicated by its features vector), which locate it within a desirable cluster of known solutions. When the chosen representation features are indeed relevant to the objective function's value, then the newly constructed solution is expected to be of comparable quality to the solutions of the cluster.

The overall functionality of our methodology for the SP problem is demonstrated in Fig. 6 and 7, on the $\langle n = 6000, m = 25 \rangle$ and $\langle n = 4000, m = 18 \rangle$ SP instances. Subsequent iterations of the algorithm in Fig. 1 are separated by the vertical grid lines on the diagrams. Within the iterations each solution should be better than the previous found. The best solution is found in the last iteration for the $\langle n = 6000, m = 25 \rangle$ instance. A natural explanation for this fact is that the *KR*-heuristic provided the most accurate predictions during this iteration, having accumulated in its training set an appropriate representation of the search space during previous iterations. However this was not the case for all instances.

One can notice in Fig. 7 that the best solution was found during the fifth iteration, whereas the system was tragically misled during the subsequent iteration. This is pos-

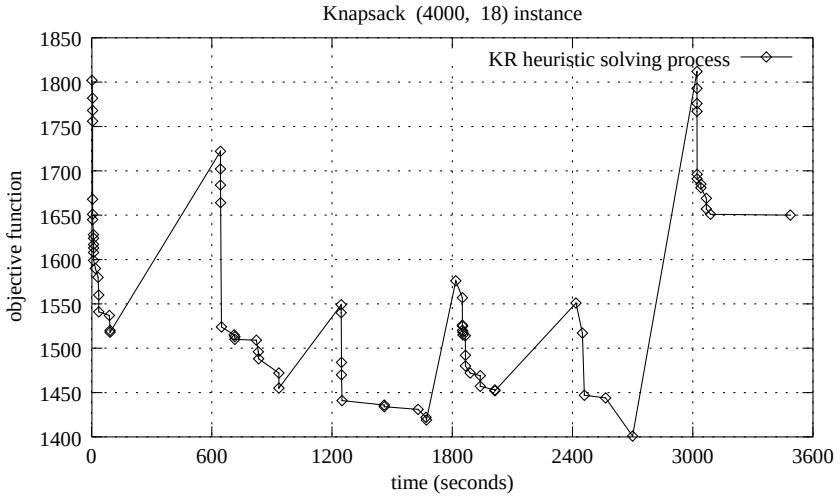

Figure 7: Solving process for an SP instance

sibly due to introduction of inconsistent data in the training set during the previous iteration. However, we expect that the heuristic is able to improve its behaviour in subsequent iterations, since solutions of lower quality are stored in the training set and contribute to future estimations. This fact confirms the need for information acquisition and exploitation regarding the search space, in order to explore its most promising portions.

Figures 8 and 9 depict the solving process for two knapsack instances. Analogous situations to the ones discussed for the set partitioning case can be observed. To be precise, one can certainly notice the heuristic's poor behaviour during the first iterations, and how it subsequently improves towards discovering some good solutions. For both instances, solutions found during iterations after the first 2100 seconds are of comparable quality.

6.5. *Depth-First Search Experiments*

In order to assess further the heuristic value of our methodology, we experimented with it in a pure manner by using the simple DFS procedure. In fact, a multi-restart variant of the original DFS scheme was used: each time a solution is found the search backtracks to the root of the tree. Thus, one can think of this search procedure as drawing multiple depth-first explorations down the tree, starting from the next preferable choice at the root, every time a new solution is found. We shall refer to it as *modified DFS* (*mDFS*). This strategy is not a complete one, in contrast to the previously used *LDS*, and its performance heavily depends on the heuristic, since it encloses a DFS scheme.

As already discussed in section 2, it is generally known that, constructive search heuristics tend to make mistaken decisions on early intermediate stages of the search, that is near the top of the search tree. By enforcing the search procedure to restart from

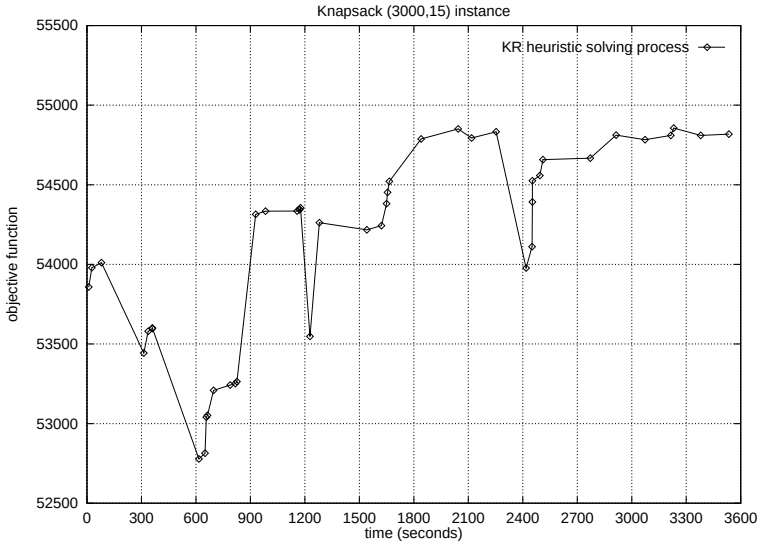

Figure 8: Solving process for a Knapsack instance

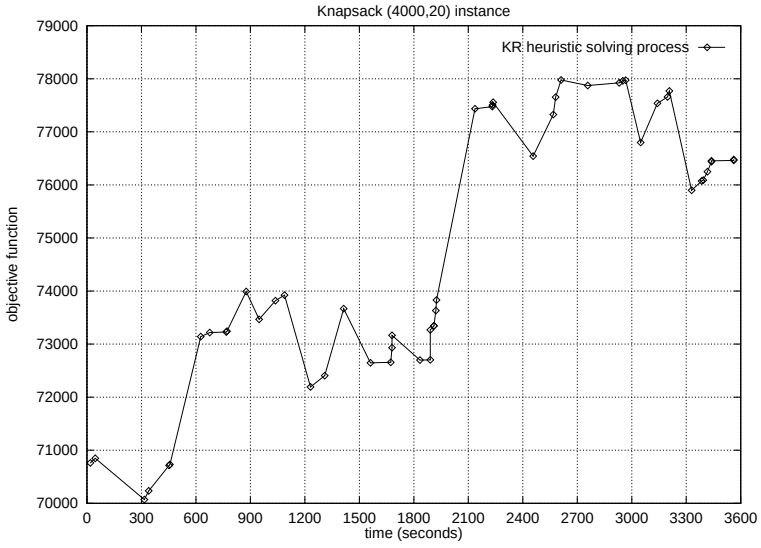

Figure 9: Solving process for a Knapsack instance

Instance		Performance			
n	μ	ml_o	α	$\alpha - \alpha_{LDS}$	$\bar{r}$
3000	15	55299	39.9	12.1	3.7
4000	20	74947	20.9	-63.9	5.1
4000	40	141893	11.3	-11.1	4.2
4000	51	181723	21.6	-48.3	3.4
4000	23	84114	26.7	3.6	5.3
4000	30	109800	39.6	0.2	3.6
6000	30	112421	46.9	17.6	3.2
6000	20	74557	60.4	47.1	3.6
8000	15	57985	48.6	22.2	3.5
8000	23	91141	14.4	-3.5	3.1

Table 3: Experimental results with *mDFS* for the knapsack problem

the next preferable choice from the root (according to the heuristic's decision), we intend to investigate the heuristic's sensitivity to this matter. For a knapsack instance with n decision variables, the meaningful available assignments at the root of the search tree are exactly n ($x_j = 1, j = 1 \ldots n$). The same holds for a set partitioning instance with n decision variables. Regarding the problem instances used in our experiments, this size reaches the order of a few thousands. It is shown that, on average per iteration of the search schema (Fig. 1), trying only a small fraction of these alternatives suffices for reaching high quality solutions. This fact confirms the heuristic's robustness with respect to its early decisions.

For each instance of both problems, the methodology was allowed 12 iterations, of 5 minutes each, that is, a total of one hour. Results are summarized in Tables 3 and 4. Table 3 shows, for each Knapsack instance, the improvement over the greedy heuristic's performance of Table 1. The last column ($\bar{r}$) shows the average number of restarts per iteration, performed by the modified DFS procedure.

The improvements achieved on some instances are unexpectedly impressive. On four out of ten instances the results were poorer than the ones obtained with *LDS*. However, in their entirety we consider them to be quite encouraging, since they only exhibit the heuristic's guiding ability, and still they are significantly improved relative to the greedy heuristic. By combining information from already known feasible solutions, *KR* provided useful heuristic predictions, thus guiding the simple *mDFS* search towards improvements. In fact, we get a confirmation of our initial assumption that improved solutions lie nearby already known solutions, in terms of euclidian distances between their features vectors (we wish to remind the reader of Fig. 5 and the corresponding discussion). The small values of the $\bar{r}$ counter indicate that the heuristic's initial decisions are generally quite informed. It actually exploits only a small fraction over the thousands of choices at the root of the search tree, yet, its performance is at least satisfactory.

The heuristic proved to be quite successful with our Set Partitioning dataset. Solutions found on almost all instances were near-optimal, and greatly exceeded the heuris-

Instance		Performance			
n	m	ml_o	α	$\alpha - \alpha_{LDS}$	$\bar{r}$
4000	15	935	86.6	46.2	8.1
4000	18	1171	83.9	63.7	4.75
4000	20	461	97.9	4.4	5.6
5000	18	10987	93.6	31.3	7.2
5000	25	544	100	81.5	4.8
5000	30	1734	100	81.6	8.1
6000	20	409	99.4	49.3	7.1
6000	25	738	73.8	27.9	6.6
6000	30	1011	34.4	21.7	4.7
7000	25	692	77.9	38.1	9.7

Table 4: Experimental results with *mDFS* for the SP problem

tic's performance with *LDS*. Table 4 summarizes the results. These results convincingly confirm the value of the heuristic and the existence of an almost functional relation between the statistical characteristics (features) of a solution and its quality.

Another aspect we examine concerns the heuristic's behaviour with respect to the underlying training set. In particular, we measure for each iteration the deviation of the best found solution from the best one contained within the training set. Let s_i be the objective function value for the best solution found during iteration i of the algorithm in Fig. 1. Suppose that the best solution contained within the training set of the ith iteration was of quality $s_i^\star$. Then for a maximization problem we define the following improvement ratio on iteration i:

$$IR_i = \frac{s_i - s_i^\star}{U - L}$$

where U and L are the highest and the lowest among the values s_i, $s_i^\star$, $i = 1, \ldots, I$. The quantity IR_i is the fraction, by which the heuristic exceeded (or missed, in case of $s_i < s_i^\star$), its best known solution (always contained within its training set) on iteration i. For a minimization problem the IR_i quantity is defined similarly, but with an opposite sign, in order to preserve it positive when real improvement occurs.

Figures 10 and 11 depict the IR_i measure for three instances of each problem. It would have been an ideal situation, if the curves were kept over zero, thus implying a constant improvement over the best known solution, from one iteration to the next. However, what the reader can actually observe is a satisfactory stability on the method's behaviour, only partially disturbed by great improvements or losses.

Particularly on all three knapsack instances (Fig. 10) improvements on iteration 1 are followed with big losses during iteration 2. The curves continue thereafter with only small positive or negative deviations from the best known solutions. This behaviour implies that, even though the heuristic's predictions may become spurious with respect to the underlying training set of some iteration, accumulation of training examples helps guiding the search more conservatively during subsequent iterations, towards informed solution constructions.

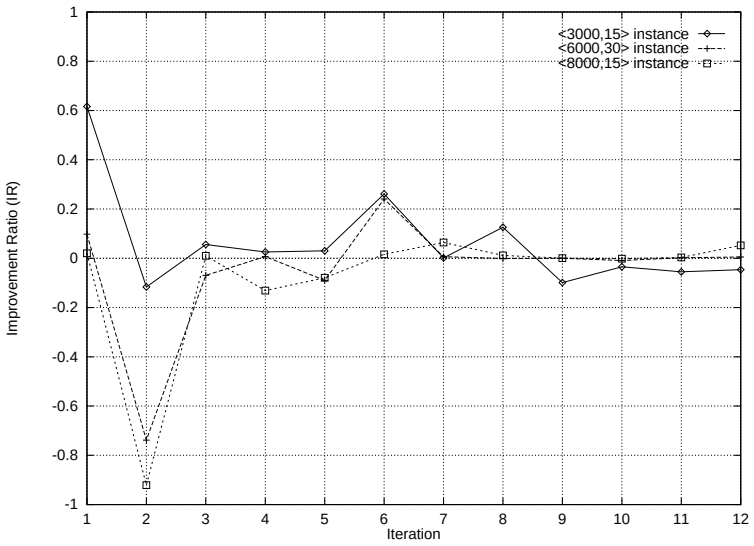

Figure 10: Improvement per Iteration on Knapsack

For the Set Partitioning instances (Fig. 11) there is an analogous behaviour, with the difference of great improvements happening in iteration 1. Subsequent iterations also provided small improvements and losses. Although positive IR leaps are generally desirable since they indicate optimization, we should also feel comfortable with small movements around zero, because they exhibit the existence of smooth functional correspondence between the chosen solution descriptive features and the objective function value. This statement stems from the way the heuristic tries to place the newly constructed solution somewhere in the geometric space defined by its training examples.

7. Conclusions and Further Work

In this paper, we propose a heuristic methodology for combinatorial optimization, which employs instance-based learning and function approximation through kernel regression, for guiding any constructive search procedure. This work is not concerned with the achievement of feasible solutions to a problem (this issue is addressed successfully by sophisticated implementations of constructive search methods, e.g. backtracking), but with the guidance of search to promising regions of the search space, as far as optimality is concerned.

Problem models grown from real world applications usually enclose vast contents of numerical information, which can be statistically handled for the construction of optimized solutions. The objective functions of such problems are generally designed upon desirable facts and dictate the intuitive policy for their optimization. We suggest that known solutions to these problems are represented via statistical information calculated upon each solution's structural constituents. The proposed policy constructs a solution by minimizing its distance (in terms of its statistical properties) from the best (in terms of objective function value) known solutions, which lie nearby.

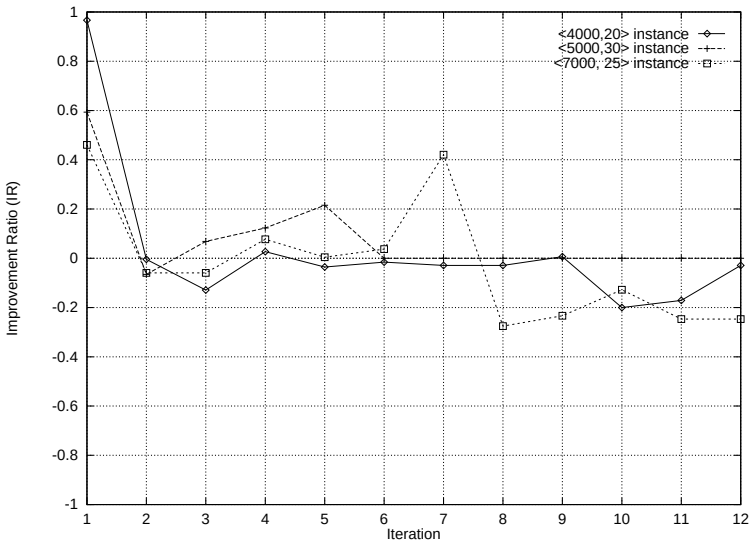

Figure 11: Improvement per Iteration on Set Partitioning

Experimental results were carried out on two widely used models of real world combinatorial problems, namely the knapsack and the set partitioning problems. These problems model important real world applications, such as nuclear waste packing and crew scheduling. The methodology performed satisfactory on these problems and obtained solutions whose quality exceeded the quality of solutions obtained by other heuristic methods, common for each of the problems.

Some directions for further research are drawn from questions that arise quite naturally. In our experiments, the proposed framework performs satisfactory for an initial training set created by some simple methods. *However, which is the proper way for systematically sampling initial solutions of a useful quality distribution from the problem's search space?* This is an important issue, which could possibly boost the performance of the heuristic function, since an initial set of solutions with known quality distribution is actually a detailed picture of the search space.

The complexity of the heuristic function depends on the size of the training set. Using big training sets slows down the search, while small sets provide little information about the search space. We have started to examine the option of partitioning the training set into consistent clusters, each of which represents a small portion of the search space. Each of these clusters is meant to be used as a separate training set, for searching the corresponding sections of the search space in a locally exhaustive manner.

As an aspect of future work, extended experimentations on a variety of optimization problems is expected to reveal valuable statistical features, strongly informative and representative of the corresponding search spaces.

References

[1] Justin A.Boyan and Andrew W. Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:77–112, 2000.

[2] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1–5):11–73, February 1997.

[3] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[4] Justin Boyan, Wray Buntine, and Arun Jagota (Eds.) Statistical machine learning for large-scale optimization. *Neural Computing Surveys*, 3:1–58, 2000.

[5] Justin A. Boyan and Andrew W. Moore. Learning evaluation functions for global optimization and boolean satisfiability. In *Proceedings of the 15th National Conference on Artificial Intelligence AAAI-98*, pages 3–10, Madison, Wisconsin, July 1998. AAAI Press.

[6] Diane J. Cook and R. Craig Varnell. Maximizing the benefits of parallel search using machine learning. In *Proceedings of the 14th National Conference on Artificial Intelligence AAAI-97*, pages 559–564, Providence, Rhode Island, July 1997. AAAI Press.

[7] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence IJCAI-95*, pages 607–613, Montréal, Québec, Canada, August 1995. Morgan Kaufmann.

[8] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[9] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence IJCAI-95*, pages 1137–1145, Montréal, Québec, Canada, August 1995. Morgan Kaufmann.

[10] Steven Minton. An analytic learning system for specializing heuristics. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence IJCAI-93*, pages 922–929, Chambéry, France, August 1993. Morgan Kaufmann.

[11] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[12] Kazuo Miyashita. Learning scheduling control knowledge through reinforcements. *International Transactions in Operational Research*, 7(2):125–138, March 2000.

[13] Robert Moll, Andrew G. Barto, Theodore J. Perkins, and Richard S. Sutton. Learning instance-independent value functions to enhance local search. *Advances in Neural Information Processing Systems*, 11:1017–1023, 1999.

[14] M. J. Realff, P. H. Kvam, and W. E. Taylor. Combined analytical and empirical learning framework for branch and bound algorithms: The knapsack problem. *Artificial Intelligence in Engineering*, 13(3):287–300, July 1999.

[15] Orestis Telelis and Panagiotis Stamatopoulos. Combinatorial optimization through statistical instance-based learning. In *Proceedings of the IEEE 13th International Conference on Tools with Artificial Intelligence ICTAI-2001*, pages 203–209, November 2001.

[16] Wei Zhang and Thomas G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence IJCAI-95*, pages 1114–1120, Montréal, Québec, Canada, August 1995. Morgan Kaufmann.