Domain-Dependent Parameter Selection of Search-based Algorithms Compatible with User Performance Criteria

Biplav Srivastava and **Anupam Mediratta**
IBM India Research Lab
Block 1, Indian Institute of Technology
Hauz khas, Delhi 110016, India
Email: {*sbiplav,anupamme*}*@in.ibm.com*

Abstract

Search-based algorithms, like planners, schedulers and satisfiability solvers, are notorious for having numerous parameters with a wide choice of values that can affect their performance drastically. As a result, the users of these algorithms, who may not be search experts, spend a significant time in tuning the values of the parameters to get acceptable performance on their particular problem domains. In this paper, we present a learning-based approach for automatic tuning of search-based algorithms to help such users. The benefit of our methodology is that it handles diverse parameter types, performs effectively for a broad range of systematic as well as non-systematic search based solvers (the selected parameters could make the algorithms solve up to 100% problems while the bad parameters would lead to none being solved), incorporates user-specified performance criteria (ϕ) and is easy to implement. Moreover, the selected parameter will satisfy ϕ in the first try or the ranked candidates can be used along with ϕ to minimize the number of times the parameter settings need to be adjusted until a problem is solved.

Introduction

Search is the basis for problem solving in some of the hardest problems in AI like planning, scheduling and satisfiability. But the search-based algorithms for these problems are notorious for having numerous parameters, many with a wide choice of values, that can affect their performance drastically. For example, the Walksat satisfiability (SAT) solver has one parameter for heuristic strategies with 9 possible values, a second parameter for noise which can take any fractional value from 0 to 1 and a third parameter for number of times to restart the search which can take any whole number. The Blackbox planner uses Walksat as one of its SAT solvers and provides its own additional parameters (e.g., -noexcl) for user customization that can affect its performance.

Having enormous flexibility in parameter settings may seem like a good algorithm design principle intuitively because the algorithm can adapt to problem characteristics. But in practice, it is a problem not only for the developers of the algorithms, who must now spend significant time in tuning the values of the parameters to get acceptable/optimal performance of the algorithm on a particular problem domain, but a major reason for inaccessibility of the search algorithms to other informed users who want to use the algorithms in their applications but are not search experts. In the case of Blackbox, based on first-hand experience and from some reports in literature(Howe *et al.* 1999), we can say that hardly anyone who uses Blackbox for comparative study of planners explores its various parameter settings, preferring to use the default settings in most cases. An illustration of the variation of the performance with parameter setting is shown in Figure 1. The X axis shows the different possible settings while the Y axis shows the number of problems solved. The multi-dimensional parameters are mapped to a single dimension for the viewing convenience. Note that though Blackbox is known to perform well in Logistics, still it cannot solve a single problem in it for half of the parameter settings, where 'NoExcl' is enabled.

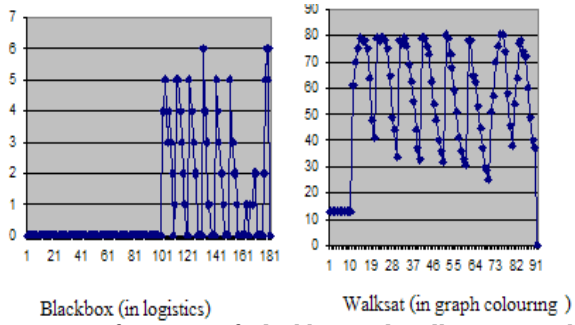

Figure 1: Performance of Blackbox and Walksat across their different parameter settings.

The problem gets compounded when the user is not a subject matter expert. For example, we are building a family of planners in Java based on a common core of programmatic interfaces and common implementation that is meant for a wide-variety of applications. The currently implemented planners are a classical (heuristic search space) planner, a metric planner, a contingent planner and an Hierarchical Task Network planner. At the last count, 8 different types of users wanted to use the same family of planners. In the planning application of web services composition, the user wants to select a simple type of planning initially (e.g., classical planning) but would like to easily switch to a more

complex form (e.g., contingent planning) as they gain experience with planning methods. Another type of user wants to explore planning in system self-management, a.k.a. autonomic computing. Yet another user wants to explore planning for data transformation while collecting sensor data. The users want to use the planners but do not want to understand the rich choice of parameter settings like search direction, heuristic function, weight, that each type of planner comes with.

As the above examples show, search-based techniques currently require the potentially large users to know more about them in order to set good algorithm parameters for their applications. In most of the usage scenarios, the user may not be interested in finding the most optimal parameter setting specific to a problem but a parameter setting which solves most of the problems *in their domain of interest*, within some performance criteria. In this paper, we try to take parameter customization away from the users with a Machine Learning (ML) based approach for automatic parameter selection that is easy to implement and quite effective. We assume that the algorithm can solve some of the problems from the domain under the given performance constraints. The parameter selection procedure is based on learning of parameters by using ML classifiers viz. Decision Tree, Naive Based Learning. In a classification problem, the data available for learning is divided into Training Data and Test Data. The training data is used by the classifier to learn a hypothesis function that can be used to label the test data.

In our solution, we classify the different parameter settings into positive and negative examples with respect to the performance criteria and available training data, and use positive part of the learnt model in conjunction with ranking functions to derive candidate parameter settings. The benefit of our methodology is that it finds a parameter setting that will satisfy the performance criteria in the first try or uses the ranked candidates to minimize the number of times the parameter settings need to be adjusted until a problem is solved. We have tested this approach on two types of search algorithms: sat solvers and planners, and found the proposed methodology to be very effective - on many problem distributions, there are parameter settings which would cause little or no solution being found while the proposed method would automatically find settings that could solve up to 100% of its problems within the performance criteria.

The contributions of the papers are:

1. We formulate a domain specific, parameter setting selection problem for search-based algorithms.
2. We present a solution that is effective for a broad category of systematic and non-systematic solvers, and is easy to implement.
3. The approach can work with diverse parameter types - numeric, categorical and ordinal.
4. The approach allows user to provide performance criteria that is used to derive parameter settings.
5. The approach is independent of any specific classifier though we use decision tree for illustration.

Symbol	Description
$\vec{i} \in X$	A problem instance of A.
$o \in Y$	The output of A on a problem instance.
$\theta \in \Theta$	A vector of parameters for A.
$A(\vec{i}, \theta)$	A search-based algorithm, A. Input: a problem instance $\vec{i}$ and parameter vector θ. Output: o.
$A_p(\vec{i}, \theta)$	The runtime performance of $A(\vec{i}, \theta)$. It is a metric about the performance of A on a problem instance. E.g. 2 min runtime and 5 MB memory.
ϕ	A boolean expression specifying the user criteria of acceptable performance. E.g. $\prec$ 5 min runtime and $\prec$ 20 MB memory.
$l \in L$	The label assigned to A's performance. L = [True, False].
h $\in H$	A hypothesis function mapping parameter instances to labels. h: $X \times \Theta \rightarrow L$.
h' $\in H'$	A hypothesis function mapping parameter instances, *in a problem domain*, to labels. h': $\Theta \rightarrow L$.
$f'(A_p(., \theta), \phi)$	A problem-domain specific function labeling the performance of A with respect to ϕ. Input: Runtime performance of $A(\vec{i}, \theta)$ and ϕ Output: a label, $l \in L$.
$T \in T$	A training set. T = $\{(\theta, l) \mid \theta \in \Theta, l \in L)\}$.
T'	Model Data: A learnable subset of T. $T' \subseteq T$
$M(T')$	A classification algorithm for parameters in a problem domain. Input: a learnable training set. Output: A hypothesis function. $M : T' \rightarrow H'$.
$N(\theta, T)$	A function which maps the θ to ♯ problems solved (within ϕ)in the data set T.
$rank$	A function which assigns a number to every $\theta \in h'$ based on its performance on T. Input: h' and T. Output: $\Re$. $rank : H' \times T \rightarrow \Re$

Table 1: Notation used in the paper.

In the next section, we formalize the problem of automatic tuning of parameters. Then we present our ML based solution and demonstrate its effectiveness empirically across Blackbox and Planner4J planning systems, and the Walksat SAT solver. Next, we discuss related work, put our approach's perspective and conclude.

Domain-specific Parameter Selection Problem

The output o of a search-based algorithm A depends on its input $\vec{i}$ and the parameter vector θ (see Table 1). The runtime performance of A is indicated by the metric A_p and it depends on both $\vec{i}$ and θ.

The domain-independent parameter setting selection problem is to find a parameter vector θ in Θ by learning in the hypothesis space H to find h that satisfies a performance criteria ϕ. We are interested in the domain-specific parameter selection problem (see Figure 2). We wish to find a parameter setting θ^* which solves most of the problems, *in a domain of interest*, within some performance criteria ϕ. Hence, we learn in the hypothesis space H' to find h' which depends only on Θ. Formally, θ^* satisfies

$$\theta^* \quad = \quad \underset{\theta \in \Theta}{argmax}\ rank(M(T'), T) \qquad (1)$$

In equation 1, $rank$ assigns a real number to each θ based on their performance on training data set T. And then the best ranked θ(s) are selected. Now, based on whether T is available up front or problem instances in the domain appear one by one (which will be collected and used to build T), the parameter selection problem is offline or online, respectively. To estimate the maximum gain we get by using

θ^*, we define *lift*:

$$\text{lift}(\theta^*, T) = \frac{N(\theta^*,T) - N(\theta_\perp,T)}{N(\theta^*,T)} * 100$$

That is, *lift* is the increase in the number of problems solved in T by the setting of a methodology compared to the worst performing setting , expressed as a percentage. Lift of 100% means that $\theta_\perp$ cannot solve a single problem and 0% means that it can solve as many problems as θ^*.

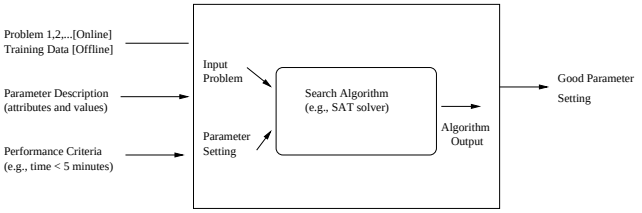

Figure 2: The Problem Setting for Automatic Tuning of Search Algorithms.

Solution Approach

We first discuss how the method works in the offline scenario and how the method can be used to minimize the number of parameter adjustments needed to solve a given problem (assuming a parameter setting will solve it). Then, we present the online scenario. In our running example, we will use the Planner4J classical planner as the algorithm whose parameters are to be configured and the problem domain will be Transport(Helmert 2001). This is not a single planning domain but a collection of domains which involve transportation of things or people like Gripper, Logistics, Miconic, etc.

Offline Parameter Selection:

In the offline scenario, the training data T is available upfront. While classifying parameters, just like any other classification task, one needs both positive and negative examples (Lerman & Minton 2000). We call a problem instance learnable if it has both examples for Θ.

Choosing model data (T'): We select the learnable problem instances from T to form T' as shown in Figure 3. The parameter selection results vary slightly with the different relative % of the examples but it depends significantly on whether both the types of examples are present or not.

For Planner4J, the parameters which affect its search performance are heuristic, heuristic weight and search direction. We experimented with every possible combination of heuristic (0-6), weight (1 - 5) and direction (Forward, Backward). The performance criterion (ϕ) we used was: if a plan was found in 5 mins then the parameter setting is positive otherwise, it is negative. For Transport domain, the training data consisted of problems from domains elevator and gripper. Total number of problems in T was 30 out of which only 20 were learnable.

Generating Parameters: We have a three step solution approach (Figure 4). First, we select a classification method (e.g., decision tree, neural networks, bayesian approach) and use model data T' to build a classifier (hypothesis function

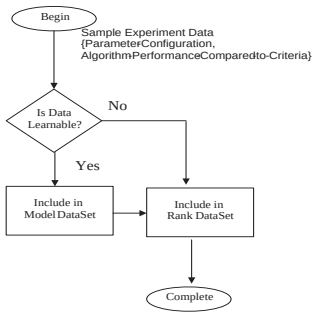

Figure 3: Processing offline sample data into model and ranking data.

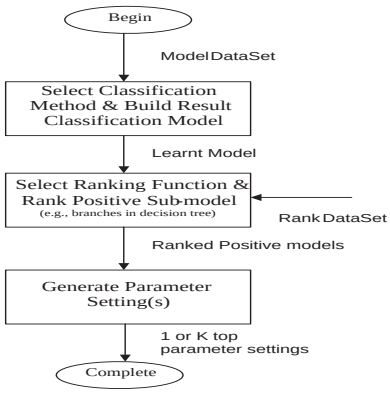

Figure 4: Outline of method to generate parameter settings.

h'). This classifier classifies the complete space of parameter settings Θ into positive and negative instances. The positively classified instances are then ranked on training/rank data T.

We chose the J48 decision tree implemented in Weka[1] as the classifier for our experiments. In a decision tree, the classification model output is a tree like structure. In our case, every node of the tree corresponds to a parameter and the branches resulting from that node are owing to different values that the parameter can take. Each leaf node corresponds to a group of $\theta's$ (a branch) and if its label is $TRUE$, then it is positive else it is negative. Figure 5 shows the decision tree we got in the case of our running example. Against every label, there are two numbers in the form (X'/Y'), where X' is the number of instances that reach to that leaf and Y' is the incorrectly classified instances for that leaf node (if Y' is absent, it means that Y' is 0). As X' and Y' are calculated on T', X and Y have the similar meaning but are calculated on T.

In the example tree, the root node is the parameter *direction* and two branches originating from it are corresponding to the value forward and backward. The leaves in the tree are labeled from $R1$ to $R8$. Among the rules, the positively classified rules are: $R1, R3, R5$ and $R6$. We have formulated two ranking functions which we use to rank the positively classified rules on T or T'. The ranking functions

[1] http://www.cs.waikato.ac.nz/ml/weka/

```
Direction = Forward
  | Hvalue = 0: TRUE (100.0/33.0) → R1
  | Hvalue = 1: FALSE (100.0/35.0) → R2
  | Hvalue = 2: TRUE (100.0/22.0) → R3
  | Hvalue = 3: FALSE (100.0/35.0) → R4
  | Hvalue = 4: TRUE (100.0/23.0) → R5
  | Hvalue = 5: TRUE (96.0/20.0) → R6
  | Hvalue = 6: FALSE (90.0/37.0) → R7
Direction = Backward: FALSE (700.0/26.0) → R8
```

Figure 5: *Decision Tree (P4J in Transport).*

are[2]:

1. $\forall\theta \in \Theta, Rf_1(h', T') = (X'_\theta - Y'_\theta)/X'_\theta$
2. $\forall\theta \in \Theta, Rf_2(h', T) = ((X'_\theta - Y'_\theta) + \Delta_\theta) \,/\, (X'_\theta + \Delta_\theta)$
3. $\forall\theta \in \Theta, Rf_3(h', T') = (X'_\theta - Y'_\theta) \,/\, \sum_{\forall\theta}(X'_\theta - Y'_\theta)$

Rf_1 estimates the probability of success for θ based on T'. Rf_2 differs from Rf_1 since the ranking is done based on the non-learnable data set (Δ_θ) also. Since all parameters of the latter were positive (or negative), a constant factor is added to both sides of the fraction. Rf_3 differs from Rf_1 due to the change in the denominator which is the sum of the total number of instances in T' that are labeled as *TRUE*. While Rf_1 (and Rf_2) give more weightage to absolute accuracy of the branch, Rf_3 values accuracy relative to other branches in the model.

To illustrate, suppose T consists of 3 problem instance on 100 parameter configuration of which 2 problems were learnable (T' and T are of sizes 200 and 300, respectively). Suppose there are 2 positive and 3 negative branches in the model: one positive branch has accuracy (4) and another has accuracy (100/5). By Rf_1, their ranked values are 1 ((4-0)/4) and 0.95 ((100-5)/100), respectively, while by Rf_3, they are 0.04 ((4-0)/(4+95)) and 0.96((100-5)/(4+95)). The Rf_2 values, assuming there are 2 and 50 θs of unlearnable data in each of the positive branches, respectively, would be 1 (4+2/4+2) and 0.97 (95+50/100+50).

The branch for which the value is higher is better ranked. Since a branch contains multiple θ, we can randomly pick one θ from the branch. To find top K $\theta's$, we select top K ranked branches and randomly pick a θ from each of them.

For the running example, selected $\theta's$ by the application of Rf_2 are shown in Table 2. When we apply Rf_3, the top three branches are: *R6*, *R3* and *R5*. We randomly pick one θ from each of them and they are shown in the Table. Note that we are illustrating the approach with specific classifiers/ranking functions but it can be easily extended to the new ones as desired by the user of the approach.

Minimizing Parameter Adjustments

By using the above methodology, we find the θ which either solves the problem within performance constrain on the first try or minimizes the number of times we have to adjust the θ to get the problem solved. Figure 7 shows the performance ($\sharp$ solved problems) for each θ for the running example. The X axis shows all the possible $\theta's$ and the Y axis shows the $\sharp$ solved problems corresponding to each θ. If we consider

[2]Though the functions are defined for a θ, they can be naturally extended to a branch (classifier dependent concept) of the model.

Domain/ Ranking fn	Heuristic	Weight	Direction	Ranked value	Lift (%)	Comb. Succ.(%)
Transport/ Rf_2	0	5	Forward	0.97	99.47	96.66
	2	2	Forward	0.87	99.38	
	2	3	Forward	0.87	99.38	
Transport/ Rf_3	5	1	Forward	0.79	100	80
	2	1	Forward	0.78	100	
	4	1	Forward	0.77	100	
Blocksworld/ Rf_2	0	5	Forward	0.93	99.83	93.33
	0	4	Forward	0.93	99.83	
	0	3	Forward	0.93	99.83	
Blocksworld/ Rf_3	0	3	Forward	0.87	100	93.33
	0	2	Forward	0.87	100	
	0	1	Forward	0.87	100	
Movie/ Rf_2	0	1	Forward	1.0	100	100
	0	2	Forward	1.0	100	
	0	3	Forward	1.0	100	
Movie/ Rf_3	0	1	Forward	1.0	100	100
	2	1	Backward	1.0	100	
	4	1	Backward	1.0	100	

Table 2: Ranking, lift and combined problems solved for Planner4J for selected $\theta's$.

$\theta's 36$ to 70, which correspond to direction backward and the other two parameters are 'don't care', we would try 35 possible values of θ and still would be able to solve only 18 out of 30 problem instances in T (60% success). Alternatively, if we use our methodology, we try at most 3 values of θ for each problem instance and are able to achieve 96.66% (Rf_1) or 80% (Rf_2) success with respect to ϕ.

Another advantage of our methodology is that when we are using a θ which violates ϕ, we need not wait for the search algorithm A to terminate to try another setting. Instead, we can stop the execution whenever ϕ gets violated. E.g., in a particular scenario, if we need to try K $\theta's$ to find an acceptable θ and ϕ is say $\prec 5$ min runtime, then the maximum time we would need is $5 \times K$ mins.

Online Parameter Selection

This method is for the scenario where no training data T is available upfront. Whenever a problem instances $\vec{i}$ arrive, we follow the procedure laid out in Figure 6. Every $\vec{i}$ is added in T and if it is learnable then it is included in T'. If T' is not empty, then we follow the parameter generation part of the offline procedure, else a default parameter setting is output. The parameter selection may proceed until the selection has stabilized or it may continuously run with a constant size of problem instances seen on the past. We omit examples due to space constraints.

Default Parameter Selection: Here is one way to generate the default setting: on all the problem instances collected, run the search algorithm A for all possible $\theta's$ and classify each instance as positive or negative according to ϕ. Then for each parameter, we select that value (among all values the parameter can take), for which the number of positive instances is maximum. The values of all the parameters is the default θ.

Experiments

We want to check that the proposed method works for algorithms based on systematic and non-systematic search, with different types of parameters and yet is effective (high *lift*).

Experimental Setting: We have tested our methodology on two planners: Planner4J and Blackbox and a sat solver:

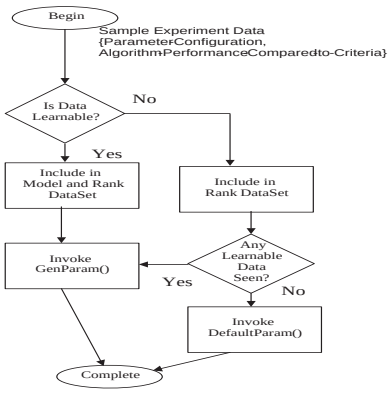

Figure 6: Processing online sample data and generating parameters.

Walksat. Parameters of Planner4J have been stated in earlier section; parameters for Walksat we considered are heuristic and noise. In Walksat, there are 9 different heuristics: random, best, tabu 1-5 (5), novelty, rnovelty; noise varies from 1 to 100 and we discretized it with a step of 10. Planner Blackbox converts the planning problem into a sat encoding and then uses a sat solver to find if the plan exists. In our experiments, the Blackbox parameter which affects its performance is "noexcl" which controls the set of exclusiveness relationships created in the encoding and the parameters of Walksat sat solver. The domains used for each of the solvers are:

1. Planner4J: Transport, Blocksworld, Movie[3].
2. Blackbox: Logistics, Blocksworld
3. Walksat: Controlled Backbone Models (CBS), Graph Colouring (GC) [4]

Planner4J: Its performance on Transport is already discussed in solution approach. Results for rest of the domains follows. *Blocksworld:* In this domain, the task is to find a plan to arrange blocks in a goal configurations starting from an initial arrangement. The sample T consisted of data corresponding to 15 problem instances and T' consisted of data corresponding to 12 problem instances (learnable). *Movie:* In this domain, the task is to get items like popcorns, chips and rewind the movie. In the training data, we had 17 problem instances off which all satisfied the criteria to be in model data.

On these problems, we apply our methodology and show the results in Table 2. We find that we can solve up to 100% problems with the top-3 settings while there exist extreme settings which would not solve any problem in the given ϕ. Figure 7 shows the contrast in the performance of different $\theta's$. The methodology is able to discover all the peaks. The results carry over to unseen problems (test data) as well. For example, in Blocksworld, we tested the parameter selections on randomly generated new problems. The combined success of $\theta's$ obtained from Rf_2 was 75% and that of Rf_3 was 62.5%.

[3]http://cs.toronto.edu/aips2000/.

[4]http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html

System/Domain/ Ranking fn	Heuristic	Noise	No Excl	Ranked value	Lift (%)	Comb. Succ. (%)
Blackbox/	tabu 1	10	No	0.67	100	88.89
Logistics/	tabu 1	20	No	0.67	100	
Rf_2	best	30	No	0.67	100	
Blackbox/	rnovelty	60	No	0.86	100	55.56
Logistics/	tabu 4	20	No	0.79	100	
Rf_3	tabu 1	50	No	0.77	100	
Blackbox/	novelty	50	No	0.40	100	40
Blocksworld/	novelty	60	No	0.40	100	
Rf_2	tabu 2	10	No	0.40	100	
Blackbox/	tabu 2	10	No	1.0	100	40
Blocksworld/	novelty	40	No	0.92	100	
Rf_3	rnovelty	60	No	0.81	100	
Walksat/	best	50	-	1.0	99.91	100
CBS/	best	20	-	1.0	99.91	
Rf_2	best	30	-	1.0	99.91	
Walksat/	tabu 1	10	-	1.0	100	100
CBS/	tabu 4	10	-	1.0	100	
Rf_3	tabu 3	10	-	0.99	100	
Walksat/	tabu 4	10	-	0.99	99.81	98.75
GC/	novelty	50	-	0.88	99.81	
Rf_2	novelty	60	-	0.76	99.81	
Walksat/	tabu 1	60	-	0.97	100	95
GC/	tabu 2	60	-	0.91	100	
Rf_3	best	80	-	0.89	100	

Table 3: Ranking, lift and combined problems solved for selected $\theta's$ for Blackbox and Walksat.

We also tested the methodology by changing the definition of learnable data. We defined a problem instance as learnable if at least k% of parameters instances were of both positive and negative examples. The higher the k, the more discriminating is the problem instance. We found that in Blocksworld, with k as 24%, 5 problems were in T' and the selected parameters could still achieve the same performance. We do not explore k further.

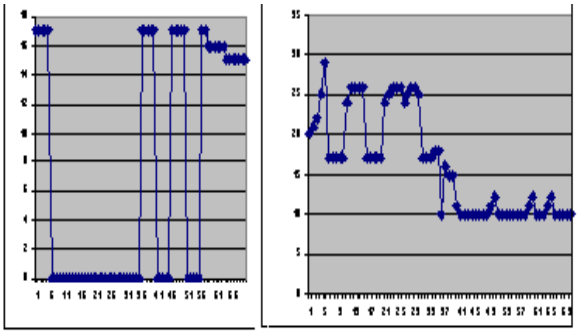

Figure 7: Performance of various settings of Planner4J across different domains.

Blackbox: We tested this planner on two domains: Logistics and Blocksworld. Number of problems in T and T' for Logistics were 9 and 7 respectively. The same for Blocksworld were 10 and 4 respectively. The selected parameter settings for both of the domains by both of the ranking functions and their performance are shown in Table 3. Blocksworld is known to be a hard domain for Blackbox and still, we are able to solve 40% of the problems (the maximum possible with any parameter setting) by parameter discovery.

The results again carry over to unseen problems. In Logistics, on the unseen data, Rf_2 gives success of 81.81% and Rf_3 is successful on 88.89% of data.

Walksat: We tested the Walksat sat solver on two domains: CBS and GC. In CBS, *T* and *T'* contain 24 and 22 problems, respectively. In GC, *T* and *T'* contain 80 and 67 problems, respectively. Table 3 shows the selected $\theta's$ and performance for both of the domains. In CBS, settings produced by both ranking functions give 100% success on the sample T. In GC, the settings gave 98.75% and 95% success, respectively.

Summary: We have shown that the methodology is effective across different planners and sat solvers (in terms on number of problems solved and % lift on training data (T) and unseen data). Since Planner4J employs A^* search (systematic) and Walksat employs a local search method (non-systematic), we can say that the methodology is effective on both kind of searches. Second, for a search algorithm, acceptable parameter settings vary across domains so a domain dependent methodology indeed is helpful to a focused user. Third, the performance difference between the selected θ^* and the worst performing $\theta_\perp$ is significant in most of the domains, so there is a need of parameter tuning in the kind of search algorithms we have considered.

Discussion and Related Work

Literature on parameter setting selection fall under two category: (a) techniques that tune arbitrary algorithms and (b) techniques that tune specific algorithms. Our focus is on tuning search-based algorithms and this category covers much of the harder and practical problem solving in areas like scheduling, satisfiability, planning and optimization.

Calibra(Adenso-Diaz & Laguna 2004) is a system that falls under category (a). It is a design-of-experiment based approach where Taguchi's fractional factorial experiment designs coupled with a local search procedure are used to select parameters. It can optimise up to five numeric parameters and the best values are not guaranteed to be optimal.

In category (b), we are not aware of any method for search algorithms that works with every type of parameters. Auto-Walksat(Patterson & Kautz 2001) is a algorithm which automatically tunes any variant of the Walksat family of stochastic satisfiability solvers. It employs a heuristic search to find the optimal value of the parameter. The results show that it works well for these kind of solvers which generally have one search parameter that is noise. In (Kohavi & John 1995), an approach is presented to automatically select parameters of learning algorithms by searching the space of parameter configurations with an aim to get optimum learning accuracy. In (Consens *et al.* 2005), the authors talk about how cumulative frequency curves can be used to rank database configuration techniques but they do not cover how the configurations are generated. In (Kohavi & John 1998), the wrapper method is developed for searching for feature subsets that still maintains or improves classification accuracy. We can use their techniques during the classification step. There has been work on automatic tuning of web servers like Apache based on control theory (Parekh *et al.* 2001). The problem is seen as designing a controller by first constructing a transfer function which relates past and present input values to past and present output values, and the controller is used to predict behavior. The methods work on numeric parameters only and do not scale.

There has been some work on meta-algorithms for search problem which can be seen as related to parameter selection. In (Howe *et al.* 1999), the authors compare 6 planners on different planning problems and use regression to learn problem features that could be used to predict the planner performance. They note that either a planner can quickly solve a problem in a domain or it generally fares very poorly on the domain. Our results validate this observation and finds it to be true for general search algorithms because our learning-based method is able to select very good parameter settings with very little sample problem instances.

Conclusion

We presented a general method for domain-dependent parameter selection of search-based algorithms that can handles diverse parameter types, performs effectively for a broad range of solvers, incorporates user-specified performance criteria (ϕ) and is easy to implement. It can be used for acceptable parameter selection, minimizing parameter adjustments and improving runtime performance.

References

Adenso-Diaz, B., and Laguna, M. 2004. Fine tuning of algorithms using fractional experimental designs and local search. In *http://leeds-faculty.colorado.edu/laguna/articles/finetune.pdf*.

Consens, M. P.; Barbosa, D.; Teisanu, A. M.; and Mignet, L. 2005. Goals and benchmarks for autonomic configuration recommenders. In *Proc. SIGMOD Conference*.

Helmert, M. 2001. On the complexity of planning in transportation domains. In *Proceedings of the 6th European Conference on Planning (ECP'01)*, 349–360.

Howe, A. E.; Dahlman, E.; Hansen, C.; Scheetz, M.; and Mayrhauser, A. 1999. Exploiting competitive planner performance. In *Proc. ECP, Durham, U.K.*

Kohavi, R., and John, G. H. 1995. Automatic parameter selection by minimizing estimated error. In *Proc. 12th International Conference on Machine Learning.*

Kohavi, R., and John, G. H. 1998. The wrapper approach. In *Feature Selection for KDD. H. Liu and H. Motoda (eds.), Kluwer Acad. Pub., pp33-50.*

Lerman, K., and Minton, S. 2000. Learning the common structure of data. In *Proc. 17th AAAI*, 609–614. AAAI Press / The MIT Press.

Parekh, S.; Hellerstein, J.; Jayram, T.; Gandhi, N.; Tilbury, D.; and Bigus, J. 2001. Using control theory to achieve service level objective in performance management. In *Proc. IM, Seattle, WA, May.*

Patterson, D. J., and Kautz, H. 2001. Auto-walksat: A self-tuning implementation of walksat. In *Proceedings of SAT2001: Workshop on Theory and Application of Satisfiability Testing*, volume 9. Elsevier.