*Appears in the Proceedings of the Tenth International Conference on Machine Learning,
Amherst, MA, June 1993, pp. 135–142.*

Learning Search Control Knowledge for Deep Space Network Scheduling

Jonathan Gratch[*], Steve Chien[+], and Gerald DeJong[*]

*Beckman Institute
University of Illinois
405 N. Mathews Av., Urbana, IL 61801
{gratch, dejong}@cs.uiuc.edu

+Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109-8099
chien@aig.jpl.nasa.gov

Abstract

While the general class of most scheduling problems is NP–hard in worst–case complexity, in practice, for specific distributions of problems and constraints, domain–specific solutions have been shown to perform in much better than exponential time. Unfortunately, constructing such techniques is a knowledge–intensive and time–consuming process that requires a deep understanding of the domain and the scheduler. The goal of our work is to develop techniques to allow for automated learning of an effective domain–specific search strategy given a general problem solver with a flexible control architecture. In this approach, a learning system searches a space of possible control strategies, using statistics to evaluate performance over the expected problem distribution. We discuss an application of the approach to scheduling satellite communications. Using problem distributions based on actual mission requirements, our approach identified strategies that both decrease the amount of CPU time required to produce schedules, and increase the percentage of problems that are solvable within computational resource limitations.

1 INTRODUCTION

General problem solving tasks like planning and scheduling are inherently complex. Nevertheless, in many practical situations these complex problems have reasonable solutions (e.g. traveling salesman problem [Held70]). Often we can take advantage of the structure of a domain or the distribution of problems to formulate effective solutions to complex problems. Unfortunately, a system designer must devote considerable expense to the performance aspects of an algorithm.

In this article we investigate the use of a machine learning approach to automatically improve a performance element with respect to a specific domain and distribution of problems. The performance element must be flexible, meaning there are control decisions that effect search that may be resolved in alternative ways. The overall learning problem we are addressing can be specified as follows. Given a flexible performance element *PE* with control points $CP_1 \ldots CP_n$, where each control point CP_i corresponds to a particular control decision and for which there is a set of alternative decision methods $M_{i,1} \ldots M_{i,k}$,[1] a *control strategy* is a selection of a specific method for every control point (e.g, $STRAT = \langle M_{1,3}, M_{2,6}, M_{3,1}, \ldots \rangle$). A control strategy determines the overall behavior of the scheduler. It may effect properties like computational efficiency or the quality of its solutions. Let *PE*(*STRAT*) be the problem solver operating under a particular control strategy. The function *U*(*PE*(*STRAT*), *d*) is a real valued *utility function* that is a measure of the goodness of the behavior of the scheduler over problem *d*. The goal of learning can be expressed as: given a problem distribution *D,* find *STRAT* so as to maximize the *expected utility* of *PE.* Expected utility is defined formally as:

$$\sum_{d \in D} U(PE(STRAT), d) \times probability(d)$$

For example, in a planning system such as PRODIGY [Minton88], when planning to achieve a goal, control points would be: how to select an operator to use to achieve the goal; how to select variable bindings to instantiate the operator; etc. A method for the operator choice control point might be a set of control rules to determine which operators to use to achieve various goals plus a default operator choice method. A strategy would be a set of control rules and default methods for every control point (e.g., one for operator choice, one for binding choice, etc.). Utility might be defined as a function of the time to construct a plan.

Our proposed solution to this learning problem, sometimes called the *utility problem* [Minton88], is embodied in the COMPOSER system. COMPOSER can be characterized as a hill-climbing search in the space of possible strategies. The learning system alternately conjectures changes to the current control strategy and statistically evaluates them to determine how well they enhance expected utility.

---

[1] Note that a method may consist of smaller elements so that a method may be a set of control rules or a combination of heuristics.

In this paper we describe an application of the COMPOSER framework to learning search strategies for an automated scheduling technique in a NASA domain of space-craft communication scheduling. Two important aspects of this evaluation are that the task and problem distribution are based on a real-world situation, and the scheduling approach was developed independently of our learning work [Bell92]. The implementation includes a novel approach for improving learning efficiency. The performance of the system, along with previous results in artificial planning domains [Gratch91, Gratch92], demonstrates COMPOSER's flexibility and its potential to identify beneficial knowledge in practical learning problems.

2 COMPOSER

COMPOSER is a statistical approach to improving the expected utility of problem solving. The overall approach is one of generate and test hill-climbing. Given an initial problem solver, a *transformation generator* constructs a set of possible transformations to the control strategy. Each of these changes is evaluated statistically over the expected distribution of problems. A transformation is adopted if it increases the expected utility of solving problems over that distribution. The generator then constructs a set of transformations to this new strategy and so on. For a complete description of the method see [Gratch92]. The algorithm is summarized in the Appendix.

COMPOSER's solution is applicable in cases where the following conditions apply.

1. The control strategy space can be structured to facilitate hill-climbing search. In general, the space of such strategies is so large as to make exhaustive search intractable. COMPOSER requires a transformation generator that structures this space into a sequence search steps, with relatively few transformations at each step. In Section 3.4 we discuss some techniques for incorporating domain specific information into the structuring of the control strategy space.

2. There is a large supply of representative training problems so that an adequate sampling of problems can be used to estimate expected utility for various control strategies.

3. Problems can be solved with a sufficiently low cost in resources so that estimating expected utility is feasible.

4. There is sufficient regularity in the domain such that the cost of learning a good strategy can be amortized over the gains in solving many problems.

COMPOSER can be seen as one of a class of statistical approaches to improving the expected utility of problem solving (see also [Greiner92, Laird92, Subramanian92]. The principle drawback of these techniques is that they find only local maxima, they may require many examples, and examples can be expensive to process. Furthermore, their statistical properties rest upon assumptions that may not hold in practice. Greiner and Jurisica demonstrate that, under very weak assumptions, the number of examples, or *sample complexity,* can be bounded by a polynomial function of the allowable statistical error [Greiner92]. Weak assumptions are a way of ensuring wide applicability. Unfortunately, the resulting bounds prove too large for most practical applications (see [Buntine89, Gratch92]). COMPOSER embodies stronger statistical assumptions suggested by the Central Limit Theorem [Hogg78 pp. 192–195]. These stronger assumptions drastically reduce the number of examples required to make statistical decisions, but they limit the applicability of the approach. In previous evaluations these stronger assumptions have proved reasonable: COMPOSER's assumptions produced sample complexities two to three order of magnitudes less than the weak assumptions adopted by Greiner and Jurisica, *without* compromising the statistical error [Gratch92]. However these promising results have the drawback that they are based on artificial domains and problem distributions. The ultimate usefulness of our approach depends on its behavior on real-world domains, which this paper addresses.

3 THE DEEP SPACE NETWORK

We applied the COMPOSER approach to improving the performance of a scheduler operating in the domain of spacecraft communication scheduling. The resulting system is called DSN-COMPOSER. The scheduling problem is a complicated real world task that has proved challenging to state-of-the-art scheduling techniques. The problem is to allocate communication requests between earth-orbiting satellites and the three 26-meter antennas at Goldstone, Canberra, and Madrid. These antennas make up part of the Deep Space Network (DSN) that is responsible for communication with earth-orbiting and inter-planetary spacecraft. Each satellite has a set of constraints, called project requirements, that define its communication needs. For example, the Nimbus-7 satellite must have at least four 15-minute communication slots per day, and these slots cannot be greater than five hours apart. Two factors complicate the problem. First, antennas are a limited resource – two satellites cannot communicate with the same antenna at the same time. Second, satellites can only communicate with certain antennas at certain times, depending on their orbits.

Scheduling is done on a weekly basis. A weekly scheduling problem is defined by three elements: (1) the set of satellites to be scheduled, (2) the constraints associated with each satellite, and (3) a set of time periods specifying all temporal intervals when a satellite can legally communicate with an antenna. Two time periods conflict if they use the same antenna and overlap in temporal extent. A valid schedule specifies a non-conflicting subset of all possible time periods where each project's requirements are satisfied.

3.1 THE SCHEDULER SYSTEM

LR-26 is a heuristic approach to the scheduling problem [Bell92] developed at the Jet Propulsion Laboratory. It provides a good platform for learning as it can be modified easily to incorporate alternative heuristic strategies. Furthermore, it uses an expert crafted control strategy that provide a challenging base-line to judge learned knowledge. There is also significant motivation to improve the effectiveness of the default control strategy. If LR-26 is chosen to replace the human scheduler, it will serve as one module in a larger interactive system. This system must make many repeated calls to the scheduler to compare variants of each weekly schedule. For this reason, LR-26 must provided solutions in a timely fashion.

Scheduling is formulated as a 0-1 integer programming problem [Taha82]. This is a methodology for finding an assignment to integer variables that maximizes the value of an *objective function*, subject to a set of linear constraints. The objective function characterizes the "value" of the solution. Many constraint satisfaction problems (CSP) are easily cast as integer programming problems [Mackworth92]. In the DSN domain, time periods are treated as 0-1 integer variables (0 if the time period is excluded from the schedule or 1 if it is included), the objective is to maximize the number of time periods in the schedule subject to the project requirements and temporal conflict constraints which are expressed as sets of linear inequalities. Integer programming is NP-hard, and the size of our scheduling problems makes the conventional approach impractical: a typical problem has approximately 650 variables and 1300 constraints. LR-26 embodies a heuristic approach called *lagrangian relaxation* [Fisher81]. Lagrangian relaxation requires identifying a set of constraints that, if removed, make the problem computationally easy. These constraints are "relaxed," meaning they no longer act as constraints but instead modify the objective function. A relaxed objective function is automatically generated such that satisfying relaxed constraints increases the value of the relaxed solution. The relaxed problem is by definition easy to solve and often finding the highest value relaxed solution solves the original problem. Furthermore, each relaxed constraint has a weight associated with it when it is added to the objective function. By systematically adjusting these weights and re-solving the relaxed problem, a solution to the unrelaxed problem is often efficiently discovered. Even if the unrelaxed problem cannot be solved in this manner, this weight adjustment cycle can move the scheduler closer to a solution, allowing the unrelaxed solution to be discovered with less search. LR-26 relaxes inter-antenna constraints. This representation facilitates an efficient implicit representation of temporal conflict constraints, which make up more than half of all constraints in a typical problem.

LR-26 combines lagrangian relaxation with standard constraint satisfaction search techniques. The scheduler performs depth-first search through a space of partial schedules. A variable is assigned a value of *in* if the associated time period is included in the partial schedule, *out* if it is excluded from the partial schedule. The scheduler constructs a complete schedule by incrementally extending the partial schedule. First it attempts to completely extend the schedule using the lagrangian relaxation method. If the relaxed solution satisfies all constraints it is returned. Otherwise, a set of possible extensions to the partial schedule is created and these are recursive explored. Extensions are created by choosing an unsatisfied constraint, identifying a set of uncommitted variables in the constraint, and assigning possible values to these variables. The set of extensions are placed on a stack to implement the depth-first search. The search continues until a solution is uncovered or a time-bound is reached. Currently the scheduler implements a time-bound of five CPU minutes. Any problem not solved within this bound is deemed unsolvable.

LR-26 can be viewed as a recursive application of four control decisions: (1) decide on a lagrangian weight adjustment scheme, terminating if a viable solution is found; otherwise (2) choose an unsatisfied constraint, (3) determine a set of extensions to the partial schedule that satisfy the constraint, and (4) determine an order to explore these extensions. The scheduler embodies a heuristic method for each of these control operations. These operations provide natural points at which to insert alternative learned methods.

3.2 PROBLEM DISTRIBUTION

We constructed a distribution of scheduling problems using the requirements and time periods of satellites using the deep space network. Ideally, we would use the identical problem distribution faced by the human experts in this domain. Unfortunately, not all of this information is in electronic form and thus is difficult to present to the LR-26 scheduler. There does, however, exist a large electronic database of information for many of the projects in the deep space network. We used this database to construct a large body of scheduling problems that are representative of, if not identical to, the type of problems faced by the human schedulers. Problems are generated by randomly choosing combinations of projects from the available data. The requirements and time periods represent the requirements and time periods of actual projects. The primary difference between these and actual problems lies in the particular combinations of projects that appear in the schedule.

We performed some initial evaluations of the LR-26 scheduler on these generated problems. We observed that some problems could not be solved by the scheduler using any of several search control strategies even with large resource bounds. This is consistent with the observation that the scheduling problem is inherently NP-Hard – there will be some problems that cannot be efficiently solved, even with good heuristics. These problems tend to dilute any performance improvement that we might gain through learning. For our experiments we eliminated this complicating factor by constructing our problem distribution without these unsolved problems. These problems were identified by solving each randomly generated problem multiple times using about twenty different search strategies (the strategies were identified during our pilot investigations). If a problem could not be solved by any of the strategies within the time bound, it was not added to the experimental distribution. For comparative purposes we include a secondary set of experiments that incorporate these unsolved problems. For a complete description of how training examples are generated, see [Gratch93].

3.3 EXPECTED UTILITY

In the DSN application a chief concern is with the computa-