OPTIMIZATION IN REAL TIME

Lon-Chan Chu and Benjamin W. Wah
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Abstract

In this paper, we present a search algorithm called **real-time search** for solving combinatorial optimization problems under real-time constraints. The problems we study are NP-hard and have good heuristic algorithms for generating feasible solutions. The algorithm we develop aims at finding the best possible solution in a given deadline. Since this objective is generally not achievable without first solving the problem, we use an alternative heuristic objective that looks for the solution with the best ascertained *approximation degree*. Our algorithm schedules a sequence of guided depth-first searches, each searching for a more accurate solution (based on the approximation degree set), or solutions deeper in the search tree (based on the threshold set), or a combination of both. Five versions of the RTS algorithm for setting approximation degrees and/or thresholds are formulated, evaluated, and analyzed. We develop an exponential model for characterizing the relationship between the approximation degrees achieved and the time taken. Our results show that our RTS algorithm finds solutions better than a guided depth-first search, and solutions found are very close to the best possible solution that can be found in the time allowed. We evaluate our results using the symmetric traveling-salesman, the knapsack, and the production-planning problems.

1. INTRODUCTION

Many important scheduling and planning problems in operations research and artificial intelligence are NP-hard **combinatorial optimization problems** (COPs) [4]. A COP is characterized by an objective to be optimized and a set of constraints to be satisfied. In this paper we are interested in COPs whose feasible solutions can be obtained easily by heuristic algorithms.

This research was supported by National Science Foundation Grant MIP 88-10584 and by National Aeronautics and Space Administration Contracts NCC 2-481 and NAG 1-613.

Proceedings of 12th Real-Time Systems Symposium, IEEE, San Antonio, TX, December 1991.

The process to solve COPs is generally limited by resource constraints. The solution process must terminate and return a solution when the resources allocated are expended. In a search limited by time and space, we will need to compromise the quality of the solution and return a suboptimal solution when time and/or space is expended. Of course, we are interested in finding the best feasible solution within the time and space constraints given. Without loss of generality, we describe our results with respect to minimization problems, since maximization problems can be transformed into minimization problems by negating their objectives.

There are two general methods to prune nodes in a search tree. The first is to prune nodes by setting thresholds so that all nodes with lower bounds exceeding the threshold are pruned. IDA* [5] is a powerful and efficient example in this class. The second method is to prune nodes by approximation. An error bound is set before or during the search so that those nodes that deviate from the best feasible solution by more than the error bound prescribed are pruned. The error bound we use is a function of the upper and lower bounds and is described in Section 2. TCA* [8] and TCGD [9] are search algorithms based on the second method.

In this paper, we consider a novel framework that combines both pruning methods described above for a search that needs to complete within a time limit. This framework is called **real-time search** (**RTS**) and consists of a sequence of primitive searches with pruning in each by thresholding and/or approximation. In this paper, we choose a guided depth-first search (GDFS) [7] as our primitive search because it does not require an exponential amount of memory space as A* does. We use a GDFS instead of a pure depth-first search because better guidance in GDFS leads to less search time. Note that a GDFS is a special case of the RTS algorithm with one iteration.

We are interested in a search algorithm that finds the best feasible solution in a given deadline. This objective is not achievable since the scheduler does not have enough information before the search is completed. We use an

alternative heuristic objective that looks for the solution with the best ascertained *approximation degree*.

The use of heuristic objectives in real-time scheduling is common. For example, in scheduling several tasks under a given deadline, the solution with the best overall solution quality is sought [2]. In this example, a problem-independent measure of the solution quality and its value are essential because they are used in scheduling time. Heuristic measures are also used in adaptive real-time systems for scheduling time [1, 3].

The real-time search proposed in this paper is different from Korf's real-time heuristic searches [6]. In Korf's method, a time constraint is associated for the expansion of a node in the search tree. During the time allowed, the searcher can either look deeper into the search tree or evaluate alternative heuristics in hoping that a better heuristic estimate or lower bound can be found. Korf showed that with more time allowed for expanding each node, less number of iterations are required for the entire search process. In our case, we associate a time constraint with respect to the entire search process and not on the expansion of individual nodes. Of course, tradeoffs similar to those studied by Korf can be applied here.

The **exact approximation degree** (**EAD**) of a solution z is defined as

$$\varepsilon^{ead} \triangleq \frac{z - z^*}{z^*} = \frac{z}{z^*} - 1, \tag{1.1}$$

where z^* is the optimal solution. The EAD is a *true* indicator of the quality of a solution because z^* is a constant for a given problem, and ε^{ead} is linear with z.

EAD is impractical to use, as the optimal solution is unknown until the problem is solved. During the search, we can use instead ε^{aad}, or **asymptotic approximation degree** (**AAD**), that is a function of the solution z and the lower bound v of the optimal solution.

$$\varepsilon^{aad} \triangleq \frac{z - v}{v} = \frac{z}{v} - 1. \tag{1.2}$$

Note that $\varepsilon^{aad} \geq \varepsilon^{ead}$ because $v \leq z^*$. It is asymptotic because it approaches ε^{ead} as the search progresses, and is equal to ε^{ead} when the search terminates with the optimal solution. One problem with using ε^{aad} is that it might not be consistent with ε^{ead}. The inconsistency occurs when two nodes to be searched have lower bounds v_1 and v_2 and solution values z_1^* and z_2^* such that $v_1 < v_2$ and $z_1^* > z_2^*$. In this case, $e_1^{aad} > e_2^{aad}$ and $e_1^{ead} < e_2^{ead}$ for a given z at the time when a decision needs to be made between searching nodes 1 or 2. The advantage of using ε^{aad} is that it can be computed at run time, while ε^{ead} cannot be computed until the optimal solution is found. In this sense, ε^{aad} is used as a heuristic measure of ε^{ead}. In our experiments, we use the AAD to guide the selection of the schedule and use the EAD to evaluate its effectiveness.

In terms of the EAD and AAD, the best algorithm for solving COPs under a given deadline is OPTA* which ''knows'' the optimal solution before the search begins and behaves like an A* during the search. OPTA* is the best in terms of the EAD and has $\varepsilon^{ead} = 0$ throughout the search (Eq. (1.1)). OPTA* also has the best ε^{aad} because the minimum lower bound in A* is always no less than that of any other search algorithm at any particular time; consequently, the AAD of OPTA* is an absolute lower bound of those of other search algorithms. However, OPTA* is not practical and can only serve as a benchmark for comparison because it requires the knowledge of the optimal solution ahead of time.

We have studied previously TCA* [8] and TCGD [9]. TCA* finds a solution with an AAD very close to that obtained by OPTA* in the time allowed, with a minimal increase in cost (in terms of the cumulated space-time product or the maximum space required) as incurred by OPTA*. Its major drawback is that it uses an exponential amount of space. An alternative is TCGD [9], which schedules a sequence of GDFSs with pruning by approximation. TCGD is a special case of RTS that prunes by approximation alone. Note that a GDFS performs very well with respect to the EAD because it finds good solutions early in the search if the guidance function is accurate. However, it does not behave well with respect to the AAD because nodes with small lower bounds are close to the root, which are expanded very late in the search. As a result, the minimum lower bound v does not improve until a large number of nodes have been expanded. Hence, in a search using the AAD as a quality measure, it is inappropriate to use the GDFS as a search strategy. This is the reason why TCGD performs worse than TCA* in terms of the AAD measure.

In this paper, we develop a novel framework of RTS using thresholding and/or approximation for solving COPs under real-time constraints. For the COPs studied, we observe that the execution time increases exponentially as the AAD achieved decreases, though anomalies may happen. RTS schedules a sequence of GDFSs, each searching for a better solution (in terms of ε^{aad}) than the previous one. By selecting thresholds and/or approximation degrees properly, RTS can perform very well in terms of both AAD and EAD. Five versions of RTS are studied: two using approximation alone (RTS[ε]), two using thresholding alone (RTS[θ]), and one using both thresholding and approximation (RTS[ε,θ]).

We study the behavior of RTS with various heuristics for selecting thresholds and approximation degrees. We verify the performance by simulations using the symmetric traveling-salesman problem (TSP), the production

planning problem (PP), and the knapsack problem (KS). Their performance is compared against that of OPTA* and GDFS in terms of both AAD and EAD.

This paper is organized as follows. Section 2 describes the background for search by approximation and thresholding. Section 3 discusses the parametrization of the AAD achieved as a function of time and its empirical justification. Section 4 shows various heuristics for approximation and/or thresholding. Analysis of the performance bounds of heuristics is also presented. Section 5 describes the experimental results on the five versions of RTS. Concluding remarks are drawn in Section 6.

2. BACKGROUND

In this section, we describe related work on primitive search, approximation, and thresholding. Important symbols used in this paper and their meaning are summarized in Table 2.1.

2.1. Search Primitive

In this paper, we use a GDFS [7] as the primitive building block of our RTS algorithm.

A node i in the search tree is associated with an upper bound u_i and a lower bound v_i. Let x_i be the best solution in the subtree rooted at node i. Then, $v_i \le x_i \le u_i$. In a monotone admissible heuristic search [7], v_i satisfies the following properties: (a) $v_i \le x_i$; (b) $v_i = x_i$ if node i is a leaf solution, and (c) $v_i \le v_j$ if node i is an ancestor of node j. Let z be the best solution found so far during the course of the search. Any node i with $v_i \ge z$ is obviously inferior and can be pruned.

2.2. Approximation

Approximation is related to pruning. A larger approximation will generally cause more nodes to be pruned and the search completed in less time. Let z^* be the optimal solution of the COP considered. The following pruning rule (P-RULE I) prunes node i when its lower bound v_i deviates from the current best solution z by an approximation degree ε.

$$\textit{P-RULE I}: \quad v_i \ge \frac{z}{1+\varepsilon}. \tag{2.1}$$

The final best solution obtained by applying P-RULE I has $\varepsilon^{aad} = \varepsilon$.

Let GDFS(ε, θ, T) denote a GDFS run with approximation degree ε, threshold θ (which is described in the next section), and deadline T. The AAD at time $t \le T$ can be computed as

$$\varepsilon^{aad}(\varepsilon, \theta, t) \triangleq \frac{z(\varepsilon, \theta, t) - v(\varepsilon, \theta, t)}{v(\varepsilon, \theta, t)} = \frac{z(\varepsilon, \theta, t)}{v(\varepsilon, \theta, t)} - 1, \tag{2.2}$$

Table 2.1. Important symbols used and their meaning.

Symbol	Meaning
RTS(T)	RTS with deadline T
OPTA*(T)	OPTA* with deadline T
π	($\triangleq (\varepsilon, \theta, T)$) constraint tuple defining approximation degree, threshold, and time constraint
GDFS(ε,θ,T)	GDFS with scheduled approximation degree ε, threshold θ, and deadline T
$\varepsilon^{aad}(\varepsilon,\theta,t)$	AAD at time t in GDFS(ε,θ,T), when $t \le T$
$\varepsilon^{aad}(\theta)$	AAD achieved in GDFS(0,θ,∞)
ε^{ead}	exact approximation degree
ε^{aad}	asymptotic approximation degree
ε_k	approximation degree scheduled for the k-th GDFS in RTS
ε_k^{aad}	AAD achieved in the k-th GDFS in RTS
ε_0^{aad}	AAD at the root node of the search tree
$\varepsilon_{gdfs(\pi)}^{aad}$	AAD achieved by GDFS(π) in deadline T
$\varepsilon_{rts(T)}^{aad}$	AAD achieved by RTS in deadline T
$\varepsilon_{opta*(T)}^{aad}$	AAD achieved by OPTA*(T) in deadline T
t^*	time to complete an A* search without a deadline, approximation, or thresholding
t'_k	time scheduled for the k-th GDFS
$t(\varepsilon, \theta, T)$	time to complete GDFS(ε, θ, T)
$t(\varepsilon, \theta)$	($\triangleq t(\varepsilon, \theta, \infty)$) time to complete a GDFS without a deadline
$t(\varepsilon)$	($\triangleq t(\varepsilon, \infty, \infty)$) time to complete a GDFS without a deadline or thresholding
τ	$\triangleq t(0, \infty, \infty)$
$z(\varepsilon, \theta, t)$	best solution found so far at time t in GDFS(ε, θ, T), where $t \le T$
$v(\varepsilon, \theta, t)$	lower bound of the optimal solution at time t in GDFS(ε, θ, T), where $t \le T$
z_k	final best solution in the k-th GDFS
θ_k	threshold scheduled in the k-th GDFS
z^*	optimal solution
g	gradient factor
r	growth rate

where $z(\varepsilon, \theta, t)$ is the best solution found so far, and $v(\varepsilon, \theta, t)$ is the lower bound of the optimal solution at time t. Let $t(\varepsilon, \theta)$ be the time needed to complete GDFS(ε, θ, ∞). If $t(\varepsilon, \theta) > T$, then the search is terminated at T, and the AAD achieved is $\varepsilon^{aad}(\varepsilon, \theta, T)$.

2.3. Thresholding

Thresholding is also related to pruning. A **threshold** θ is a value such that any node i with lower bound $v_i \ge \theta$ will be pruned. A smaller threshold will generally

Table 3.1. Statistical analysis on the standard deviations (normalized with respect to ε_0^{aad}) between the actual profile and the regressed profile for 50 different instances for each of TSP, PP and KS problems.

Problem	$\mu(D_i)$	$\sigma(D_i)$	$H_0 : \Pr(D_i \le B) \ge p$
TSP	.056	.0013	accept H_0 for $B = .1$, $p = .95$
PP	.051	.0018	accept H_0 for $B = .1$, $p = .95$
KS	.12	.0002	accept H_0 for $B = .15$, $p = .95$

cause more nodes to be pruned and the search completed in less time; however, the solution obtained may be worse.

Consider the following pruning rule (P-RULE II):

$$\textit{P-RULE II}: \quad v_i \ge \theta. \tag{2.3}$$

The final best solution z obtained by applying P-RULE II has

$$\varepsilon^{aad}(\theta) = \begin{cases} \dfrac{z-\theta}{\theta} = \dfrac{z}{\theta} - 1 & \text{if } \theta \le z^*, \\ 0 & \text{if } \theta > z^*. \end{cases} \tag{2.4}$$

Note that $\varepsilon^{aad}(\theta) \ge \varepsilon^{ead}(\theta)$ when $\theta \le z^*$; and $\varepsilon^{aad}(\theta) = \varepsilon^{ead}(\theta) = 0$ when $\theta \ge z^*$.

3. EXPONENTIAL MODEL

It is empirically observed that *the execution time for solving a COP grows exponentially as the AAD decreases linearly*. An intuitive explanation is that the COPs we study are NP-hard and that the execution time grows exponentially with the solution quality. In this section, we propose a model for this behavior and verify it empirically.

A **profile** is a trace of resource usage (such as the execution time incurred) versus solution quality (such as the AAD achieved). Experimental results described in this paper are presented in logical time (in terms of the number of search nodes expanded) so they are not implementation and computer dependent and can be reproduced on different computers.

The profiles studied here can be classified into one of the two types: actual profiles and parametric profiles. An **actual profile** shows the relationship between the execution time expended and the AAD computed in terms of z^* (rather than z). It is unknown until the search has been carried out. In contrast, a **parametric profile** is an estimate of the actual profile and is obtained by regressing the partial actual profile. The aptness and usefulness of the parametric profiles is demonstrated later in this section.

Two general algebraic profiles have been studied [8]. We found that high-order regression may be intractable in prediction; hence, we restrict to using the simple parametric profile in this paper. Let

$$\varepsilon = \beta_0 + \beta_1 \log t, \tag{3.1}$$

with the following two boundary conditions,

$$t(0) = \tau \quad \text{and} \quad t(\varepsilon_0^{aad}) = 1, \tag{3.2}$$

where τ is the execution time for the GDFS to find the optimal solution, and ε_0^{aad} is the AAD of the root node in the search tree. Note that τ is a problem-dependent constant. Substituting the boundary conditions into Eq. (3.1), we have

$$\varepsilon(t) = \varepsilon_0^{aad} \left(1 - \log_\tau t\right) \tag{3.3}$$

Rewriting Eq. (3.3) in terms of ε, we have

$$t(\varepsilon) = \tau^{\,1 - \frac{\varepsilon}{\varepsilon_0^{aad}}}. \tag{3.4}$$

Table 3.1 shows the statistical analysis of the standard deviations (normalized with respect to ε_0^{aad}) between the actual profile and the regressed actual profile for 50 different instances of each of TSP, PP, and KS problems. D_i is the root-mean-square of the difference between the actual profile for the i-th problem and the regressed profile. The KS problem has large deviations because it can be solved in pseudo-polynomial time, and its behavior is more like polynomial than exponential [4].

It may be inappropriate for us to conclude from our limited experiments that the profiles of other COPs can be parametrized by the simple exponential function. In general, the distribution of lower bounds with respect to the number of subproblems in a finite search space is *bell-shaped*.[1] The exponential function used in modeling the execution time fits well when the optimal solution lies on the rising edge of the bell-shaped curve, which is the case for the TSP and PP problems. The exponential model fails when (a) the optimal solution is near the peak or on the falling edge of the bell-shaped profile, or (b) the profile function is growing more than exponentially. Both cases happen frequently in hard-constrained COPs (such as general integer programming problems), which are not addressed in this paper. For these problems, feasible solutions cannot be derived easily. Case (a) also happens when the lower-bound function is loose in predicting the optimal solution.

[1] The statement that the distribution of lower bounds with respect to the number of subproblems is bell-shaped is not true in a COP with an infinite search space, such as the 15 puzzle problem.

4. RTS HEURISTICS

RTS limits the time used in each GDFS by a combination of thresholding and approximation. It selects a sequence of constraint tuples π, one for each GDFS, in terms of the approximation degree ε to be used, the threshold θ, and the time limit t so that the final approximation degree achieved, $\varepsilon_{RTS(T)}$, is minimized.

Two decision strategies are studied: **static** and **predictive**. By static, we mean that the parameters used in the decision heuristic is not changed during the RTS process. By predictive, we mean that the parameters used in the decision heuristic is predicted based on the previous performance or profile.

Two decision heuristics are studied: **linear gradient** (LG) and **first-order regression** (FR). By linear gradient, we mean that the pruning element (approximation degree or threshold) is changed (decreased or increased) in a linear fashion. By first-order regression, we mean that the approximation degree, threshold, or upper bound is regressed by a first-order polynomial function.

The pruning elements in our study are the approximation degree and threshold. Either or both can be applied to an individual GDFS. For simplicity, we restrict to cases in which all GDFSs are using one or both.

4.1. Approximation Heuristics

Let RTS[ε] denote the RTS with approximation alone. Two versions are studied: linear gradient (RTS[ε][LG]) and first-order regression (RTS[ε][FR]).

4.1.1. Approximation by Linear Gradient

Linear-gradient approximation works by decreasing the approximation degree in a *linear* fashion in hoping that the time expended will increase in an *exponential* fashion according to the simple exponential model. The k-th GDFS is scheduled with an approximation degree ε_k

$$\varepsilon_k = \varepsilon_0^{aad} (1 - k\, g), \tag{4.1}$$

where g is a **gradient factor** that controls the linear rate at which the approximation degree decreases.

The analysis is based on the simple exponential model, assuming that thresholding is not applied concurrently. The following theorem shows that the time expended in GDFS(ε_k, ∞, ∞) is exponential with k.

Theorem 4.1. The execution time in solving GDFS(ε_k, ∞, ∞) is

$$t(\varepsilon_k) = \tau^{k\,g}. \tag{4.2}$$

Proof. The theorem is proved by applying Eq. (4.1) to Eq. (3.4). □

The following theorem shows that the number of completed GDFSs are bounded below and above by functions of deadline T and gradient factor g.

Theorem 4.2. Assume that the number of completed GDFSs is k, and that g is not equal to 0. k is bounded by

$$k_{\min} = \left\lceil \frac{1}{g} \log_\tau \left(T(\tau^g - 1) + 1 \right) - 2 \right\rceil < k \le \left\lceil \frac{1}{g} \log_\tau \left(T(\tau^g - 1) + 1 \right) - 1 \right\rceil = k_{max.} \tag{4.3}$$

Proof. Since the k-th GDFS is the last completed one, the $(k+1)$-th GDFS is terminated prematurely due to deadline T. As a result, deadline T must be bounded *below* by the sum of times expended (defined in Eq. (4.2)) in all completed GDFSs, and must be bounded *above* by the sum of times expended in all completed ones plus the time expended for completing the $(k+1)$-th GDFS, namely, $\sum_{i=0}^{k-1} \tau^{i\,g} \le T < \sum_{i=0}^{k} \tau^{i\,g}$. The theorem is proved by rearranging these two inequalities with respect to k. □

For $T \ge 1$, $k_{\max}$ monotonically increases as T increases. When T is 1, $k_{\max}$ is 0. As a result, $k_{\max}$ is non-negative. To assure that $k_{\min}$ is non-negative, we can assume that $T \ge 2$, and that g is in the range $(0, \log_\tau (T-1)]$. Note that the worst-case value of ε_k can be computed by substituting $k_{\min}$ into Eq. (4.1).

In terms of the EAD and AAD, the best algorithm for solving COPs under a given deadline is OPTA*, which knows the optimal solution before the search starts. The AAD at the root node of OPTA* is

$$\varepsilon_{0,opta*}^{aad} \triangleq \frac{z^* - v_0}{v_0} = \frac{z^*}{v_0} - 1. \tag{4.4}$$

The time for completing an OPTA* search is the time for completing an A* search, *i.e.*, t^*. Assuming that OPTA* satisfies the simple exponential model, the boundary conditions of the profile of OPTA* are

$$t(0) = t^* \quad \text{and} \quad t(\varepsilon_{0,opta*}^{aad}) = 1. \tag{4.5}$$

By applying the boundary conditions above in Eq. (3.1), we obtain the parametric profile of OPTA* as follows.

$$\varepsilon_{opta*(T)}^{aad}(t) = \varepsilon_{0,opta*}^{aad} \left(1 - \log_{t*} t \right). \tag{4.6}$$

Define the **approximation loss** L^{aad} as the difference between the AAD of the root node in a GDFS and that in OPTA*; that is,

$$L^{aad} = \varepsilon_0^{aad} - \varepsilon_{0,opta*}^{aad}. \tag{4.7}$$

It represents the gap between the approximation degree achieved when the search starts and what is possible.

Let $\varepsilon^{aad}_{rts[\varepsilon][lg](T)}$ and $\varepsilon^{aad}_{opta*(T)}$ be the AADs achieved by RTS[ε][LG] and OPTA* in a deadline T, respectively. Also let $\Delta\varepsilon^{aad}_{rts[\varepsilon][lg](T)}$ be the upper bound of the difference in AAD between $\varepsilon^{aad}_{rts[\varepsilon][lg](T)}$ and $\varepsilon^{aad}_{opta*(T)}$. The following theorem shows this upper bound.

Theorem 4.3. In terms of the worst-case value of ε_k, $\Delta\varepsilon^{aad}_{rts[\varepsilon][lg](T)}$ is a function of deadline T and gradient factor g.

$$\varepsilon^{aad}_{rts[\varepsilon][lg](T)} - \varepsilon^{aad}_{opta*(T)} \le \Delta\varepsilon^{aad}_{rts[\varepsilon][lg](T)} = L^{aad} + F(g, T) + H(T), \quad (4.8)$$

$$\text{where } F(g, T) = 2\,\varepsilon^{aad}_0\, g - \varepsilon^{aad}_0 \log_\tau \left(\tau^g - 1 + \frac{1}{T}\right), \quad (4.9)$$

$$H(T) = \varepsilon^{aad}_{0,opta*} \log_{t*} T - \varepsilon^{aad}_0 \log_\tau T. \quad (4.10)$$

Proof. The worst-case $\varepsilon^{aad}_{rts[\varepsilon][lg](T)}$ can be obtained by applying the worst-case value of ε_k (due to $k_{\min}$) to Eq. (4.1). $\varepsilon^{aad}_{opta*(T)}$ can be obtained by using Eq. (4.6). The theorem is proved by subtracting the latter from the former. □

Note that L^{aad}, $F(g, T)$, and $H(T)$ are positive. $\Delta\varepsilon^{aad}_{rts[\varepsilon][lg](T)}$ is roughly logarithmic with respect to T, since the effect of T in $F(g, T)$ is very small and is negligible. As a result, as T becomes larger, $\Delta\varepsilon^{aad}_{rts[\varepsilon][lg](T)}$ will become looser. Also, $\Delta\varepsilon^{aad}_{rts[\varepsilon][lg](T)}$ is roughly linear with respect to g when g is large, since $\lim_{g\to 1} F(g, T) = \varepsilon^{aad}_0\, g$; it is independent of g when g is small, since $\lim_{g\to 0} F(g, T) = \varepsilon^{aad}_0 \log_\tau T$. When g is in between 0 and 1, the bound is curvilinear with g; so that there exists a best choice of g such that the bound is minimal. The following theorem shows the best choice of g for minimizing $\Delta\varepsilon^{aad}_{rts[\varepsilon][lg](T)}$.

Theorem 4.4. In terms of minimizing $\Delta\varepsilon^{aad}_{rts[\varepsilon][lg](T)}$, the optimal choice of the gradient factor g^* is

$$g^* = \log_\tau \left[2\left(1 - \frac{1}{T}\right)\right] \approx \log_\tau 2. \quad (4.11)$$

Proof. The theorem is proved by taking the derivative of Eq. (4.8) with respect to g, equating it to zero, and solving for the root. □

It is interesting to note that the effect of T on g^* is small, and that g^* is independent of ε^{aad}_0. One problem with g^* is that τ is unknown. If certain property of the target problem is known, such as the order-of-magnitude of τ, then g^* can be estimated quite accurately. For instance, g^* is in the range of [0.03,0.075] for τ in $[10^4,10^{10}]$. The following theorem shows the upper bound of the best $\Delta\varepsilon^{aad}_{rts[\varepsilon][lg](T)}$. Note that the bound is the best bound (due to g^*) in the worst case (due to $k_{\min}$).

Theorem 4.5. The minimal $\Delta\varepsilon^{aad}_{rts[\varepsilon][lg](T)}$ is bounded above by

$$\Delta\varepsilon^{aad}_{rts[\varepsilon][lg](T)}(g^*) < L^{aad} + 2\,\varepsilon^{aad}_0 \log_\tau 2 + H(T), \quad (4.12)$$

Proof. By applying g^* to Eq. (4.8), it can be rewritten as

$$\begin{aligned}\Delta\varepsilon^{aad}_{rts[\varepsilon][lg]}(g^*) &= L^{aad} + 2\varepsilon^{aad}_0 \log_\tau \left[2\left(1-\frac{1}{T}\right)\right] - \varepsilon^{aad}_0 \log_\tau \left[\left(1-\frac{1}{T}\right)\right] + H(T) \\ &= L^{aad} + 2\,\varepsilon^{aad}_0 \log_\tau 2 + \varepsilon^{aad}_0 \log_\tau \left[\left(1-\frac{1}{T}\right)\right] + H(T) \\ &< L^{aad} + 2\,\varepsilon^{aad}_0 \log_\tau 2 + H(T).\end{aligned}$$

□

4.1.2. Approximation by First-Order Regression

RTS[ε][FR] works by regressing the relationship between AADs and times expended in GDFSs i, $i = 0, ..., k$, and using the regressed curve to predict the ε^{aad}_k to be used next. Specifically, to predict ε^{aad}_k, the relationship in Eq. (3.1) is regressed over all the pairs $(\varepsilon^{aad}_i, t(\varepsilon_i))$ for $i = 0, ..., k-1$.

A GDFS scheduled later should expend more time than previous ones. As a result, the last GDFS is expected to produce the best solution. However, if the last GDFS cannot produce a better solution, then the time scheduled for it is wasted. Therefore, the prediction of the last completed GDFS is essential. Further, there needs a way to determine the time allocated to the GDFSs before the last one.

The basic rationale is that the ratio of times expended by successive GDFSs should be a constant r (called the **growth rate**). As a result, the time expended in a GDFS grows exponentially with respect to the number of GDFSs evaluated, as in Eq. (4.2).

Consider the k-th GDFS to be scheduled. Assume it will expend t'_k time units. The remaining time after completing it must be greater than t'_k in order for another GDFS of higher complexity to be scheduled. Therefore, the k-th GDFS is evaluated if

$$\sum_{i=0}^{k-1} t(\varepsilon_i) + (r+1)\, t'_k \le T \quad (4.13)$$

If the inequality is false, then the k-th GDFS is not granted, and a GDFS with an AAD that uses up the remaining time is scheduled.

Theorem 4.6. Assume that the simple exponential model is satisfied. The best growth rate r^* in terms of minimizing $\Delta\varepsilon^{aad}_{rts[\varepsilon][lg]}$ is

$$r^* = \tau^{g^*} \simeq 2. \quad (4.14)$$

Proof. By applying Eq. (4.2) for $t(\varepsilon_k)$ and $t(\varepsilon_{k-1})$, r can be obtained as

$$r \triangleq \frac{t(\varepsilon_k)}{t(\varepsilon_{k-1})} = \frac{\tau^{kg}}{\tau^{(k-1)g}} = \tau^g \quad (4.15)$$

The theorem is proved by applying g^* in Eq. (4.11). □

Note that this is the optimal r for an upper bound $\Delta\varepsilon^{aad}_{rts[\varepsilon][lg]}$ and not for the exact difference $\varepsilon^{aad}_{rts[\varepsilon][lg](T)} - \varepsilon^{aad}_{opta^*(T)}$.

4.2. Thresholding

Let RTS[θ] denote the RTS with thresholding alone. Two versions are studied: linear gradient (RTS[θ][LG]) and first-order regression (RTS[θ][FR]). These are discussed in the following sections.

4.2.1. Thresholding by a Linear Gradient

RTS[θ][LG] is similar to RTS[ε][LG]. Thresholds are selected in a linear fashion in hoping that the time expended in successive GDFSs increases in an exponential fashion [10].

$$\theta_k = v_0 + k\,g\,(z_0 - v_0), \quad (4.16)$$

where g is the **gradient factor** that controls the linear step of thresholding, v_0 is the lower bound of the root node, and z_0 is the heuristic (or upper-bound) solution at the root node. When the k-th GDFS with $\varepsilon = 0$, $\theta = \theta_k$, and $t_k = T - \sum_{i=0}^{k-1} t(0, \theta_i)$ is completed, the AAD achieved is

$$\varepsilon^{aad}_k(\theta_k) = \frac{z_k - \theta_k}{\theta_k} = \frac{z_k}{k\,g\,(z_0 - v_0) + v_0} - 1, \quad (4.17)$$

where z_k is the final best solution in the k-th GDFS.

The analytic bound for the AAD achieved by RTS[θ][LG] is difficult to derive because the relationship between thresholds and AADs is generally unknown.

4.2.2. Thresholding by First-Order Regression

RTS[θ][FR] is similar to RTS[ε][FR]. Thresholds are selected based on the relationship between thresholds and times expended in the past GDFSs. To predict θ_k, a first-order regression is done over all pairs of $(\theta_i, t(0, \theta_i))$ for $i = 1, \ldots, k-1$.

Note that the simple exponential model described in this paper is on AADs achieved versus times, rather than on thresholds versus times. However, we believe that the simple exponential model also holds for thresholds versus times [10]. In this paper, no empirical justification is attempted for thresholds versus times, because our goal is to obtain the best solution within a deadline for the heuristic objective AAD, and not on achieving the best threshold within the deadline.

As in RTS[θ][LG], the analytic bound for the AAD achieved by RTS[θ][FR] is difficult to derive.

4.3. Approximation and Thresholding by Linear Gradient

Improper settings of thresholds may cause serious waste of resources. To avoid this problem, a threshold must be greater than that in the previous GDFS, namely, $\theta_i > \theta_{i-1}$ for all i. Thresholding and approximation interact with each other, and one may dominate the other.

Consider GDFS(ε,θ,∞). Define the **optimality estimate** $\Omega(\varepsilon, \theta, t)$ as an estimate of the optimal solution at time t, namely,

$$\Omega(\varepsilon, \theta, t) \triangleq \frac{z(\varepsilon, \theta, t)}{1 + \varepsilon}, \quad (4.18)$$

where $z(\varepsilon, \theta, t)$ is the best solution found so far at time t. Note that $\Omega(\varepsilon, \theta, t(\varepsilon, \theta)) \le z^*$. Also note that the optimality estimate during the course of the GDFS is non-increasing because the best solution found is non-increasing and ε is fixed. In particular, $\Omega(\varepsilon, \theta, t(\varepsilon, \theta)) \le \Omega(\varepsilon, \theta, 0)$.

In terms of the profile of optimality estimates in the course of the GDFS, the interaction between approximation and thresholding can be classified into one of the three cases: (a) $\theta < \Omega(\varepsilon, \theta, t(\varepsilon, \theta))$, (b) $\theta > \Omega(\varepsilon, \theta, 0)$, and (c) $\Omega(\varepsilon, \theta, 0) \ge \theta \ge \Omega(\varepsilon, \theta, t(\varepsilon, \theta))$. In Case (a), thresholding dominates approximation because all search nodes pruned by approximation are also pruned by thresholding. In Case (b), approximation dominates thresholding because all search nodes pruned by thresholding are also pruned by approximation. In Case (c), neither dominates the other.

In our experience with search problems, it is easier to set tight thresholds in the beginning of a search so that thresholding dominates approximation. Later in the search, better approximation degrees can be set with the profile information obtained, and approximation dominates thresholding. This combination performs better than approximation or thresholding alone because (i) it can achieve the same AAD as a search using approximation alone but generally expends less time than the latter, and (ii) it expends less time than the one using thresholding alone.

Let RTS[ε,θ][LG] denote the RTS with linear-gradient approximation and thresholding. The basic idea is to select a threshold so that Case (c) above is satisfied.

The optimality estimate lies within the following range.

Table 5.1. Summary of the five versions of the RTS algorithm (NA denotes ''not applicable.'')

Notation of Version of RTS	Pruning Element(s) Used	Linear Gradient Heuristic	Parameter(s) for Linear Gradient	First-order Regression Heuristic	Parameter for First-order Regression
RTS[ε][LG]	ε	Yes	ε	No	NA
RTS[ε][FR]	ε	No	NA	Yes	ε
RTS[θ][LG]	θ	Yes	θ	No	NA
RTS[θ][FR]	θ	No	NA	Yes	θ
RTS[ε, θ][LG]	ε, θ	Yes	ε, θ	No	NA

$$\Omega(\varepsilon, \theta, t) \in \left[\frac{z_{k-1}}{1+\varepsilon_{k-1}^{aad}}, \frac{z_{k-1}}{1+\varepsilon_k} \right]. \quad (4.19)$$

Note that this range is defined by the final optimality estimate in the previous search and the initial optimality estimate in the current search. Let δ_k as the width of the bounds defined in Eq. (4.19); that is,

$$\delta_k \triangleq \frac{z_{k-1}}{1+\varepsilon_k} - \frac{z_{k-1}}{1+\varepsilon_{k-1}^{aad}}. \quad (4.20)$$

If the $(k-1)$-th GDFS is in Case (a), then θ_{k-1} is too small, and θ_k should be selected so that it is larger than the final optimality estimate in the previous search.

$$\theta_k = \frac{z_{k-1}}{1+\varepsilon_{k-1}^{aad}} + k\, g\, \delta_k, \quad (4.21)$$

where g is the gradient factor. The first term on the right-hand side of Eq. (4.21) is the final optimality estimate in the $(k-1)$-th GDFS, whereas the second term is the increment. The underlying idea is to set the increment closer to the width of optimality estimated defined in Eq. (4.19) as the number of completed GDFSs increases.

If the $(k-1)$-th GDFS is in Case (c), then θ_{k-1} was well selected and θ_k can be set by adding an increment that depends on the width defined in Eq. (4.19).

$$\theta_k = \theta_{k-1} + g\, \delta_k. \quad (4.22)$$

It is interesting to note that θ_k computed in Eq. (4.21) and that in Eq. (4.22) are ''equivalent'' in the following sense. Consider the range defined in Eq. (4.19) divided into $\lceil 1/g \rceil$ grids. Then θ_k defined in Eq. (4.21) is simply the k-th grid, whereas θ_k in Eq. (4.22) is to move from the $(k-1)$-th grid to the k-th.

The performance can be analyzed by by introducing a random variable that reflects the perturbances due to thresholding. This is not shown due to space limitation.

5. EXPERIMENTAL RESULTS

The performance of RTS with various heuristics is studied by simulations of the symmetric traveling-salesman problem (TSP), the production planning problem (PP), and the knapsack problem (KS) [4]. For clarity, the notations of the five versions of the RTS algorithm and their meaning are summarized in Table 5.1.

The simulation results of the five versions of the RTS algorithm, GDFS, and OPTA* on TSP, PP, and KS problems are shown in Figure 5.1. Other experimental results exhibit similar behavior, but are not presented here due to space limitation. Curves on the left-hand side of each figure are normalized with respect to ε_0^{aad}. This normalization is important for eliminating dependence on problem instances. Curves on the right-hand side of each figure are *not* normalized, because the EAD is a true indicator of the quality of a solution. It is important that *the time axis be logarithmically scaled* so that the major portion of time is on the right-hand side of each graph.

It is interesting to note that the AAD curves of OPTA* are always lower than those of other search algorithms. This is true because the AADs achieved by OPTA* are the absolute lower bounds. Also note that the EAD curves of OPTA* are horizontal lines at $\varepsilon^{aad} = 0$. This happens because the EAD measures deviation from the optimal solution, which is known to OPTA* before it starts.

As seen in Figure 5.1, GDFS(0,∞,T) generally performs well in the beginning in terms of the EAD but performs poorly in terms of the AAD. This happens because the GDFS can go deep in the search tree and finds a good solution very quickly, but the minimum lower bound is determined by the nodes close to the root that are evaluated very late; as a result, the AAD achieved is poor. Further, note that in the TSP and PP problems, GDFS(0,∞,T) outperforms others in terms of the EAD in the beginning only for a *minor* portion of time. (Note that the time axis is logarithmically scaled.) On the other hand, the RTS algorithm outperforms GDFS(0,∞,T) later on in terms of both EAD and AAD for a *major* portion of time,

(a) Experimental results of various search algorithms on TSP(20,2).

(b) Experimental results of various search algorithms on PP(18,2).

(c) Experimental results of various search algorithms on KS(100,2).

Figure 5.1. Experiments results of various searches studied in this paper. The symbols used in the figure and their corresponding algorithms are defined as follows.

‘‘a’’: RTS[ε][LG](*T*) ‘‘b’’: RTS[ε][FR](*T*) ‘‘c’’: RTS[θ][LG](*T*) ‘‘d’’:RTS[θ][FR](*T*)
‘‘e’’: RTS[ε,θ][LG](*T*) ‘‘*’’: OPTA*(*T*) ‘‘+’’: GDFS(0,∞,*T*) ‘‘B’’: Bound in Eq. (4.12)

especially in the TSP and PP problems. Also note that GDFS(0,∞,*T*) usually reaches a good solution very quickly, but little progress on the solution is made later on. This is true because GDFS(0,∞,*T*) generally goes deep in the search tree and may get trapped there. On the other hand, the RTS algorithm can explore different parts of the search tree; consequently, better solutions may be found, and both the AAD and EAD improve.

As seen in these figures, among the five versions of the RTS algorithm, RTS[θ][LG] (''c'') and RTS[θ][FR] (''d'') sometimes perform worse than other versions in terms of the EAD, as in the KS problems. This is true because thresholding primarily moves the lower bound to the optimal solution (as seen in Eq. (4.17)) rather than moving the best solution found to the optimal one. In terms of the AAD, RTS[ε][FR] (''b'') performs worse than other versions of the RTS algorithm. The other versions perform almost equally well for all the problems studied in terms of both EAD and AAD. The differences between $\varepsilon^{aad}_{rts[\varepsilon][lg]}$ and $\varepsilon^{aad}_{opta*}$ for most problems are generally within their analytic bounds defined in Eq. (4.12), which is only approximate as it is the best bound in the worst case.

As discussed before, GDFS(0,∞,*T*) outperforms the others in the beginning in terms of EAD for a minor portion of time, while RTS outperforms GDFS(0,∞,*T*) later on in terms of both EAD and AAD for a major portion of time. One way to combine the merits of both algorithms is to allocate a small portion of the time to GDFS(0,∞,*T*) in order to reach a good solution quickly, and then switch to the RTS algorithm in the remaining time. This method is ad hoc and the best division of time is generally difficult to choose. Experiments on this ad-hoc method are not carried out because we do not have a good algorithm for dividing the time between the two searches.

6. CONCLUDING REMARKS

We have the following observations on the different versions of the RTS algorithm we study in this paper.

(a) The worst performance with respect to the heuristic objective is obtained for a sequence of approximate guided depth-first searches with approximation degrees predicted by first-order regression.

(b) The best performance with respect to the original objective is obtained for a sequence of guided depth-first searches with approximation degrees predicted by a linear-gradient method, or a sequence of guided depth-first searches combining approximation and thresholding and predicted by a linear-gradient method.

(c) It is difficult to set thresholds and approximation degrees when they are used together.

(d) All versions of the RTS algorithm always perform better than a pure guided depth-first search with respect to the heuristic objective.

(e) The RTS algorithm performs better than a pure guided depth-first search most of the time with respect to the original objective.

REFERENCES

[1] S. R. Biyabani, J. A. Stankovic, and K. Ramamritham, ''The Integration of Deadline and Criticalness in Hard Real-Time Scheduling,'' *Proc. of the Ninth Real-Time System Symposium*, pp. 152-160, IEEE, Dec. 1988.

[2] S.-C. Cheng, J. A. Stankovic, and K. Ramamritham, ''Scheduling Algorithms for Hard Real-Time Systems: A Brief Survey,'' in *Tutorial in Hard Real-Time Systems*, ed. K. Ramamritham, pp. 150-173, IEEE, 1988.

[3] J.-Y. Chung and J. W. S. Liu, ''Algorithms for Scheduling Periodic Jobs to Minimize Average Error,'' *Proc. of the Ninth Real-Time System Symposium*, pp. 142-151, IEEE, Dec. 1988.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman, San Francisco, CA, 1979.

[5] R. E. Korf, ''Depth-First Iterative Deepening: An Optimal Admissible Tree Search,'' *Artificial Intelligence*, vol. 27, pp. 97-109, North-Holland, 1985.

[6] R. E. Korf, ''Real-Time Heuristic Search: First Results,'' *Proc. National Conf. on Artificial Intelligence*, pp. 133-8, AAAI, Inc., Seattle, Washington, June 1987.

[7] N. J. Nilsson, *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, NY, 1971.

[8] B. W. Wah and L.-C. Chu, ''TCA*--A Time-Constrained Approximate A* Search Algorithm,'' *Proc. Int'l Workshop on Tools for Artificial Intelligence*, pp. 314-320, IEEE, Nov. 1990.

[9] B. W. Wah and L.-C. Chu, ''TCGD: A Time-Constrained Approximate Guided Depth-First Search Algorithm,'' *Proc. Int'l Computer Symp.*, pp. 507-516, Tsing Hua Univ., Hsinchu, Taiwan, Dec. 1990.

[10] B. W. Wah, *MIDA*: An IDA* Search with Dynamic Control*, Research Report CRHC-91-09, Center for Reliable and High Performance Computing, Coordinated Science Laboratory, Univ. of Illinois, Urbana, IL, April 1991.