International Journal on Artificial Intelligence Tools, Vol. 6, No. 2 (1997) 255–271

TCGD: A TIME-CONSTRAINED APPROXIMATE GUIDED DEPTH-FIRST SEARCH ALGORITHM *

BENJAMIN W. WAH
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801, U.S.A.
b-wah@uiuc.edu

LON-CHAN CHU
Microsoft Corporation
Redmond, WA 98052, U.S.A.
lonchanc@microsoft.com

Received 14 October 1995
Revised 20 January 1997

ABSTRACT

In this paper, we develop *TCGD*, a problem-independent, time-constrained, approximate guided depth-first search (GDFS) algorithm. The algorithm is designed to achieve the best ascertained approximation degree under a fixed time constraint. We consider only searches with finite search space and admissible heuristic functions. We study NP-hard combinatorial optimization problems with polynomial-time computable feasible solutions. For the problems studied, we observe that the execution time increases exponentially as approximation degree decreases, although anomalies may happen. The algorithms we study are evaluated by simulations using the symmetric traveling-salesperson problem.

Keywords: A^*, TCA^*, *TCGD*, approximate branch-and-bound algorithm, best-first search, guided depth-first search, iterative deepening, time constraint, symmetric traveling-salesman problem.

*Research was supported partially by National Science Foundation Grants MIP 88-10584, MIP 92-18715 and MIP 96-32316 and National Aeronautics and Space Administration Contracts NCC 2-481 and NAG 1-613.

The paper has been extended from the papers published in references [1, 2].

1 Introduction

The search for optimal solutions in a combinatorially large solution space is important in artificial intelligence and operations research. A combinatorial optimization problem is characterized by an objective to be optimized and a set of constraints to be satisfied. Most of these problems are NP-hard and require exponential time to find the optimal solutions.

General search strategies that solve these problems include best-first search (BFS), depth-first search (DFS), guided depth-first search ($GDFS$), breadth-first search, and iterative deepening A^* (IDA^*) [3]. In this paper, we focus only on guided depth-first searches. The difference between DFS and $GDFS$ is that in $GDFS$ all immediate descendents of a node selected for expansion are expanded first before proceeding to descendents of expanded nodes, whereas in DFS only one immediate descendent is selected for expansion before descendents of this expanded node are examined. In general, $GDFS$ and DFS require much less memory space than BFS, although they generally expand more nodes than BFS because they may need to explore deep in a search tree before finding a solution. $GDFS$ expands less nodes than DFS when the guidance function is accurate, since all immediate descendents can be examined simultaneously before proceeding deeper in the tree.

The search for a solution in a large solution space is generally limited by resource constraints, such as the maximum time allowed and the maximum space available; hence, approximations must be applied to terminate the search when resources are expended. Approximation algorithms generally prune search nodes in a search tree. We are interested in those that give an ascertained (and perhaps minimal) degree of deviation from the optimal solution when the search terminates, assuming that admissible heuristic functions are used.

There are three ways to establish an upper bound on the best ascertained approximation degree from the optimal solution that can be achieved in a time limit. They all require the knowledge of the best approximation degree achievable before a search begins. The best of such strategy-independent approximation algorithms is OPT that "knows" the approximation degree that can be obtained in the time allowed before the search begins, so that all nodes in the open list are either pruned or expanded when time is expended. There are two other strategy-dependent performance upper bounds that have the same or worse behavior as OPT. We define OPT_{A^*} (resp. OPT_{GD}) as an approximation search that achieves the minimum approximation degree when BFS (resp. $GDFS$) is used. When the search strategy is given, *anomalies* in performance may happen; that is, a search with a lower approximation degree may take a shorter time than one with a higher approximation degree [4]. Note that these three upper bounds cannot be achieved in practice, as the best approximation degree achievable before a search begins is unknown. However, they are useful in providing benchmarks for comparison.

Lawler and Wood first proposed an approximate search algorithm in their seminal paper [5]. Their algorithm is based on BFS that works as follows. To solve a problem P under time constraint T, they start BFS without any approximation for $\frac{T}{2}$ units of time. If the optimal solution is not found within $\frac{T}{2}$ units of time, they relax the approximation degree to 5% and try to solve the problem for another $\frac{T}{4}$ units of time. If the 5% approximate solution is not found within $\frac{T}{4}$ units

of time, then they relax the approximation degree by another 5% (*i.e.*, a total of 10%) and try to solve the problem for another $\frac{T}{8}$ units of time. This process is repeated until the time constraint is exceeded or an approximate solution is found. Their algorithm of reducing the approximation degree with respect to search time is suboptimal, since it spends a major portion of its time resource in solving harder problems first. Further, the step size they used (namely, 5%) should not be fixed and should be tuned at run time based on domain knowledge of the target problem.

We have previously studied TCA^* [6], a family of approximation strategies based on BFS. TCA^* allows approximate solutions very close to those obtained by OPT_{A^*} to be found in a given time constraint with a minimal increase in cost as incurred by OPT_{A^*}. Costs in our study were measured in terms of either the cumulative space-time product (CST) or the maximum space required. The algorithm is based on the empirically observed fact that the time spent in a search and the CST cost when the search terminates grow exponentially when the approximation degree decreases, if anomalies are ignored. By applying a principle similar to Korf's iterative deepening [3], time resource is initially allocated to solve simpler problems (with less accurate solutions but lower costs). The approximation degree is progressively reduced in a linear fashion when time is running out. The exponential relationship between cost (in terms of CST product or maximum space incurred) and the approximation degree dictates that the cost of the final search overwhelms the costs of earlier searches. Hence, the solution is only marginally better if the earlier searches were not performed.

In this paper we study $TCGD$, a family of approximation strategies similar to TCA^* [6] but is based on $GDFS$ instead of BFS. In TCA^*, the objective is to minimize the approximation degree when time is expended while minimizing either the CST product or the maximum space incurred during a search. Since the maximum space used in $GDFS$ is bounded and small, and the time constraint is specified, the CST cost and the maximum space incurred are bounded. The sole objective in $TCGD$ is, therefore, to minimize the approximation degree of the search when time is expended. Another major difference between $TCGD$ and TCA^* lies in the lower-bound value obtained when the search terminates. In $GDFS$, the minimum lower bound of all active nodes is not advanced as fast as that of BFS because nodes close to the root may remain unevaluated until late in the search. For this reason, the approximation degree obtained by $TCGD$ when the search terminates is generally worse than that of TCA^*.

Three versions of $TCGD$ are studied in this paper: naive $TCGD$ ($nTCGD$), static $TCGD$ ($sTCGD$), and predictive $TCGD$ ($pTCGD$).

$nTCGD$ simply solves a search problem without any approximation until either time is exceeded or the exact optimal solution is found. If an optimal solution cannot be found when time is exceeded, the incumbent, the corresponding best feasible solution found, and the final run-time approximation degree are reported.

$sTCGD$ uses a *static* strategy to progressively reduce the approximation degree in a linear fashion. Since the approximation degree is a negative exponential of the execution time, the time for solving the last $GDFS$ overwhelms those for solving earlier ones.

$sTCGD$ can be improved by deriving dynamically the profile function relating approximation degrees and costs. Instead of using a static strategy, $pTCGD$ applies

sTCGD for a relatively small amount of time and, based on a parametric relationship between time spent and approximation degree, estimates the approximation degree that can be achieved in the remaining time and solves the search problem with the estimated approximation degree.

The branch-and-bound algorithm used in this paper is implemented as *GDFS*. The target search problems studied are combinatorial optimization problems that have a finite search space and many feasible solutions, such that both upper and lower bounds of search nodes can be computed in polynomial time. This class of problems cover a wide spectrum of important problems in artificial intelligence and operations research. Performance of *TCGD* is demonstrated by simulations of the symmetric traveling salesperson problem (TS); results obtained for the knapsack (KS) and production planning (PP) problems are not shown due to space limitation. The heuristic functions we used may not be the best available. However, they do not affect the validity of our results, since the goal of this paper is to demonstrate the power of *TCGD*.

The symmetric TS problem we use to evaluate our algorithms assume that all cities are fully connected, and that distances between them are symmetric and are generated by a random number generator using the well-known linear congruence algorithm. The property of the triangular inequality is satisfied during the generation of sample problems as all cities are first mapped onto a Cartesian plane before their distances are calculated. The lower-bound value is evaluated by finding the cost of the spanning tree that covers the cities not visited, while 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. Section 3 describes the parameterization of the profiles and demonstrates empirically their aptness and usefulness. Section 4 describes *nTCGD* and *sTCGD* and proves some of their properties. Section 5 describes *pTCGD* and shows examples. Section 6 compares *TCGD*, *TCA**, and the algorithm proposed by Lawler and Wood [5] and draws conclusion. To avoid comparing the original Lawler and Wood's algorithm, which uses exponential amount of space, against *TCGD*, which uses polynomial amount of space, we modify Lawler and Wood's algorithm so that *GDFS* is carried out instead of *BFS*.

2 Approximate Branch-and-Bound Algorithms

A branch-and-bound (B&B) search [5, 7] is a general formulation of a wide range of heuristic searches [8], such as *A** [9], *AO**, *B**, game-tree, and dynamic programming algorithms. A B&B search 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 describing the B&B algorithm. Maximization problems are duals of minimization problems and can be solved by minimizing a negated objective.

A search node i in a B&B search is associated with a lower-bound value f_i. Let

f_i^* be the minimal solution value in the subtree rooted at node i. f_i satisfies the following properties: (a) $f_i \leq f_i^*$; (b) $f_i = f_i^*$ when node i is a goal node, *i.e.*, node i is a feasible solution; and (c) $f_i \leq f_j$ when node i is an ancestor of node j.

The *incumbent* value z is the minimal feasible-solution value found so far in a search. Note that the incumbent value is initially set to infinity in a minimization problem. Any search node i with $f_i \geq z$ is obviously inferior and can be pruned. Let z^* and α be the optimal-solution value of a search and the *approximation degree*, respectively. We have the following pruning rule.

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

The final feasible solution obtained by applying the above pruning rule is called an *α-approximate solution*

When a search is run with approximation degree α and time constraint T, we denote it as $S(\alpha, T)$. If the time constraint is not specified, then we denote it as $S(\alpha, -)$ which is equivalent to $S(\alpha, \infty)$. Let $t(\alpha)$ be the time needed to complete $S(\alpha, -)$. Consider a search instance $S(\alpha, T)$. If $t(\alpha) > T$, then the search is terminated at T, and the following *run-time approximation degree* is computed.

$$\alpha_{run-time}(\alpha, T) = \frac{z(\alpha, T) - f_{global}(\alpha, T)}{f_{global}(\alpha, T)}, \tag{2.2}$$

where $z(\alpha, T)$ is the incumbent value when $S(\alpha, T)$ is stopped at T, and $f_{global}(\alpha, T)$ is the minimum lower-bound value of all active nodes at time T.

3 Parametric Profiles

The *profile* we use in this paper is a trace of resource usage (*i.e.*, execution time incurred) versus solution quality (*i.e.*, approximation degree). Execution time measured in an experiment can be real or virtual. Real time is measured by the built-in timer of the computer on which the search is run and reflects the characteristics of the particular computer involved. In contrast, virtual time is measured by the number of search nodes expanded. It may not be a true indicator of the overhead involved since different computers may spend different amounts of time in expanding a node and for memory management. However, it is useful for reporting results that can be reproduced in the future. For this reason, we present experimental results in this paper based on virtual time.

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 *GDFS* without any approximation. An *actual profile* shows the experimentally obtained performance of OPT_{A^*} or OPT_{GD}, which gives the best approximate solutions with the minimum costs under a given time constraint and search strategy. A *performance profile* is similar to an actual profile except that it is based on a given search strategy, such as *sTCGD* or *pTCGD*. Finally, a *parametric profile* is an estimation of the actual profile and is obtained by regressing on the partial actual profile. The aptness and usefulness of parametric profiles are demonstrated later.

In studying TCA^*, we have proposed two general algebraic profiles [6], one of them is used in this paper.

$$\alpha = \sum_{k=0}^{n} \beta_k log^k t \tag{3.3}$$

where α is the approximation degree, t is the time used, and β_ks are constants.

One difficulty with higher-order regressions in (3.3) is their intractability in prediction. In $pTCGD$ described in Section 5, we collect only partial actual profiles, regress them, and predict the entire profile. Due to the high-order coefficients in the parametric function, the function may sometimes become a convex curve, which changes curvilinearly to a larger approximation degree as time progresses. To avoid this problem, we restrict to using the simple exponential profile in this paper,

$$\alpha = \beta_0 + \beta_1 \, log \, t, \tag{3.4}$$

with the following two boundary conditions,

$$t = \tau_{GD} \geq 1, \quad \alpha = 0; \tag{3.5}$$

$$\text{and} \quad t = 1, \quad \alpha = \alpha_0 \geq 0, \tag{3.6}$$

where τ_{GD} is the time required to find the optimal solution using $GDFS$, and α_0 is the run-time approximation degree obtained after the root has been expanded.

Substituting the boundary conditions into (3.4), we have

$$\alpha(t) = \alpha_0 \, (1 - log_{\tau_{GD}} t) \, . \tag{3.7}$$

Rewriting (3.7) in terms of α, we have

$$t_{GD}(\alpha) = \tau_{GD}^{1-\frac{\alpha}{\alpha_0}} . \tag{3.8}$$

The above condition applies when $GDFS$ is used. A similar condition when A^* is used is

$$t_{A^*}(\alpha) = \tau_{A^*}^{1-\frac{\alpha}{\alpha_0}}, \tag{3.9}$$

where τ_{A^*} is the time required to find the optimal solution using A^*.

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 1 shows the standard deviation (normalized with respect to α_0) between the actual profile and the regressed curve for different values of n in (3.3) and for 10 problem instances of TS of various sizes. 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, (3.4) is a sufficient approximation to the actual profile.

It may be inappropriate for us to conclude from our limited experimental results that the profiles of other combinatorial optimization problems can be parameterized 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

[1] The statement is not true in a search problem with an infinite search space, such as the 15 puzzle problem.

Table 1. Summary of standard deviations between the actual profile and the regressed curve of 10 TS problem instances of size ranging from 11 to 20. The standard deviation is the root-mean-square of the difference between the actual and the parametric profiles.

Std. Dev.	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$
Minimum	0.028	0.028	0.018	0.018	0.018
Average	0.040	0.038	0.030	0.030	0.030
Maximum	0.056	0.054	0.054	0.054	0.054

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 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 hard-constrained optimization problems, such as discrete optimization problems with very few or no feasible solutions. Such problems are not addressed in this paper. Situation (a) can also happen when the lower-bound function is loose in predicting the optimal solution. Situation (b) happens in very hard search problems. In the last two cases, an algorithm which dynamically predicts the profile during a search is needed. This is applied in a *dynamic* version of *TCGD* that is described elsewhere [2, 10]

4 Naive TCGD and Static TCGD

In this section, we describe two simple versions of *TCGD*, *nTCGD* and *sTCGD*. We identify the parameters that significantly affect the performance profile and present experimental results based on the TS problem.

4.1. Naive TCGD

For time constraint T, the naive *TCGD* (or *nTCGD*) algorithm is defined as

$$nTCGD = \{S(0,T)\}. \tag{4.10}$$

Consider problem P solved under time constraint T, *nTCGD* works as follows.

1. Solve P using $S(0,T)$ until either T is expended or the exact optimal solution is found.

2. If the exact optimal solution is found, then report it; otherwise, report the incumbent and the run-time approximation degree computed using (2.2).

If *nTCGD* can find the exact optimal solution under the time constraint for a problem instance, then *nTCGD* is the best algorithm for solving the instance in terms of execution time. However, if time is expended before the exact optimal

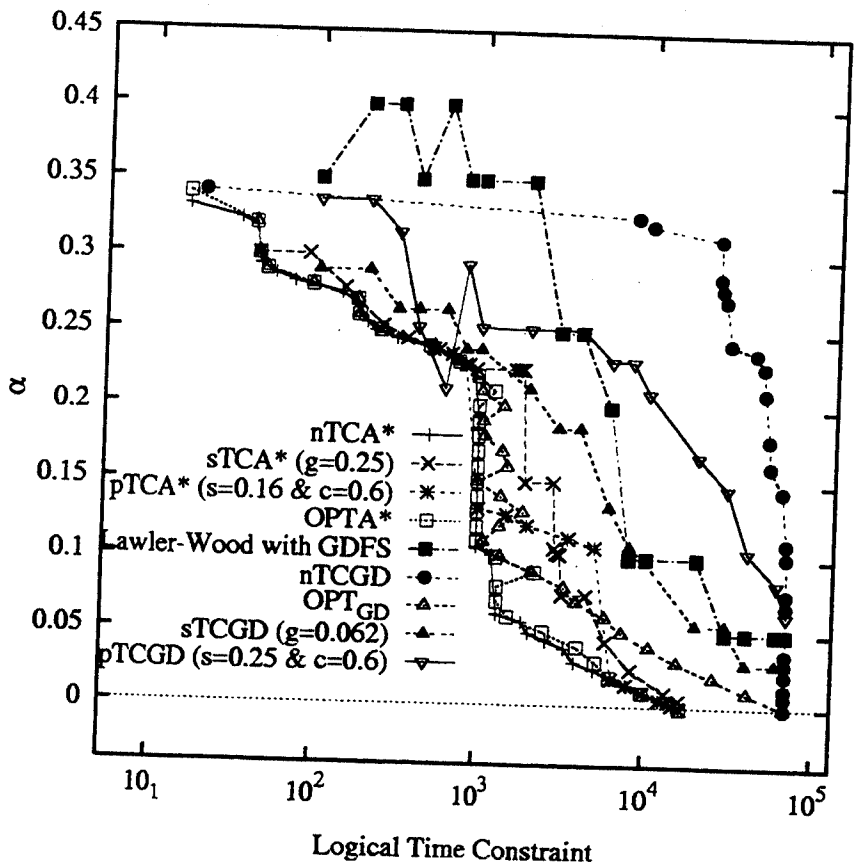

Figure 1. Comparison of performance profiles for a 20-city TS problem instance under various strategies.

solution is found, then the run-time approximation degree obtained is much worse than the best approximation degree achievable when BFS (which approximates OPT_{A^*}) is used instead. This happens because the minimum lower bound of all active nodes in $GDFS$ is usually governed by nodes close to the root, which are not expanded until much later in the search. As a result, the run-time approximation degree achieved by $GDFS$ and DFS are usually worse than that of BFS.

Figure 1 plots the performance profiles of $nTCGD$ for a 20-city TS problem instance. The quality of a solution is determined by its approximation degree achieved within the time constraint. A better algorithm should find more accurate α-approximate solutions in a shorter amount of time. An algorithm whose performance is very close to the actual profile is desirable. As expected, Figure 1 shows that $nTCGD$ has the worst performance.

4.2. Static TCGD

For time constraint T, the static $TCGD$ (or $sTCGD$) algorithm can be defined as

$$sTCGD = \{S(\alpha_0, -), S(\alpha_1, -), \ldots, S(\alpha_k, -), S(\alpha_{k+1}, t_{rem})\}, \quad (4.11)$$

$$\text{where} \quad t_{rem} = T - \sum_{j=0}^{k} t(\alpha_j) > 0 \tag{4.12}$$

$$\text{and} \quad \alpha_0 > \alpha_1 > \ldots > \alpha_k > \alpha_{k+1}. \tag{4.13}$$

The approximation degree achieved by *sTCGD* is

$$\alpha_{sTCGD} = \min\{\alpha_k, \alpha_{run-time}(\alpha_{k+1}, t_{rem})\}. \tag{4.14}$$

The approach used in *sTCGD* is to select a sequence of $\alpha_0, \alpha_1, \ldots, \alpha_k, \alpha_{k+1}$, such that α_{sTCGD} is minimized.

The approximation degree used in the k-th *GDFS*, α_k, is defined as follows.

$$\alpha_k = (1 - k \times g)\alpha_0, \quad 0 < g \leq 1, \tag{4.15}$$

where g is a constant *stepping factor* (or gradient factor).

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

1. In the zeroth search, evaluate the root node and compute its approximation degree α_0. Set i as 0.

2. If the problem is not solved and the time constraint is not violated, then compute α_{i+1} using (4.15). Execute $S(\alpha_{i+1}, -)$.

3. Repeat Step (2) until time constraint T is violated or the exact optimal solution is found. Compute α_{sTCGD} using (4.14).

From the boundary conditions in (3.5), we have

$$g \times n_s = 1, \tag{4.16}$$

where n_s is the number of *GDFS*s that must be evaluated before the optimal solution is found. To achieve the best approximation degree under the time constraint, parameter g must be chosen properly based on the profile such that α_{sTCGD} defined in (4.14) is minimized.

Several important symbols used for the lemmas and theorems in this section are listed in Table 2. In the following discussion, we assume that the profile equation defined in (3.8) is satisfied.

The following lemma shows the relationship among the time for solving the k-th *GDFS*, iteration index k, and stepping factor g.

Lemma 1. *The execution time $t(\alpha_k)$ for solving the k-th GDFS with approximation degree α_k is*

$$t(\alpha_k) = \tau_{GD}^{gk}. \tag{4.17}$$

Proof. This lemma is proved by substituting (4.15) to (3.8). ■

The following lemma shows the lower and upper bounds for the number of completed *GDFS*s in *sTCGD* under time constraint T.

Table 2. Summary of important symbols used in the lemmas and theorems.

Symbol	Meaning
T	Time constraint allowed for the search
τ_{A^*}	Time required to find the optimal solution using A^*
τ_{GD}	Time required to find the optimal solution using $GDFS$
$\alpha_{OPT_{A^*}}$	Approximation degree of OPT_{A^*} under time constraint T
α_{sTCGD}	Approximation degree of $sTCGD$ under time constraint T
α_0	Approximation degree of the root node
α_k	k-th approximation degree in $sTCGD$
g	Stepping factor used in $sTCGD$

Lemma 2. *Assuming the profile equation defined in (3.8) is satisfied, the lower and upper bounds of k, the number of completed $GDFS$s, is*

$$\begin{aligned} k_{min} &= \tfrac{1}{g}\, log_{\tau_{GD}}\,(T(\tau_{GD}^{g}-1)+1)-2 \;\; < k \\ &\leq \tfrac{1}{g}\, log_{\tau_{GD}}\,(T(\tau_{GD}^{g}-1)+1)-1 \;\; = k_{max}, \end{aligned} \tag{4.18}$$

if g is not equal to 0.

Proof. Assume that $k+1$ $GDFS$s, $S(\alpha_0,-),\ldots,S(\alpha_k,-)$, are done to completion, and that $S(\alpha_{k+1},-)$ is terminated prematurely because time is expended. (See (4.12).) The following inequalities must be satisfied.

$$\sum_{j=0}^{k} t(\alpha_j) \leq T < \sum_{j=0}^{k+1} t(\alpha_j). \tag{4.19}$$

Applying (4.17) and simplifying the geometric series, we can rewrite (4.19) as:

$$\frac{\tau_{GD}^{g(k+1)}-1}{\tau_{GD}^{g}-1} \leq T < \frac{\tau_{GD}^{g(k+2)}-1}{\tau_{GD}^{g}-1}. \tag{4.20}$$

The lemma is proved by rewriting (4.20) into an inequality with respect to k. ■

The lower bound of k defined in (4.18) may be negative if stepping factor g is improperly large or time constraint T is too small, which are not interesting conditions. When the time constraint is fairly large, the lower bound will be positive. For $T \geq 1$, the upper bound of k defined in (4.18) monotonically increases as time constraint T increases. Hereafter, T is assumed to be larger than or equal to 1. When T is equal to the minimal value, *i.e.*, 1, this upper bound is equal to zero. Therefore, the upper bound of k defined in (4.18) is non-negative.

If the stepping factor g is in its *operational range*, *i.e.*,

$$g \in (0, log_{\tau_{GD}}(T-1)], \tag{4.21}$$

then the lower bound of k defined in (4.18) is non-negative. The operational range of the stepping factor can be derived by simply solving the inequality that the lower

bound of k defined in (4.18) is greater than or equal to 0. Note that the operational range of the stepping factor is dependent on the time constraint. The worst-case value of α_k is given as follows.

$$\alpha_k < \alpha_0(1 - g \times k_{min}). \tag{4.22}$$

The following theorem shows that the difference between α_{sTCGD} and $\alpha_{OPT_{A^*}}$ is bounded from above by a function strongly dependent on the stepping factor but weakly dependent on the time constraint.

Theorem 1. *The* least *upper bound of the difference in approximation degrees between* α_{sTCGD} *and* $\alpha_{OPT_{A^*}}$ *is*

$$\alpha_{sTCGD} - \alpha_{OPT_{A^*}} \leq F(g,T) + G(T) \tag{4.23}$$

where
$$F(g,T) = 2\alpha_0 g - \alpha_0 log_{\tau_{GD}} \left(\tau_{GD}^g - 1 + \frac{1}{T} \right), \tag{4.24}$$

$$G(T) = \alpha_0 \left(log_{\tau_{A^*}} T - log_{\tau_{GD}} T \right). \tag{4.25}$$

Proof. Assume that $k+1$ $GDFS$s, $S(\alpha_0, -), \ldots, S(\alpha_k, -)$, are done to completion, and that $S(\alpha_{k+1}, -)$ is terminated prematurely because time is expended. (See (4.12).) We compute α_k as a pessimistic measure of α_{sTCGD} (instead of using the exact formula given by (4.14)). The worst-case α_k is

$$\begin{aligned} \alpha_{sTCGD}^{k_{min}} &= \alpha_0(1 - g \times k_{min}) \\ &= \alpha_0 \left[1 - log_{\tau_{GD}} (T(\tau_{GD}^g - 1) + 1) + 2g \right] \\ &= \alpha_0 \left[1 + 2g - log_{\tau_{GD}} \left(\tau_{GD}^g - 1 + \frac{1}{T} \right) - log_{\tau_{GD}} T \right]. \end{aligned} \tag{4.26}$$

Applying (3.9), $\alpha_{OPT_{A^*}}$ is

$$\alpha_{OPT_{A^*}} = \alpha_0 \left(1 - log_{\tau_{A^*}} T \right). \tag{4.27}$$

The difference in approximation degrees can be obtained by simplifying the difference between (4.26) and (4.27). The bound in (4.23) is tight, since in some cases the $(k+1)$-th search cannot find more accurate solution than the k-th search due to the time constraint (4.12). ■

Theorem 1 shows that the maximum difference between the approximation degrees obtained by $sTCGD$ and that by OPT_{A^*} is dependent on g as well as on T. Note that both $F(g,T)$ and $G(T)$ are positive. The maximum difference is roughly logarithmic with respect to time constraint T, since the effect of T in $F(g,T)$ is very small and negligible, and $G(T)$ can be rewritten into

$$G(T) = log_{\tau_{GD}} T \left[(log_{\tau_{A^*}} \tau_{GD}) - 1 \right]. \tag{4.28}$$

As a result, as time constraint T becomes larger, the upper bound will become looser. The difference in approximation degrees is roughly linear with respect to stepping factor g when g is large, since $\lim_{g \to 1} F(g,T) = \alpha_0 g$. When g is close to

zero, the difference approaches $\alpha_0 log_{\tau_A*} T$, which is very large in terms of approximation. Note that g cannot be zero; otherwise, the lower bound of k is not well defined. When g is in between 0 and 1, the difference in approximation degrees is curvilinear with g. There exists a best choice of stepping factor such that the upper-bound difference is minimal.

Using the upper-bound difference derived in Theorem 1, we can derive g^*, the optimal value of g that minimizes the difference. Such an optimal g represents a pessimistic estimate of the minimum difference in approximation degrees because it is derived based on the upper-bound difference rather than the exact difference. The following lemma shows that g^* should be kept small.

Lemma 3. *In terms of the upper bound of the difference in approximation degrees, the best stepping factor g^* is*

$$g^* = log_{\tau_{GD}} \left[2 \left(1 - \frac{1}{T} \right) \right] \tag{4.29}$$

Proof. Let $Q(g,T)$ be the difference in approximation degrees defined in (4.23). Taking partial directives of $Q(g,T)$ with respect to g, we have

$$\frac{\partial Q(g,T)}{\partial g} = \frac{\partial F(g,T)}{\partial g} = 2\alpha_0 - \frac{\alpha_0 \tau_{GD}^g}{\tau_{GD}^g - 1 + \frac{1}{T}} \tag{4.30}$$

The lemma is proved by setting (4.30) to zero and solving for g^*. ■

One problem with the above lemma is that τ_{GD} is unknown until the search is completed. However, it shows that the effect of time constraint on g^* is small, and that g^* is independent of α_0. These imply that g^* is robust, and a suboptimal choice of g may not have a significant effect on the upper bound in (4.23). By using the best stepping factor, the following theorem shows that the minimum difference in approximation degrees is very small.

Theorem 2. *The minimum difference in approximation degrees between α_{OPT_A*} and $\alpha_{sTCGD}(g^*)$ is bounded by a constant,* i.e.,

$$\alpha_{sTCGD}(g^*) - \alpha_{OPT_A*} < 2\alpha_0 log_{\tau_{GD}} 2 + G(T) \tag{4.31}$$

where $\alpha_{sTCGD}(g^*)$ is the approximation degree obtained by sTCGD using g^* and $G(T)$ is defined in Theorem 1.

Proof. The theorem is proved by substituting g^* in (4.29) into (4.23), leading to

$$\alpha_{sTCGD}(g^*) - \alpha_{OPT_A*} \leq \alpha_0 log_{\tau_{GD}} \left(1 - \frac{1}{T} \right) + 2\alpha_0 log_{\tau_{GD}} 2 + G(T) \tag{4.32}$$

The first term on the right-hand side is negative since $1 - \frac{1}{T}$ is smaller than 1 and τ_{GD} is no less than 1. Hence, this term can be neglected and the inequality (without the equality sign) still holds. ■

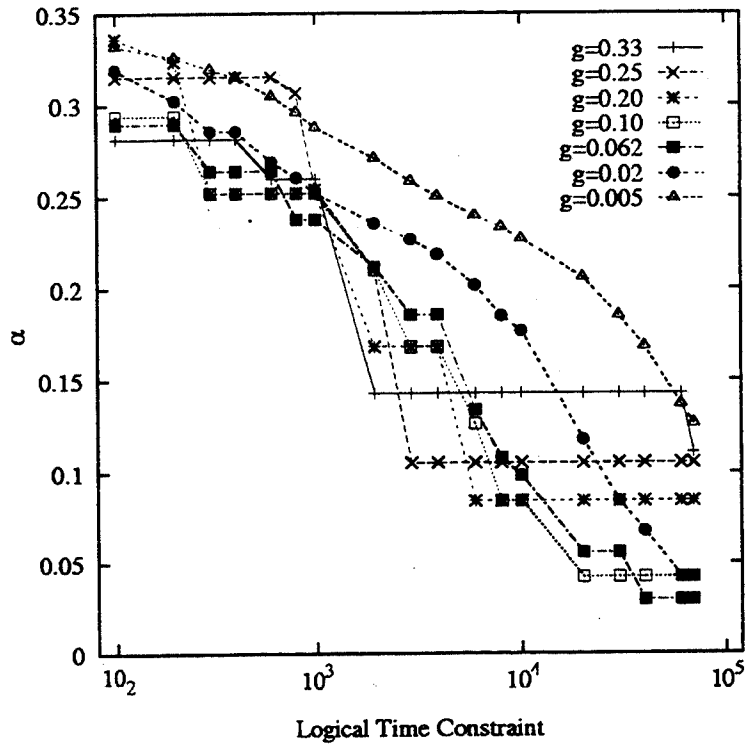

Figure 2. Performance profiles of *sTCGD* for a 20-city TS problem instance under various strategies.

Theorem 2 shows that *sTCGD* is very powerful, and the difference in approximation degrees is small if the best stepping factor is used. It is interesting to note that the best bound is *looser* than that derived for TCA^* in the sense that $G(T)$ appears in the bound for *sTCGD* and not in that for TCA^*. Also note that this best bound is logarithmic with respect to time constraint T ((4.28) and (4.31)). The larger is time constraint T, the looser will be the best bound.

sTCGD (and $sTCA^*$) has the *warm start* problem at the beginning of a new *GDFS* because it must spend time initially to estimate the approximation degree of the last *GDFS*. This problem is more serious when g is small. However, choosing large gs may not allow the last search to be completed or may result in a large part of the remaining time unused. A succinct choice of g is, therefore, essential.

Figure 2 shows the performance profiles of *sTCGD* for a TS problem instance of size 20, where $\alpha_0 = 0.42$ and $\tau_{GD} = 69,154$. According to Lemma 3, the best stepping factor $g^* = 0.062$ for T=69,154. The performance profile corresponding to g^* is generally better than others, since it achieves better approximation degrees in a wider range of time constraints. Note that the axis of time constraints is logarithmically scaled such that the rightmost region in Figure 2 covers almost all the time span. One possible reason why the performance profile for g^* is not the best for small time constraints is that the effect of warm start is more prominent under small time constraints.

One interesting point in Figure 2 is the trend of performance profiles with respect to stepping factors. A performance profile improves as the stepping factor

is decreased. This phenomenon is illustrated by curves with g of 0.33, 0.25, 0.2, 0.1 and 0.062. As the stepping factor decreases further, the performance profiles worsen. This is illustrated by the curves with g of 0.02 and 0.005.

When $g = 0.062$, the performance of $sTCGD$ is shown in Figure 1. According to Theorem 2, the upper bound of the difference in approximation degrees is 0.052. Figure 1 shows that the performance profile corresponding to g^* sometimes has larger difference in approximation degrees than 0.052. This happens because our exponential model of the actual profile is an approximation to the actual behavior, and the difference between the parametric and actual profiles is not included in our analysis. Figure 1 shows that the difference in approximation degrees between the actual and performance profiles for g^* is generally independent of the time constraint, and that most of them are well bounded by the upper bound 0.052.

5 Predictive TCGD

With time constraint T, the predictive $TCGD$ (or $pTCGD$) algorithm is defined as

$$pTCGD = \{sTCGD(0, T_p), S(\alpha_{pred}, T - T_p)\} \tag{5.33}$$

where $T_p \leq T$.

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

1. *Profiling phase.* Execute $sTCGD$ using a relatively small amount of time $T_p = s \times T$, $s < 1$. Collect the partial actual profile.

2. *Regression phase.* Using (3.8), estimate the entire parametric profile by regressing from the partial actual profile collected in the profiling phase.

3. *Prediction phase.* Predict α_{pred}, the best approximation degree 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 $\hat{\alpha} = c \times \alpha_{pred}$, where c is a positive real number around 1. If $\hat{\alpha} > \alpha_{sTCGD}$, then stop the search, and the final approximation degree is α_{sTCGD}.

4. *Solution phase.* Execute $S(\hat{\alpha}, T - T_p)$ in the remaining time until either time has expended or the α-approximate solution with the predicted approximation degree α is found. If P is not completely solved when time is expended, compute the run-time approximation degree using (2.2). The final approximation degree achieved is the minimum of α_{sTCGD} and $\alpha_{run-time}(\hat{\alpha}, T - T_p)$.

Note that during the Profiling Phase of $pTCA^*$ [6], $nTCA^*$ is used for collecting the partial *run-time* profile. In contrast, $sTCGD$ is used for collecting the partial *actual* profile here.

Two parameters in $pTCGD$, 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, whereas the *corrective factor* c ($> 1, = 1$, or < 1) adjusts the predicted best approximation degree to avoid overshooting or undershooting.

The selection of the stopping factor will significantly affect the prediction as well as the accuracy of the solution. The larger the stopping factor is, the more accurate will be the prediction. However, when more time is spent in profiling, less time will be spent in the solution phase. A tradeoff 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 ($s_a = \frac{T_s}{T}$, where T_s is the maximum allowable time for profiling). A hybrid between relative and absolute can be chosen as follows.

$$s = \min\{s_r, s_a\} \tag{5.34}$$

For large T and large problems, choosing an absolute stopping factor is preferable because it avoids spending a significant amount of time in the profiling phase. In contrast, for small T, choosing an amount of time relative to T for profiling is better. A hybrid between relative and absolute is, therefore, a good compromise.

pTCGD suffers from the *cold start* problem that happens because the actual or run-time profiles cannot be linearly regressed well for small time constraints. This inaccurate prediction can cause either an overshoot or an undershoot when time is expended. When there is an overshoot, the search does not complete and reports a poor run-time approximation degree. An undershoot causes the search to be done with a loose approximation degree and results in a poor α-approximate solution.

There are two solutions to alleviate the cold start problem. First, the stopping factor must be chosen large enough so that sufficient profiling is performed. Second, the corrective factor should be chosen properly to avoid overshoot or undershoot. A small corrective factor will likely cause an overshoot in prediction, whereas a large corrective factor will likely cause an undershoot.

To see the effects of the stopping and corrective factors on approximation degrees, performance profiles for various factors are plotted. Figure 3 shows the performance profiles for various stopping factors using $c = 0.6$. Figure 4 shows the performance profiles for various corrective factors using $s = 0.25$. Smaller corrective factors result in better approximation (Figure 3), although there is no significant difference in approximation degrees among the different stopping factors (Figure 4) when the time constraint is large. These results indicate that a good compromise between cost and approximation degree is to have $s = 0.25$ and $c = 0.6$.

With the corrective and stopping factors selected as above, the performance of *pTCGD* is plotted in Figure 1. Due to the cold-start problem and poor run-time approximation, *pTCGD* generally does not perform as well as *sTCGD* and the modified Lawler and Wood's algorithm based on *GDFS*.

6 Concluding Remarks

In this paper, we have presented *TCGD*, a family of problem-independent, time-constrained, approximate *GDFS* algorithms. Three versions are studied: naive *TCGD* (*nTCGD*), static *TCGD* (*sTCGD*), and predictive *TCGD* (*pTCGD*). They are designed to solve combinatorial optimization problems using bounded memory space, with an objective of maximizing the degree of accuracy achievable

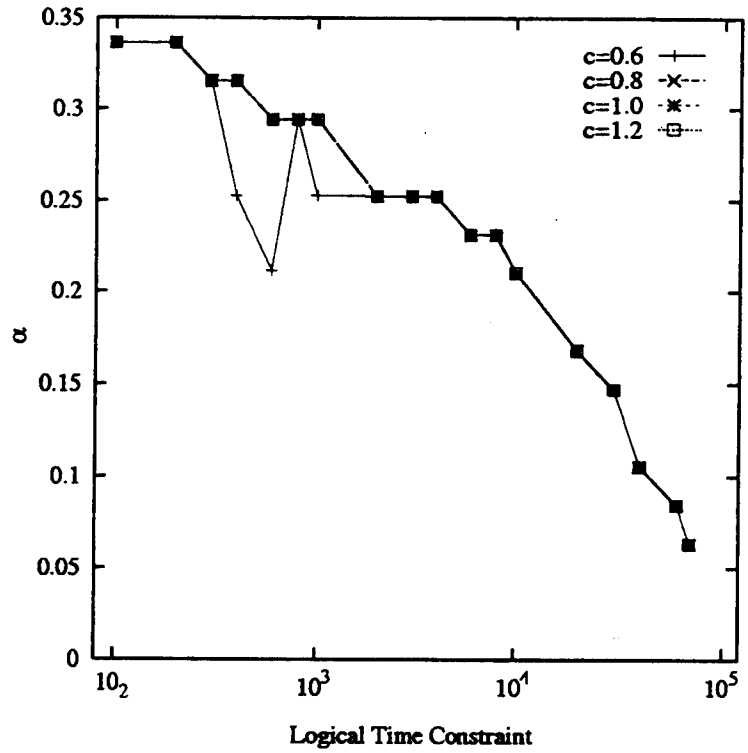

Figure 3. Performance profiles of $pTCGD$ with $s = 0.25$ for a 20-city TS problem instance under various strategies.

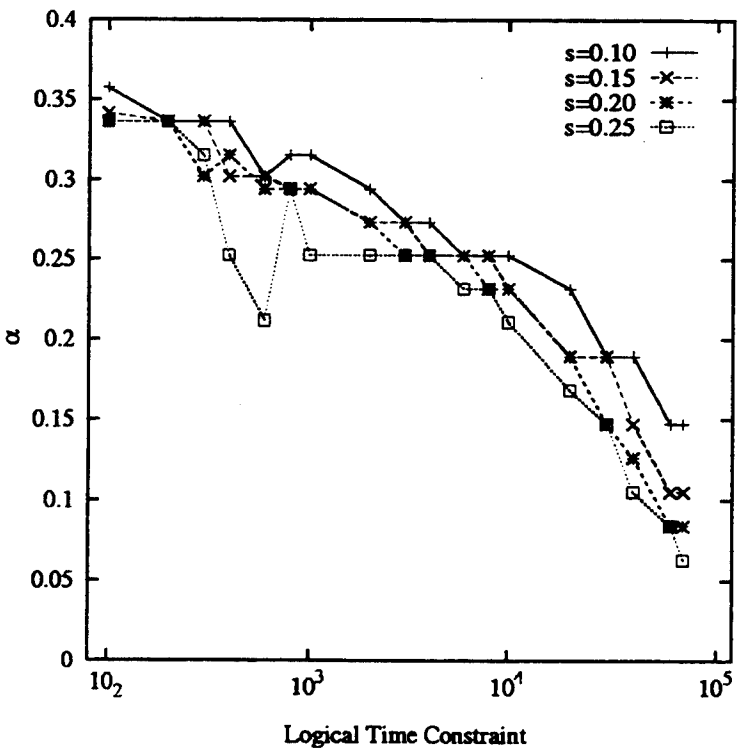

Figure 4. Performance profiles of $pTCGD$ with $c = 0.6$ for a 20-city TS problem instance under various strategies.

within the given time constraint. Among these three versions, only *sTCGD* achieves this goal by progressively increasing the accuracy of solutions found.

Comparing our methods against the modified Lawler and Wood's time constrained search based on *GDFS*, we found that *sTCGD* consistently performs better and has performance very close to that of OPT_{GD}. In contrast, *pTCGD*'s performance is fair and is comparable to that of the modified Lawler and Wood's algorithm. Finally, *nTCGD* has the worst performance.

The worse performance of the modified Lawler and Wood's algorithm is attributed to the large, inflexible step size used and to the search of more accurate solutions in the beginning of the process (which uses time without finding good solutions). Instead of searching for more accurate solutions first, *sTCGD* searches for less accurate solutions in the beginning; hence, time is consumed slowly in gradually approaching the maximum achievable degree of accuracy.

References

[1] B. W. Wah and L.-C. Chu, *TCGD: A time-constrained approximate guided depth-first search algorithm*, Proc. Int'l Computer Symp., (Hsinchu, Taiwan), Tsing Hua Univ. Dec. 1990 507–516.

[2] L.-C. Chu and B. W. Wah, *Optimization in real time*, Proc. Real Time Systems Symp., IEEE Nov. 1991 150–159.

[3] R. E. Korf, *Depth-first iterative deepening: An optimal admissible tree search*, Artificial Intelligence **27** North-Holland (1985) 97–109.

[4] G. J. Li and B. W. Wah, *Computational efficiency of combinatorial OR-tree searches*, IEEE Trans. on Software Engineering **16** IEEE Jan. 1990 13–31.

[5] E. L. Lawler and D. W. Wood, *Branch and bound methods: A survey*, Operations Research **14** ORSA (1966) 699–719.

[6] B. W. Wah and L.-C. Chu, *TCA*–A time-constrained approximate A* search algorithm*, Proc. Int'l Workshop on Tools for Artificial Intelligence, IEEE Nov. 1990 314–320.

[7] L. G. Mitten, *Branch-and-bound methods: General formulation and properties*, Oper. Res. **18** (1970) 23–34.

[8] V. Kumar and L. N. Kanal, *A general branch and bound formulation for understanding and synthesizing and/or tree search procedures*, Artificial Intelligence **21**, no. 1-2 North-Holland (1983) 179–198.

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

[10] B. W. Wah and L.-C. Chu, *Time constrained combinatorial search*, IEEE Trans.