TCA* -- A TIME-CONSTRAINED APPROXIMATE A* SEARCH ALGORITHM

Benjamin W. Wah and Lon-Chan Chu

Center for Reliable and High Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1101 West Springfield Avenue
Urbana, IL 61801
wah%aquinas@uxc.cso.uiuc.edu

ABSTRACT

In this paper, we develop TCA*, a family of problem-independent, time-constrained, approximate A* search algorithms. The algorithms are designed to achieve the best ascertained degree of approximation with the minimum cost under a fixed time constraint, where cost is measured by either the cumulative space-time (CST) product or the maximum space encountered during the search. We consider only A* searches with admissible heuristic functions; a branch-and-bound algorithm with a best-first strategy is an example of such a search. We study NP-hard combinatorial optimization problems whose feasible solutions can be found easily in polynomial time. For the problems studied, we observe that the execution time, the CST product, and the maximum space required all increase exponentially as the degree of approximation decreases. The algorithms developed are evaluated by simulations using the traveling-salesman problem.

KEYWORDS AND PHRASES: A*, approximate branch-and-bound algorithm, best-first search, iterative deepening, space-time product, time constraint, traveling-salesman problem.

1. INTRODUCTION

The search for optimal solutions in a combinatorially large feasible space is important in artificial intelligence and operations research. Most of these problems are NP-hard and require an exponential amount of time to find the optimal solutions. The search is generally limited by resource constraints, such as the maximum time allowed; hence, approximations must be applied to terminate the search when resources are expended.

Approximation algorithms generally prune search nodes in the search tree in order to reduce the time and space required. We are interested in ones that give an ascertained (and perhaps minimum) degree of deviation from the optimal solution when the search terminates, assuming that admissible heuristic functions are used. The best of such approximation algorithms is $\mathbf{OPT_{A^*}}$, which "knows" the degree of approximation that can be obtained in the time allowed before the search begins. By ignoring the effects of anomalies due to approximations [4], OPT_{A^*} is the optimal approximation algorithm in the sense that it minimizes both the cumulative space-time (CST) product when the search terminates and the maximum space encountered during the search. Of course, OPT_{A^*} is not a practical algorithm and can only serve as a benchmark for comparison.

In this paper, we develop a family of problem-independent cost-effective approximate search algorithms, called TCA*, which can obtain approximate solutions very close to that obtained by OPT_{A^*} under the given time constraint, with a minor increase in cost as incurred by OPT_{A^*}. *The algorithm is based on the empirically observed fact that the time spent in a search, the CST cost when the search terminates, and the maximum space incurred during the search all grow exponentially when the degree of approximation decreases.*

A* in the form of a best-first branch-and-bound (B&B) search is used in this paper. The target search problems studied are combinatorial optimization problems which have many feasible solutions, such that the upper bounds of search nodes in the B&B tree can be computed in polynomial time. This class of problems covers a wide spectrum of important problems in artificial intelligence and combinatorial optimization. The results we develop are demonstrated by the traveling-salesman (TS) problem; results obtained for the knapsack (KS) and production planning (PP) problems are not shown due to space limitation.

The TS problems studied in this paper are generated by calling the random number generator *rand()*, which randomly determines the location of cities, one at a time, on a 100-by-100 Cartesian plane. Distances between cities computed this way satisfy the triangular inequality. All cities are assumed fully connected, and distances between them are symmetric. The lower-bound value in the B&B search is evaluated by finding the cost of the spanning tree that covers the cities not visited; the upper-bound value is computed by a hill-climbing heuristic. For simplicity, dominance due to symmetry is not tested in our implementation.

This paper is organized as follows. Section 2 defines terminologies in approximate branch-and-bound algorithms and TCA*. Section 3 describes the parametrization of the profiles and demonstrates empirically their aptness and usefulness. Three versions of the TCA* algorithm are described in Sections 4, 5, and 6 respectively. Conclusions are drawn in Section 7.

Research was supported partially by National Aeronautics and Space Administration Contract NCC 2-481 and by National Science Foundation Grant MIP 88-10584.

Proc. IEEE International Workshop on Tools for Artificial Intelligence, 1990.

2. APPROXIMATE BRANCH-AND-BOUND ALGORITHMS

A branch-and-bound (B&B) search [3] is a general formulation of a wide range of heuristic searches [2]. In particular, A* [5] can be considered as a B&B search with a best-first strategy. It decomposes a problem into smaller subproblems and repeatedly decomposes them until either a solution is found or infeasibility is proved. Each subproblem is represented by a node in the search tree. The method is characterized by four components: branching rule, selection rule, pruning rule, and termination criterion. Without loss of generality, only minimization problems are considered in this paper. Maximization problems are duals of minimization problems and can be solved by minimizing a negated objective function.

Let α, f_i, and z denote the degree of approximation, the lower-bound value of node i, and the incumbent value, respectively. The following pruning rule can be applied.

$$f_i \geq \frac{z}{\alpha+1}. \tag{2.1}$$

The final feasible solution obtained by applying the above pruning rule is called an **α-approximate semi-optimal solution**. The minimum possible degree of approximation, α_{min}, when the search terminates with incumbent value z is defined as follows.

$$\alpha_{min} = \frac{z-z^*}{z^*}. \tag{2.2}$$

Note that α_{min} is not known unless z^* is known.

When an A* algorithm is run with an approximation degree α and a time constraint T, we denote it as A*(α, T). Let t(α) be the time needed to complete the search with approximation degree α. For a search A*(α, T), if t(α) > T, then the search is terminated at T, and the following **run-time approximation degree** is computed.

$$\alpha_{run\text{-}time}(\alpha, T) = \frac{z(\alpha, T) - f_{global}(\alpha, T)}{f_{global}(\alpha, T)}, \tag{2.3}$$

where $z(\alpha, T)$ is the incumbent value when the search is stopped, and $f_{global}(\alpha, T)$ is the global lower-bound value obtained at time T.

Lawler and Wood first proposed an approximate search algorithm in their seminal paper [3]. Their algorithm works as follows. To solve a problem P under time constraint T, they start the search without any approximation for $T/2$ units of time. If the optimal solution is not found within $T/2$ units of time, they relax the degree of approximation to 5% and try to solve the problem for another $T/4$ units of time. If the 5%-approximate semi-optimal solution is not found within $T/4$ units of time, then they relax the degree of approximation by another 5% (*i.e.*, a total of 10%) and try to solve the problem for another $T/8$ units of time. This process is repeated until either the time constraint is exceeded or an approximate solution is found. Their algorithm for reducing the degree of approximation with respect to the search time is suboptimal in terms of the CST cost or the maximum space incurred, since it spends a major portion of its time resource in solving harder problems first. Further, the step size they have used (namely, 5%) should be tuned at run time based on the dynamic information collected.

The TCA* algorithm studied in this paper schedules a sequence of approximate A* searches. For a time constraint T, TCA* is defined formally as

$$\text{TCA}^* = \{ A^*(\alpha_0, t_0), ..., A^*(\alpha_k, t_k), A^*(\alpha_{k+1}, t_{k+1}) \}, \tag{2.7}$$

$$\text{and} \quad \sum_{j=0}^{k+1} t_j' = T, \tag{2.8}$$

where t_j is the time allowed in the j-th A* search, and t_j' is the actual time spent. Let $t(\alpha_j)$ be the time required to achieve approximation degree α_j, then

$$t_j' = \min\{ t_j, t(\alpha_j) \} \tag{2.9}$$

The degree of approximation achieved by TCA* is

$$\alpha_{\text{TCA}^*} = \min\{ \alpha_k, \alpha_{run\text{-}time}(\alpha_{k+1}, t_{k+1}') \} \tag{2.10}$$

The approach is to select a sequence of α_0, ..., α_{k+1}, and t_0, ..., t_{k+1} such that both α_{TCA^*} and cost are minimized.

Three versions of the TCA* algorithm are studied here: naive TCA* (*n*TCA*), static TCA* (*s*TCA*), and predictive TCA* (*p*TCA*).

The *n*TCA* algorithm simply solves the search problem without any approximation until either the time constraint is exceeded or an optimal solution is found. If an optimal solution cannot be found when the time constraint is exceeded, the feasible solution found and the final run-time approximation degree are reported. Formally,

$$n\text{TCA}^* = \{ A^*(0, T) \} \tag{2.11}$$

By applying a principle similar to Korf's iterative deepening [1], the *s*TCA* algorithm uses a *static* strategy to progressively reduce the degree of approximation in a linear fashion. Formally,

$$s\text{TCA}^* = \left\{ A^*(\alpha_0, T), ..., A^*\left[\alpha_{k+1}, T - \sum_{j=0}^{k} t(\alpha_j) \right] \right\} \tag{2.12}$$

Since the degree of approximation is a negative exponential of the cost spent, the cost of the final A* search overwhelms the costs of the earlier A* searches, and the solution obtained is only marginally better if the earlier searches were not performed.

The *s*TCA* algorithm can be improved by deriving at run time the profile function relating the degree of approximation and the cost. Instead of using a static strategy, the *p*TCA* algorithm applies *n*TCA* for a relatively small amount of time t_0, and based on a parametric relationship between the time spent and the degree of approximation achieved, estimates α_{pred}, the degree of approximation achievable in the remaining time, and continues the previous search with α_{pred} and a time constraint T. Formally,

$$p\text{TCA}^* = \{ A^*(0, t_0), A^*(\alpha_{pred}, T) \} \tag{2.13}$$

To illustrate the space-usage behavior of various heuristics, we simulate various instances of the symmetric TS problem. Figure 2.1 shows the space usage (in terms of the number of active nodes) versus time spent (in terms of the number of nodes expanded) for one instance of a 20-city TS problem. In general, the space used goes up in an exponential fashion and comes down abruptly after a good feasible solution is found. The top curve represents the behavior of *n*TCA*, *p*TCA*, and Lawler and Wood's method (LW). A time constraint of 16642 is put on the search, which is the time needed to find the optimal solution. As a result of this time constraint, approximations are scheduled to be applied

in LW at times 8321 (5%), 12482 (10%), 14562 (15%), etc. At time 8321, when 5% approximation is applied, all active nodes are pruned and the search terminates. This is indicated by the dotted line, which drops abruptly to zero at time 8321. nTCA* and pTCA* have the same space usage in this particular example, since the prediction in pTCA* overshoots, and the predicted degree of approximation used is 0. The behavior of sTCA* is indicated by a sawtooth curve, since a new and more accurate search is started whenever the previous approximate search is finished.

3. PARAMETRIC PROFILES

A **profile** is a trace of resource usage (*e.g.*, execution time, CST cost, and maximum space incurred) versus solution quality (*i.e.*, degree of approximation). In this paper, execution time is measured by the number of search nodes expanded, while memory space is measured by the number of active search nodes. Consequently, our results are implementation and computer independent.

The profiles studied in this paper are classified into four types: run-time profiles, actual profiles, performance profiles, and parametric profiles. A **run-time profile** is the profile collected by using a pure A* search without any approximation. An **actual profile** shows the experimentally obtained performance of OPT_{A^*}, which gives the best approximate solutions with the minimum costs under given time constraints. A **performance profile** is similar to an actual profile except that it is based on a given search algorithm, such as sTCA* or pTCA*. Finally, a **parametric profile** is an estimation of the actual profile and is obtained by regressing on the run-time profile.

One general parametric profile is proposed here.

$$\alpha = \sum_{k=0}^{n} \beta_k (\log t)^k, \tag{3.1}$$

where α is the degree of approximation, t is the time used, and β_k are constants. One difficulty with using Eq. (3.1) is that the regressed function may sometimes become a convex curve, which changes curvilinearly to a larger degree of approximation as time progresses. This is not acceptable, because a parametric profile must satisfy the condition that more accurate solutions are obtained as time increases. To avoid this problem, we restrict to using the simple exponential parametric profile in this paper.

$$\alpha = \beta_0 + \beta_1 \log t, \tag{3.2}$$

with the following two boundary conditions,

$$t = M \geq 1 \text{ for } \alpha = 0, \quad \text{and} \quad t = 1 \text{ for } \alpha = \alpha_0 > 0, \tag{3.3}$$

where M is the maximum time required to find the optimal solution, and α_0 is the run-time approximation degree obtained after the root has been expanded.

Substituting the boundary conditions into Eq. (3.2), we have

$$\alpha(t) = \alpha_0 - \frac{\alpha_0}{\log M} \log t = \alpha_0 \left[1 - \log_M t \right]. \tag{3.4}$$

Rewriting Eq. (3.4) in terms of α, we have

$$t(\alpha) = M^{1 - \frac{\alpha}{\alpha_0}}. \tag{3.5}$$

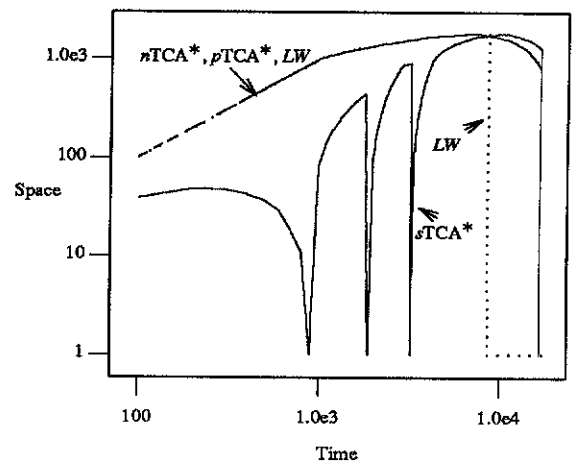


Figure 2.1. Comparison of space usage versus time for a TS problem instance of size 20 using nTCA*, sTCA*, pTCA*, and Lawler and Wood's (LW) method.

Table 3.1. Summary of standard deviations between the run-time profile and the regressed curve of 10 traveling-salesman problem instances of size ranging from 11 to 20. (n is the maximum degree of polynomials used in the regression.)

Standard Deviations	n=1	n=2	n=3	n=4	n=5
Minimum	0.034	0.025	0.021	0.021	0.021
Average	0.066	0.054	0.038	0.036	0.036
Maximum	0.114	0.096	0.086	0.071	0.071

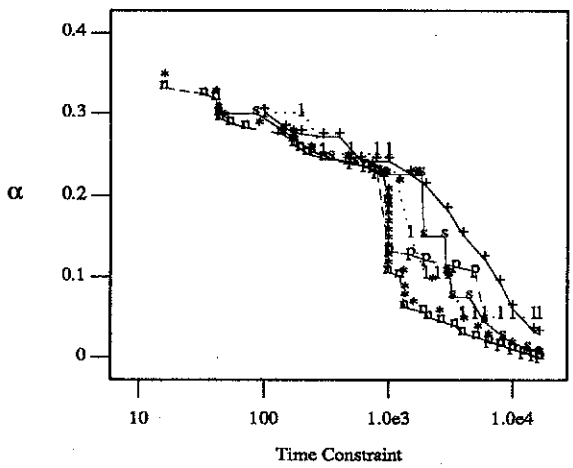
By observing the actual profiles of more than 30 problem instances of KS, TS, and PP problems, these profiles can be well fitted by regression. Table 3.1 shows the standard deviation (normalized with respect to α_0) between the run-time profile and the regressed curve for different values of n defined in Eq. (3.1) and for 10 instances of TS. It shows that there is a close fit between the predicted and the actual curves. Moreover, it shows that it is sufficient to use n=1 in the regression; that is, Eq. (3.2) is a sufficient approximation to the run-time profile.

Note that the parametric profile is estimated from the run-time profile. Figures 4.1a thru 4.1c show that there are minor differences between the run-time and actual profiles. Therefore, it is quite apt to predict the parametric profile from the run-time profile instead of the actual profile. The reason for using the run-time profile to do regression is that more samples can be collected as compared to using the actual profile for regression.

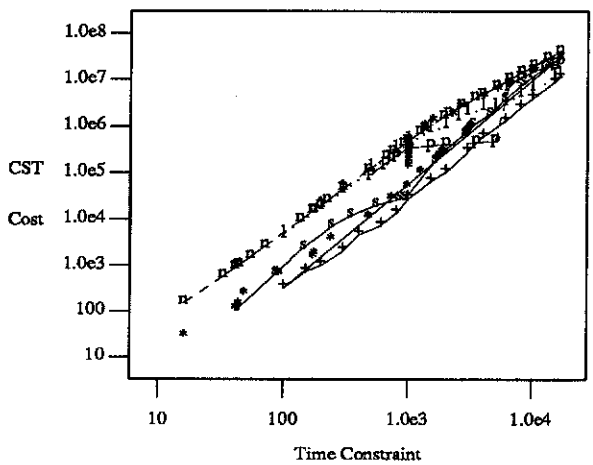
In general, the distribution of lower bounds with respect to the number of subproblems in a finite search space is bell-shaped.* The exponential function used in modeling the approximation behavior fits well when the optimal solution lies on the rising edge of the bell-shaped curve, which is

* This statement is not true in a search problem with an infinite search space, such as the 15-puzzle problem.

the case for the KS, TS, and PP problems. The exponential model fails when (a) the optimal solution is near the peak or on the falling edge of a bell-shaped profile, or (b) the profile function is growing more than exponentially. Situation (a) happens frequently in constrained optimization problems, such as integer programming problems. These problems are not addressed in this paper since their feasible solutions cannot be derived easily. Situation (a) can also happen when the lower-bound function is loose in predicting the optimal-solution value. Situation (b) happens in very hard search problems. In the last two cases, an algorithm which dynamically predicts the profile during the search is needed. This is applied in the *dynamic* TCA* algorithm, which is not be described in this paper due to space limitation.



(a) Approximation degree versus time constraint



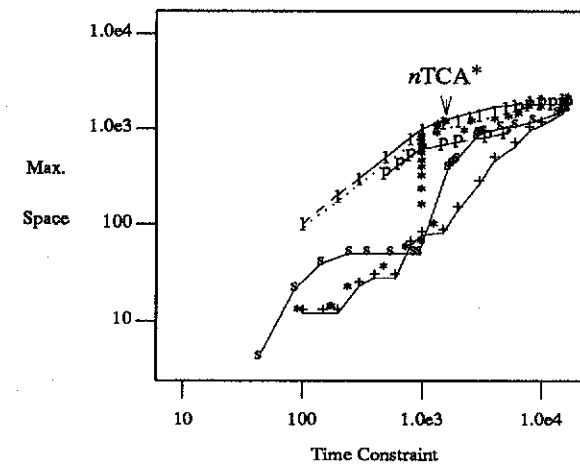
(b) CST cost versus time constraint.

4. NAIVE TCA*

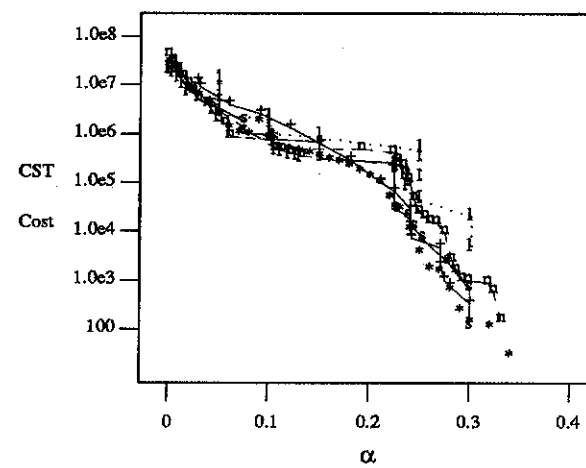
The naive TCA* (or *n*TCA*) algorithm solves the search problem without any approximation and reports the incumbent found when either the time resource is exceeded or the problem is solved. The run-time approximation is quite close to the best degree of approximation achievable under the time constraint, but the corresponding cost is high since the algorithm only tries to solve the *hardest* search problem, namely, the one without any approximation.

Applying Eq. (3.4), the run-time approximation degree achieved is

$$\alpha_{n\text{TCA}^*}(T) = \alpha_0(1 - \log_M T) \tag{4.1}$$



(c) Maximum space used versus time constraint.



(d) CST cost versus approximation degree.

Figure 4.1. Performance profiles for a TS problem instance of size 20 using OPT(*), Lawler and Wood's method (l), *n*TCA* (n), *s*TCA* (s, with g=0.25; and +, with g=0.071) and *p*TCA* (p, with s=0.15 and c=0.6).

The quality of a solution is determined by its degree of approximation achieved in the time constraint. A better algorithm should find more accurate semi-optimal solution with less cost in a shorter amount of time. An algorithm whose performance is very close to the actual profile is desirable.

The performance profiles of nTCA* are plotted in Figures 4.1a thru 4.1c in terms of approximation degree, CST cost, and maximum space used with respect to a given time constraint. As a comparison, Lawler and Wood's method achieves a worse approximation degree at a higher cost.

Figure 4.1a shows that it is possible for nTCA* to achieve a better approximation degree than OPT_A* for a given time constraint. This anomaly happens because nodes in the search tree may be searched in different orders in OPT_A* (searched with approximations) and in nTCA* (searched without approximations) [4], and the global upper bound found in nTCA* may sometimes be better than the feasible-solution value found in OPT_A*.

Figure 4.1d shows a tradeoff between the approximation degree achieved and the associated CST cost (under different specified time constraints). It shows that Lawler and Wood's methods and nTCA* have much higher costs for the approximation degrees achieved as compared to OPT_A*.

5. STATIC TCA*

The static TCA* (or sTCA*) algorithm works by progressively solving the problem using A* searches with smaller degrees of approximation. Referring to Eq. (2.12), the degree of approximation used in the j-th A* search, α_j, is defined as follows.

$$\alpha_j = (1 - g \times j)\alpha_0 \qquad 0 < g \le 1, \tag{5.1}$$

where g is a constant **stepping factor**.

For a combinatorial optimization problem instance P solved under time constraint T, sTCA* works as follows.

(1) In the zeroth search, evaluate the root node and compute its run-time approximation degree α_0. Set j to 0.

(2) If the problem is not solved and the time constraint is not violated, then compute α_{j+1} using Eq. (5.1). Execute $A^*\left(\alpha_{j+1}, T - \sum_{i=1}^{j} t(\alpha_i)\right)$.

(3) Repeat Step (2) until the time constraint T is violated. Compute α returned using Eq. (2.10).

To achieve the best degree of approximation and the least cost under a time constraint, g must be chosen properly such that α_{sTCA*} defined in Eq. (2.10) is minimized.

All the lemmas and theorems shown below assume that the profile equation defined in Eq. (3.5) is satisfied. Several important symbols used in the lemmas and theorems are listed in Table 5.1.

The following lemma shows the relationship among the time $t(\alpha_j)$ for solving the j-th A* search $A^*(\alpha_j, t(\alpha_j))$, the iteration index j, and the stepping factor g.

Lemma 5.1. The execution time $t(\alpha_j)$ for evaluating $A^*(\alpha_j, t(\alpha_j))$ is exponential with respect to j and g.

$$t(\alpha_j) = M^{g \times j}. \tag{5.2}$$

Proof. Proof is done by applying Eq. (5.1) to Eq. (3.5). □

The following lemma shows the bounds on the number of completed iterations in sTCA*. Note that the $(k+1)$-th search is terminated prematurely due to the time constraint.

Table 5.1. Summary of symbols used.

Symbol	Meaning
T	Time constraint allowed for the search.
M	Max. time required to find the optimal solution.
k	Number of completed iterations in sTCA*.
α_{OPT_A*}	Approx. degree of OPT_A* under time constr. T.
α_{sTCA*}	Approx. degree of sTCA* under time constr. T.
α_0	Run-time approx. degree of the root node.
α_j	j-th approx. degree in sTCA*.
t_j	Time allowed in the j-th iteration.
$t(\alpha_j)$	Time needed in achieving α_j.
g	Stepping factor used in sTCA*.

Lemma 5.2. The bounds on k, the number of A* searches in sTCA* completed under time constraint T, satisfies

$$\frac{\log_M \left[T(M^g - 1) + 1 \right] - 2}{g} < k \le \frac{\log_M \left[T(M^g - 1) + 1 \right] - 1}{g}, \tag{5.3}$$

if g is not equal to 0.

Proof. Assume that $A^*(\alpha_0, t(\alpha_0))$, ..., $A^*(\alpha_k, t(\alpha_k))$, are done to completion, and that $A^*\left(\alpha_{k+1}, T - \sum_{j=1}^{k} t(\alpha_j)\right)$ is terminated prematurely because time is expended (Eq. (2.12)). The following inequalities must hold.

$$\sum_{j=0}^{k} t(\alpha_j) \le T < \sum_{j=0}^{k+1} t(\alpha_j). \tag{5.4}$$

Applying Eq. (5.2) and simplifying the geometric series, we can rewrite Eq. (5.4) as follows.

$$\frac{M^{g(k+1)} - 1}{M^g - 1} \le T < \frac{M^{g(k+2)} - 1}{M^g - 1}. \tag{5.5}$$

The lemma is proved by rewriting Eq. (5.5) into an inequality with respect to k. □

For $T \ge 1$, the upper bound of k defined in Eq. (5.3) increases monotonically as T increases. When T is 1, this upper bound is zero. Hence, the upper bound defined in Eq. (5.3) is non-negative.

Assuming that $T \ge 1$, the lower bound of k defined in Eq. (5.3) may be negative if the stepping factor g is improperly large. However, if g is in the following range,

$$g \in (0, \log_M (T-1)], \tag{5.6}$$

then the lower bound defined in Eq. (5.3) is non-negative. Eq. (5.6) is derived by solving the inequality that the lower bound in Eq. (5.3) is greater than or equal to 0. Then, the lower bound on k, denoted by κ, is

$$\kappa = \frac{1}{g} \log_M \left[T(M^g - 1) + 1 \right] - 2 \ge 0. \tag{5.7}$$

The worst-case value of α_k based on κ is

$$\alpha_k < \alpha_0 (1 - g \times \kappa). \tag{5.8}$$

The following theorem shows that the difference between α_{sTCA^*} and $\alpha_{OPT_A^*}$ is bounded.

Theorem 5.1.

$$\alpha_{sTCA^*} - \alpha_{OPT_A^*} \le 2\,\alpha_0\, g - \alpha_0 \log_M \left[(M^g - 1) + \frac{1}{T} \right]. \quad (5.9)$$

Proof. Assuming that $A^*(\alpha_0, t(\alpha_0))$, ..., $A^*(\alpha_k, t(\alpha_k))$, are done to completion, and that $A^*\left[\alpha_{k+1}, T - \sum_{j=1}^{k} t(\alpha_j) \right]$ is terminated prematurely because time is expended (Eq. 2.12), we compute α_k as a *pessimistic* measure of α_{sTCA^*} (instead of using the exact formula given by Eq. (2.10)). The worst-case α_k is equal to α_κ, which can be rewritten by applying Eq. (5.7) to Eq. (5.8).

$$\alpha_\kappa = \alpha_0 \left[1 - \log_M \{ T(M^g - 1) + 1 \} + 2g \right]. \quad (5.10)$$

Applying Eq. (3.4), $\alpha_{OPT_A^*}$ is

$$\alpha_{OPT_A^*} = \alpha_0 \left[1 - \log_M T \right]. \quad (5.11)$$

The theorem is proved by finding the difference between Eq's (5.10) and (5.11). □

Theorem 5.1 shows that g is the dominant factor in the maximum difference between the degrees of approximation obtained by sTCA* and that by OPT_A^*, since $1/T$ is usually small. The bound on the normalized difference can be derived by dividing both sides of Eq. (5.9) by α_0.

sTCA* has the **warm start** problem at the beginning of an A* search because execution time (and CST cost) must be spent in order to achieve the degree of approximation in the last A* search. This problem is more serious when the stepping factor is too small. However, choosing large stepping factors may result in high cost in the last search in the sTCA*. A succinct choice of the stepping factor is, therefore, essential. The following lemma shows that the best stepping factor which minimizes the (normalized) difference of approximation degrees should be kept small.

Lemma 5.3. In terms of the (normalized) difference of approximation degrees, the best stepping factor g^* is

$$g^* = \log_M \left\{ 2\left(1 - \frac{1}{T}\right) \right\} \quad (5.12)$$

Proof. Let $D(g,T)$ and $\tilde{D}(g,T)$ be the difference defined in Eq. (5.9) and the normalized difference, respectively. Note that $D(g,T) = \alpha_0 \tilde{D}(g,T)$. Taking partial directives on $D(g,T)$ and $\tilde{D}(g,T)$ with respect to g, we have

$$\frac{\partial D}{\partial g} = \alpha_0 \frac{\partial \tilde{D}}{\partial g} = 2\alpha_0 \left[1 - \frac{\alpha_0 M^g}{M^g - 1 + \frac{1}{T}} \right] \quad (5.13)$$

The lemma is proved by setting $\partial D/\partial g = 0$ (or equivalently, $\partial \tilde{D}/\partial g = 0$) and solving for g^*. □

By using the best stepping factor, the following theorem shows that the difference in approximation degrees is very small.

Theorem 5.2.

$$\left[\alpha_{sTCA^*}(g^*) - \alpha_{OPT_A^*} \right] < 2\,\alpha_0 \log_M 2 \quad (5.14)$$

Proof. Substituting g^* defined in Eq. (5.12) into Eq. (5.9) and simplifying the combined equation, we have

$$\alpha_{sTCA^*}(g^*) - \alpha_{OPT_A^*} \le \alpha_0 \log_M \left[1 - \frac{1}{T} \right] + 2\,\alpha_0 \log_M 2 \quad (5.15)$$

The first term on the right-hand side is negative since $1 - 1/T$ is smaller than 1. Hence, this term can be neglected and the inequality still holds. The theorem is proved by simply ignoring the first term on the right-hand side. □

Theorem 5.2 shows that sTCA* is very powerful when the best stepping factor is used. It gives the best performance bound of sTCA*. However, M is unknown ahead of time, which implies that the best stepping factor cannot be found until we have solved the problem. The performance when a suboptimal stepping factor is used is worse than that given by Eq. (5.14).

The performance profiles of sTCA* for the TS problem with different stepping factors are not shown here due to space limitations. In general, larger stepping factors g result in better approximation degrees, but higher CST costs and higher maximum space usage. For example, for a 20-city TS problem instance, $t(0) = M = 16642$. According to Lemma 5.3, $g^* = 0.071$. Using Theorem 5.2, we compute the upper bound on the difference of approximation degrees to be 0.059. With $g = 0.071$, the performance of sTCA* is shown in Figures 4.1a thru 4.1d.

The performance of sTCA* with $g = 0.071$ is not as good as the performance with $g = 0.25$, as shown in Figures 4.1a thru 4.1d. The associated costs (in terms of CST or maximum space used) in sTCA* with $g = 0.25$ is consistently lower and approach that of OPT_A^*. One possible reason is that the analysis shown above is based on the assumption that the profile is exponential. The actual profile, as seen in Figure 4.1a, is not.

6. PREDICTIVE TCA*

The predictive TCA* (or pTCA*) algorithm works (a) by using nTCA* to collect a partial run-time profile, (b) by regressing it, (c) by predicting the best degree of approximation achievable under the time constraint by the parametric profile derived from the partial run-time profile, and (d) by solving the search problem with the predicted best degree of approximation.

Consider a combinatorial optimization problem instance P to be solved under time constraint T. pTCA* works as follows.

(1) Profiling phase. Execute $A^*(0, T_0)$ using a relatively small amount of time $T_0 = s \times T$, $0 < s \le 1$. Collect the partial run-time profile.
(2) Regression phase. Using Eq. (3.4), estimate the entire parametric profile by regressing from the partial run-time profile collected in the profiling phase.
(3) Prediction phase. Predict α_{pred}, the best degree of approximation achievable in the remaining time. If α_{pred} is negative, set α_{pred} to zero, which implies that the problem can be solved optimally without approximation. Compute $\alpha = c \times \alpha_{pred}$.
(4) Solution phase. Resume the search and switch to $A^*(\alpha, T)$ in the remaining time until either time has

expended or the semi-optimal solution with the predicted degree of approximation is found. If P is not completely solved when time is expended, compute the run-time approximation degree using Eq. (2.3) as the final degree of approximation achieved.

Two parameters in pTCA*, the stopping factor and the corrective factor, are needed to fine tune the performance profile and alleviate the effects of inaccurate profiling. The **stopping factor** s ($0 < s \leq 1$) defines how much time should be spent in the profiling phase. The **corrective factor** c adjusts the predicted best degree of approximation to avoid overshooting or undershooting.

The selection of the stopping factor will significantly affect the prediction. The larger stopping factor is, the more accurate will be the prediction. However, when more time in spent in profiling, more CST cost will be incurred. A trade-off on the value of s must be made.

The stopping factor can be set relatively or absolutely. A stopping factor (s_r) is relative if it defines a fraction of the total time T allowed. The stopping factor is absolute if it defines an absolute maximum amount of time that is applied in profiling ($s_a = T_s/T$, where T_s is the maximum allowable time for profiling). A hybrid between relative and absolute can be chosen as $s = \min\{ s_r, s_a \}$. For large T and large problems, choosing an absolute stopping factor is preferable because it avoids spending a significant amount of time and CST cost in the profiling phase. In contrast, for small T, choosing an amount of time relative to T for profiling is better.

pTCA* suffers from the **cold start** problem, which happens because the actual or run-time profiles cannot be linearly regressed well at the beginning of the solution process. This inaccurate prediction can cause either an overshoot or an undershoot when time is expended. An overshoot causes a high CST cost since more search nodes than necessary are generated. An undershoot causes the search problem to be solved with a looser degree of approximation, which results in a worse semi-optimal solution. The cold start problem can be alleviated by choosing the stopping factor large enough so that enough profiling is performed.

The performance profiles for various stopping factors and corrective factors are not shown here due to space limitations. In general, large corrective factors result in poor approximation with low CST cost and maximum-space usage. On the other hand, small corrective factors result in high costs but not significant improvement in approximation degrees. Our experiments indicate that a good compromise between cost and approximation degree is to have s=0.15 and c=0.6.

With the corrective and stopping factors selected as above, the performance of pTCA* is plotted in Figures 4.1a thru 4.1d. From the results depicted and others not shown, we observe that, for small time constraints, the time spent in profiling is small, and the prediction may be very poor. As a result, the approximation degrees obtained are low and the costs are high. In these cases, sTCA* is a better choice than pTCA*. For large time constraints, more time is spent in profiling and more accurate prediction can be made by pTCA*. In these cases, pTCA* is marginally better than sTCA*.

7. CONCLUDING REMARKS

In this paper, we develop TCA*, a family of problem-independent, time-constrained, approximate A* search methods. Three versions of TCA* are studied: naive TCA* (nTCA*), static TCA* (sTCA*), and predictive TCA* (pTCA*). These algorithms are designed to solve combinatorial optimization problems with low cost and a degree of accuracy almost equal to the maximally achievable accuracy under a given time constraint. They achieve this goal by progressively increasing the accuracy of solutions found and by using a relatively small amount of resources (in terms of cumulative space-time product or maximum-space usage).

Our methods are compared with Lawler and Wood time-constrained A* search, which achieves a degree of approximation at a cost about one order of magnitude worse than the best cost achieved in TCA*. The poor performance of Lawler and Wood's method is attributed to the large and inflexible step size used, and the search of more accurate solutions in the beginning of the process (which causes time and space used without finding a solution).

Instead of searching for more accurate solutions first, sTCA* and pTCA* search for less accurate solutions in the beginning (hence using resources slowly and gradually approaching the maximum degree of approximation that can be achieved in the time constraint). sTCA* and pTCA* both achieve degrees of approximation very close to the maximum achievable with moderate to low costs. sTCA* performs better when the time constraint is tight, while pTCA* works better otherwise. With a larger time constraint, better prediction can be made in the profiling phase of pTCA*, which contributes to better performance.

REFERENCES

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

[2] V. Kumar and L. N. Kanal, "A General Branch and Bound Formulation for Understanding and Synthesizing And/Or Tree Search Procedures," *Artificial Intelligence*, vol. 21, no. 1-2, pp. 179-198, North-Holland, 1983.

[3] E. L. Lawler and D. W. Wood, "Branch and Bound Methods: A Survey," *Operations Research*, vol. 14, pp. 699-719, ORSA, 1966.

[4] G. J. Li and B. W. Wah, "Computational Efficiency of Combinatorial OR-Tree Searches," *Trans. on Software Engineering*, vol. 16, no. 1, IEEE, Jan. 1990.

[5] N. J. Nilsson, *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, 1971.