Proceedings of International Computer Symposium 1990
December 17-19, Hsinchu, Taiwan, R.O.C.

TCGD: A TIME-CONSTRAINED APPROXIMATE GUIDED DEPTH-FIRST 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
Urbana, IL 61801
wah%aquinas@uxc.cso.uiuc.edu

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 degree of approximation 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 the degree of approximation decreases, although anomalies may happen. The algorithms we study are evaluated by simulations using the symmetric traveling-salesperson problem.

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

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 which 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*) [1]. In this paper, we focus only on guided depth-first searches. The difference between a DFS and a GDFS is that in the latter search all immediate descendents of a node selected for expansion are expanded first before proceeding to descendents of expanded nodes, whereas in the former search only one immediate descendent is selected for expansion before descendents of this expanded node is examined. In general, GDFS and DFS require much less memory space than BFS, although they generally expand more nodes because they may need to explore deep in the search tree before finding a solution. A GDFS expands less nodes than a 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 the search tree. We are interested in ones 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. The best of such approximation algorithms is OPT, which "knows" the degree of approximation 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. OPT can provide a performance upper bound and is strategy independent.

A strategy-dependent performance upper bound, which has the same or worse behavior as OPT, can also be established. In this paper, we define OPT_{A*} (resp. OPT_{GD}) as an approximation search which achieve the minimum degree of approximation when a BFS (resp. GDFS) strategy is used. When the search strategy is given; anomalies in performance may happen; that is, a search with a lower degree of approximation may take a shorter time than one with a higher degree of approximation [4]. The performance bounds defined, namely, OPT, OPT_{A*}, and OPT_{GD}, cannot be achieved in practice, as they require the knowledge of the best degree of approximation before the search begins. However, they are useful in providing benchmarks for comparison.

Lawler and Wood first proposed an approximate search algorithm in their seminal paper [3]. Their

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

algorithm is based on a best-first search strategy and works as follows. To solve a problem P under time constraint T, they start a best-first 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 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 the time constraint is exceeded or an approximate solution is found. Their algorithm of reducing the degree of approximation with respect to the 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 the domain knowledge of the target problem.

We have previously studied TCA* [7], a family of approximation strategies based on best-first search which allows approximate solutions very close to those obtained by OPT_A* to be found in the 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 degree of approximation decreases, if anomalies are ignored.* By applying a principle similar to Korf's iterative deepening [1], time resource is initially allocated to solving simpler problems (with less accurate solutions but lower costs). The degree of approximation is progressively reduced in a linear fashion when time resource is running out. The exponential relationship between the cost (in terms of CST product or the maximum space incurred) and the degree of approximation 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* [7] but is based on GDFSs instead of BFSs. In TCA*, the objective is to minimize the degree of approximation when time is expended while minimizing either the CST product or the maximum space incurred during the 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 degree of approximation 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 a GDFS, the minimum lower bound of all active nodes is not advanced as fast as that of a best-first search because nodes close to the root may remain unevaluated until late in the search. For this reason, the degree of approximation obtained by TCGD when the search terminates is generally worse than that of TCA*.

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

The *n*TCGD algorithm simply solves the search problem without any approximation until either the time resource is exceeded or the exact optimal solution is found. If an optimal solution cannot be found when the time resource is exceeded, the incumbent, the corresponding best feasible solution found, and the final degree of run-time approximation are reported.

The *s*TCGD algorithm uses a *static* strategy to progressively reduce the degree of approximation in a linear fashion. Since the degree of approximation is a negative exponential of the execution time, the time for solving the final GDFS overwhelms those for solving the earlier ones.

The *s*TCGD algorithm can be improved by deriving dynamically the profile function relating the approximation degree and the cost. Instead of using a static strategy, the *p*TCGD algorithm applies *s*TCGD for a relatively small amount of time and, based on a parametric relationship between the time spent and the degree of approximation, estimates the degree of approximation that can be achieved in the remaining time and solves the search problem with the estimated degree of approximation.

The GDFS is implemented as a branch-and-bound algorithm in this paper. The target search problems studied are combinatorial optimization problems which 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 covers 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 the random number generator using the well-known linear congruence algorithm. The property of the triangular inequality is satisfied during the generation of the 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 parametrization of the profiles and demonstrates empirically their aptness and usefulness. Section 4 describes the *n*TCGD and *s*TCGD and shows examples. Section 5 describes the *p*TCGD and shows examples. Section 6 compares the TCGD, TCA*, and the algorithm proposed by Lawler and Wood [3], 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 a GDFS is carried out instead of a BFS.

2. APPROXIMATE BRANCH-AND-BOUND ALGORITHMS

A branch-and-bound (B&B) search [3,5] is a general formulation of a wide range of heuristic searches [2], such as A* [6], AO*, B*, game-tree, and dynamic programming algorithms. A B&B search decomposes a problem into smaller subproblems and repeatedly decomposes them until 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 function.

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) $f_i \le f_i^*$; (b) $f_i = f_i^*$ when node i is a goal node, *i.e.*, node i is a feasible solution; and (c) $f_i \le f_j$ 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 $f_i \ge z$ is obviously inferior and can be pruned. Let z^* and α be the optimal-solution value of the search and the degree of approximation, respectively. We have the following pruning rule.

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

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

When a search is run with an approximation degree α and a 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.

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

where $z(\alpha, T)$ is the incumbent value when $S(\alpha, T)$ is stopped at T, and $f_{\text{global}}(\alpha, T)$ 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.*, degree of approximation). Execution time measured in an experiment can be real or virtual. Real time is measured by the built-in timer of the system, while virtual time is measured by the number of search nodes expanded. Real time reflects the characteristics of the particular computer in which the search is run, while virtual time is the same for all computers. Virtual time may not be a true indication of the overhead involved since different computers may spend different amounts of time in expanding a node and also may have different overhead for the memory management. Experimental results described in this paper are presented in virtual time, so they will not be implementation and computer dependent.

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 the 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 *s*TCGD or *p*TCGD. 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 the parametric profiles are demonstrated later.

In studying TCA*, we have proposed two general algebraic profiles [7], one of them is used in this paper.

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

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

One difficulty with higher-order regressions defined in Eq. (3.1) is their intractability in prediction. In *p*TCGD 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 degree of approximation as time progresses. To avoid this problem, we restrict to using the simple exponential profile in this paper,

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

with the following two boundary conditions,

$$t = \tau_{GD} \geq 1; \quad \alpha = 0; \text{ and} \tag{3.3a}$$

$$t = 1; \quad \alpha = \alpha_0 \geq 0, \tag{3.3b}$$

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 Eq. (3.2), we have

$$\alpha(t) = \alpha_0 \left[1 - \log_{\tau_{GD}} t\right]. \tag{3.4}$$

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

$$t_{GD}(\alpha) = \tau_{GD}^{\;1-\frac{\alpha}{\alpha_0}}. \tag{3.5}$$

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

$$t_{A^*}(\alpha) = \tau_{A^*}^{\;1-\frac{\alpha}{\alpha_0}}, \tag{3.6}$$

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 3.1 shows the standard deviation (normalized with respect to α_0) between the actual profile and the regressed curve for different values of n in Eq. (3.1) 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, Eq. (3.2) 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 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.* 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 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 the search is needed. This is applied in the *dynamic* TCGD algorithm, which will be described in a future paper.

Table 3.1. Summary of standard deviations between the actual profile and the regressed curve of 10 traveling-salesperson problem instances of size ranging from 11 to 20. The standard deviation is the root-mean-square of the difference between the actual and 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

4. NAIVE TCGD AND STATIC TCGD

In this section, two simple versions of TCGD, *n*TCGD and *s*TCGD, are described. The parameters that significantly affect the performance profile are identified. Finally, experimental results based on the TS problem are presented.

4.1. Naive TCGD

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

$$n\text{TCGD} = \{ S(0, T) \}. \tag{4.1}$$

Consider an optimization problem P to be solved under time constraint T, *n*TCGD 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 Eq. (2.2).

If *n*TCGD can solve the exact optimal solution under the time constraint, then *n*TCGD is the best algorithm for solving this problem instance in terms of the execution time incurred. However, if time is expended before the exact optimal solution is found, then the run-time approximation degree obtained is much worse than the best degree of approximation achievable when a BFS (which approximates OPT_{A^*}) is used instead. This happens because the minimum lower bound of all active nodes in a 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.

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

The performance profiles of *n*TCGD are plotted in Figures 4.1. The quality of a solution is determined by its degree of approximation achieved within the time constraint. A better algorithm should find more accurate semi-optimal solution in a shorter amount of time. An algorithm whose performance is very close to the actual profile is desirable. As expected, Figures 4.1 shows that *n*TCGD has the worst performance.

4.2. Static TCGD

For a time constraint T, the static TCGD (or *s*TCGD) algorithm can be defined as

$$sTCGD = \{ S(\alpha_0,-), S(\alpha_1,-), S(\alpha_2,-), \cdots,$$
$$S(\alpha_k,-), S(\alpha_{k+1}, t_{rem}) \}, \qquad (4.2)$$

$$\text{where } t_{rem} = T - \sum_{j=0}^{k} t(\alpha_j) > 0 \qquad (4.3)$$

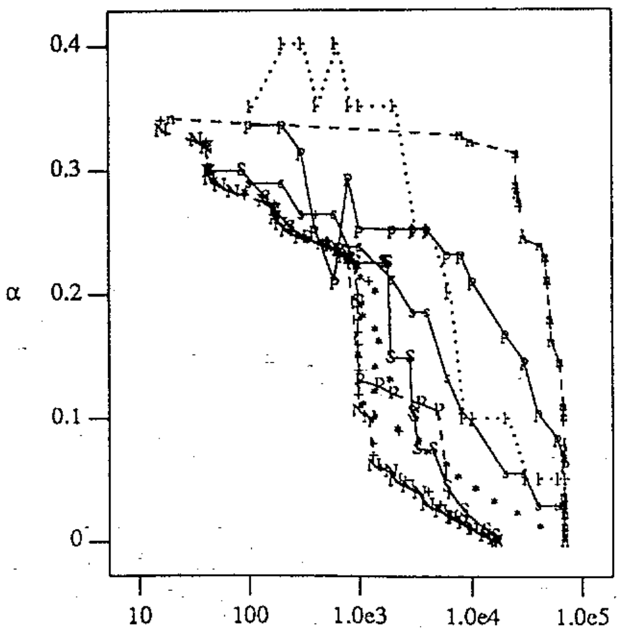

Figure 4.1. Comparison of performance profiles in approximation degree versus time constraint for a TS problem instance of size 20 using OPT$_{GD}$ (*), modified Lawler and Wood's method based on GDFS (l), *n*TCGD (n), *s*TCGD (s, with g=0.062), *p*TCGD (p, with s=0.25 and c=0.6), OPT$_{A^*}$ (+), *n*TCA* (N), *s*TCA* (S, with g=0.25), and *p*TCA* (P, with s=0.15 and c=0.6).

$$\text{and } \alpha_0 > \alpha_1 > \cdots > \alpha_k > \alpha_{k+1}. \qquad (4.4)$$

The degree of approximation achieved by *s*TCGD is

$$\alpha_{sTCGD} = \min\{ \alpha_k, \alpha_{run-time}(\alpha_{k+1}, t_{rem}) \}. \qquad (4.5)$$

The approach used in *s*TCGD is to select a sequence of α_0, α_1, ..., α_k, α_{k+1}, such that α_{sTCGD} is minimized.

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

$$\alpha_k = (1 - k \times g)\alpha_0 \qquad 0 < g \le 1, \qquad (4.6)$$

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

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

(1) In the zeroth search, evaluate the root node and compute its degree of approximation 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 Eq. (4.6). Execute $S(\alpha_{i+1},-)$.
(3) Repeat Step (2) until the time constraint T is violated or the exact optimal solution is found. Compute α_{sTCGD} using Eq. (4.5).

From the boundary conditions in Eq. (3.3), we have

$$g \times n_s = 1, \qquad (4.7)$$

where n_s is the number of GDFSs that must be done before the optimal solution is found. To achieve the best degree of approximation under the time constraint, parameter g must be chosen properly based on the profile such that α_{sTCGD} defined in Eq. (4.5) is minimized.

Several important symbols used for the lemmas and theorems in this section are listed in Table 4.1. In the following discussion, it is assumed that the profile equation defined in Eq. (3.5) 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 4.1. The execution time $t(\alpha_k)$ for solving the k-th GDFS with approximation degree α_k is

$$t(\alpha_k) = \tau_{GD}^{\,g \times k}. \qquad (4.8)$$

Proof. This lemma is proved by substituting Eq. (4.6) to Eq. (3.5). □

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

Lemma 4.2. Assuming the profile equation defined in Eq. (3.5) is satisfied, the lower and upper bounds of k, the number of completed GDFSs, is

Table 4.1. 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 *s*TCGD under time constraint T.
α_0	Approximation degree at root node.
α_k	k-th approximation degree in *s*TCGD.
g	Stepping factor used in *s*TCGD.

$$k_{\min} = \frac{1}{g}\log_{\tau_{GD}}\left[T(\tau_{GD}{}^{g}-1)+1\right]-2 < k$$

$$\le \frac{1}{g}\log_{\tau_{GD}}\left[T(\tau_{GD}{}^{g}-1)+1\right]-1 = k_{\max}, \qquad (4.9)$$

if g is not equal to 0.

Proof. Assume that k+1 GDFSs, $S(\alpha_0,-)$, ..., $S(\alpha_k,-)$, are done to completion, and that $S(\alpha_{k+1},-)$ is terminated prematurely because time is expended (Eq. (4.3)). The following inequalities must be satisfied.

$$\sum_{j=0}^{k} t(\alpha_j) \le T < \sum_{j=0}^{k+1} t(\alpha_j). \qquad (4.10)$$

Applying Eq. (4.8) and simplifying the geometric series, we can rewrite Eq. (4.10) as follows.

$$\frac{\tau_{GD}{}^{g(k+1)}-1}{\tau_{GD}{}^{g}-1} \le T < \frac{\tau_{GD}{}^{g(k+2)}-1}{\tau_{GD}{}^{g}-1}. \qquad (4.11)$$

The lemma is proved by rewriting Eq. (4.11) into an inequality with respect to k. □

The lower bound of k defined in Eq. (4.9) may be negative if the stepping factor g is improperly large or the 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 $T \ge 1$, the upper bound of k defined in Eq. (4.9) monotonically increases as the 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 Eq. (4.9) is non-negative.

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

$$g \in (0, \log_{\tau_{GD}}(T-1)], \qquad (4.12)$$

then the lower bound of k defined in Eq. (4.9) 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 Eq. (4.9) 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.

$$\alpha_k < \alpha_0(1-g\times k_{\min}). \qquad (4.13)$$

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 4.1. The *least* upper bound of the difference in approximation degrees between α_{sTCGD} and $\alpha_{OPT_A^*}$ is

$$\alpha_{sTCGD} - \alpha_{OPT_A^*} \le F(g,T) + G(T) \qquad (4.14)$$

$$\text{where } F(g,T) = 2\alpha_0 g - \alpha_0 \log_{\tau_{GD}}\left[\tau_{GD}{}^{g}-1+\frac{1}{T}\right] \qquad (4.15)$$

$$G(T) = \alpha_0\left[\log_{\tau_{A^*}} T - \log_{\tau_{GD}} T\right]. \qquad (4.16)$$

Proof. Assume that k+1 GDFSs, $S(\alpha_0,-)$, ..., $S(\alpha_k,-)$, are done to completion, and that $S(\alpha_{k+1},-)$ is terminated prematurely because time is expended (Eq. (4.3)). We compute α_k as a pessimistic measure of α_{sTCGD} (instead of using the exact formula given by Eq. (4.5)). The worst-case α_k is

$$\alpha_{sTCGD}^{k_{\min}} = \alpha_0(1-g\times k_{\min}) \qquad (4.17)$$

$$= \alpha_0\left[1-\log_{\tau_{GD}}\{T(\tau_{GD}{}^{g}-1)+1\}+2g\right]$$

$$= \alpha_0\left[1+2g-\log_{\tau_{GD}}\left[\tau_{GD}{}^{g}-1+\frac{1}{T}\right]-\log_{\tau_{GD}} T\right].$$

Applying Eq. (3.6), $\alpha_{OPT_A^*}$ is

$$\alpha_{OPT_A^*} = \alpha_0\left[1-\log_{\tau_{A^*}} T\right]. \qquad (4.18)$$

The difference in approximation degrees can be obtained by simplifying the difference between Eq's (4.17) and (4.18). The bound in Eq. (4.14) 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 (Eq. (4.3)). □

Theorem 4.1 shows that the maximum difference between the degrees of approximation obtained by *s*TCGD and that by OPT$_A$* is dependent on the stepping factor g as well as on the time constraint T. Note that both $F(g,T)$ and $G(T)$ are positive. The maximum difference is roughly logarithmic with respect to the time constraint T, since the effect of T in $F(g,T)$ is very small and negligible, and $G(T)$ can be rewritten into

$$G(T) = \log_{\tau_{GD}} T \left[(\log_{\tau_{A^*}} \tau_{GD}) - 1 \right]. \tag{4.19}$$

As a result, as the time constraint T becomes larger, the upper bound will become looser. The difference in approximation degrees is roughly linear with respect to the 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 4.1, we can derive g^*, the optimal value of g which minimizes the difference. Such an optimal g represents a pessimistic estimate on 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 4.3. In terms of the upper bound of the difference in approximation degrees, the best stepping factor g^* is

$$g^* = \log_{\tau_{GD}} \left\{ 2 \left[1 - \frac{1}{T} \right] \right\} \tag{4.20}$$

Proof. Let $Q(g,T)$ be the difference in approximation degrees defined in Eq. (4.14). Taking partial directives of $Q(g,T)$ with respect to g, we have

$$\frac{\partial Q(g,T)}{\partial g} = \frac{\partial F(g,T)}{\partial g} \tag{4.21}$$

$$= 2\alpha_0 - \frac{\alpha_0 \tau_{GD}{}^{g}}{\tau_{GD}{}^{g} - 1 + \frac{1}{T}}$$

The lemma is proved by setting Eq. (4.21) 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 Eq. (4.14). By using the best stepping factor, the following theorem shows that the minimum difference in approximation degrees is very small.

Theorem 4.2. The minimum difference in approximation degrees between $\alpha_{OPT_A^*}$ and $\alpha_{sTCGD}(g^*)$ (the approximation degree obtained by *s*TCGD using g^*) is bounded by a constant, *i.e.*,

$$\alpha_{sTCGD}(g^*) - \alpha_{OPT_A^*} < 2\alpha_0 \log_{\tau_{GD}} 2 + G(T) \tag{4.22}$$

where $G(T)$ is defined in Theorem 4.1.
Proof. The theorem is proved by substituting g^* in Eq. (4.20) into Eq. (4.14), which results in

$$\alpha_{sTCGD}(g^*) - \alpha_{OPT_A^*}$$

$$\le \alpha_0 \log_{\tau_{GD}} \left[1 - \frac{1}{T} \right] + 2\alpha_0 \log_{\tau_{GD}} 2 + G(T) \tag{4.23}$$

The first term on the right-hand side is negative since $1 - 1/T$ 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. □

Theorem 4.2 shows that *s*TCGD 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 *s*TCGD and not in that for TCA*. Also note that this best bound is logarithmic with respect to the time constraint T (Eq. (4.19) and Eq. (4.22)). The larger is the time constraint T, the looser will be the best bound.

*s*TCGD has the **warm start** problem (so is *s*TCA*) at the beginning of a new GDFS because execution time must be spent in order to achieve the degree of approximation in the last GDFS. This problem is more serious when the stepping factor is small. However, choosing large stepping factors may not allow the last search in the *s*TCGD to be completed or may result in a large part of the remaining time not used for improving the degree of approximation. A succinct choice of the stepping factor is, therefore, essential.

Figure 4.2 shows the performance profiles of *s*TCGD for a TS problem instance of size 20, where $\alpha_0 = 0.42$ and $\tau_{GD} = 69154$. According to Lemma 4.3, the best stepping factor $g^* = 0.062$ for T=69154. The performance profile corresponding to $g*$ is generally better than others, since it achieves better degrees of approximation in a wider range of time constraints. Note that the axis of time constraints is logarithmically scaled such that the rightmost region in Figure 4.2 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 for small time constraints.

One interesting point in Figure 4.2 is the trend of the performance profiles with respect to the stepping factors. The performance profiles improve as the stepping factors are decreased. This phenomenon is illustrated by curves A ($g = 0.33$), B ($g = 0.25$), C ($g = 0.2$), D ($g = 0.1$), and E

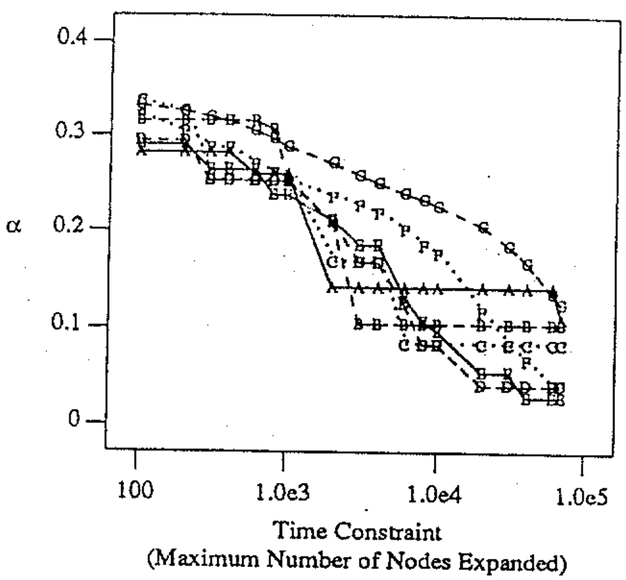

Figure 4.2. Performance profiles of sTCGD for a TS problem instance of size 20. (Symbols A, B, C, D, E, F, and G indicate g=0.33, g=0.25, g=0.20, g=0.10, g=0.062, g=0.02, and g=0.005, respectively.)

($g=0.062$). As the stepping factors decrease further, the performance profiles worsen. This is illustrated by curves F ($g=0.02$) and G ($g=0.005$).

With g=0.062, the performance of sTCGD is shown in Figures 4.1. According to Theorem 4.2, the upper bound of the difference in approximation degrees is 0.052. Figure 4.1 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 differences between the parametric and actual profiles are not included in our analysis. Figure 4.1 shows that the differences in approximation degrees between the actual and performance profiles for g^* is generally independent of the time constraint, and most of them are well bounded by the upper bound, *i.e.*, 0.052.

5. PREDICTIVE TCGD

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

$$pTCGD = \{ sTCGD(0, T_p), S(\alpha_{pred}, T-T_p) \} \qquad (5.1)$$

where $T_p \le T$.

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

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

(2) Regression phase. Using Eq. (3.5), 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 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 $\hat{\alpha} = c \times \alpha_{pred}$, where c is a positive real number around 1. If $\hat{\alpha} > \alpha_{sTCGD}$, then stop the search, and the final degree of approximation is α_{sTCGD}.

(4) Solution phase. Execute $S(\hat{\alpha}, T-T_p)$ 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. (2.2). The final degree of approximation achieved is the minimum of α_{sTCGD} and $\alpha_{run\text{-}time}(\hat{\alpha}, T-T_p)$. Note that during the Profiling Phase of pTCA* [7], nTCA* is used for collecting the partial *run-time* profile rather than sTCGD, which 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 ($0<s\le 1$) defines how much time should be spent in the profiling phase. The **corrective factor** c ($c>1, c=1, \text{or } c<1$) adjusts the predicted best degree of approximation 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 ($s_a = T_s/T$, where T_s is the maximum allowable time for profiling). A hybrid between relative and absolute can be chosen as follows.

$$s = \min \{ s_r, s_a \} \qquad (5.2)$$

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.

*p*TCGD suffers from the cold start problem, which 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. An overshoot causes the search problem to be incompletely solved so that a poor run-time approximation is reported. An undershoot causes the search problem to be solved with a loose degree of approximation, which results in a poor semi-optimal solution.

There are two solutions to alleviate the cold start problem. First, the stopping factor must be chosen large enough so 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, while a large corrective factor will likely cause an undershoot.

To see the effects of the stopping and corrective factors on the degrees of approximation, performance profiles for various factors are plotted. Figure 5.1 shows the performance profiles for various stopping factors at a corrective factor c=0.6. Figure 5.2 shows the performance profiles for various corrective factors at a stopping factor s=0.25. Smaller corrective factors result in better approximation (see Figure 5.1), although there is no significant difference in approximation degrees among the different stopping factors (see Figure 5.2) 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 *p*TCGD is plotted in Figures 4.1. Due to the cold-start problem and poor run-time approximation, *p*TCGD generally does not perform as well as *s*TCGD and the modified Lawler and Wood's algorithm based on GDFS.

6. CONCLUDING REMARKS

In this paper, we develop TCGD, a family of problem-independent, time-constrained, approximate GDFS algorithm. Three versions are studied: naive TCGD (*n*TCGD), static TCGD (*s*TCGD), and predictive TCGD (*p*TCGD). They are designed to solve combinatorial optimization problems using bounded memory space, with an objective of maximizing the degree of accuracy achievable within the given time constraint. Among these three versions, only *s*TCGD 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 *s*TCGD consistently performs better and has performance very close to that of OPT_{GD}. In contrast, *p*TCGD's performance is fair and is comparable to that of the modified Lawler and Wood's algorithm. *n*TCGD has the worst performance.

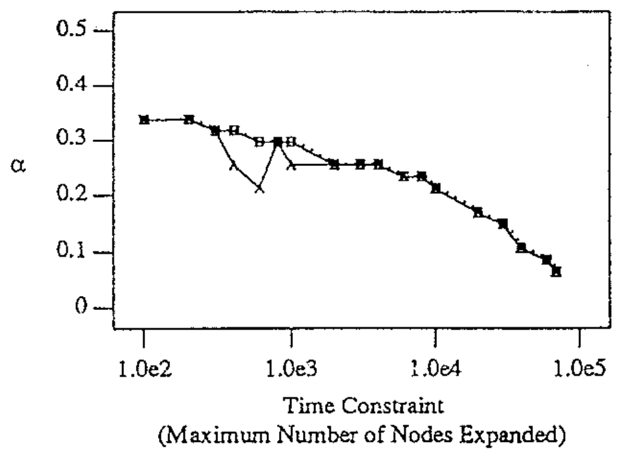

Figure 5.1. Performance profiles of *p*TCGD with s=0.25 for a TS problem instance of size 20. (Symbols A, B, C, and D indicate c=0.6, c=0.8, c=1.0, and c=1.2, respectively.)

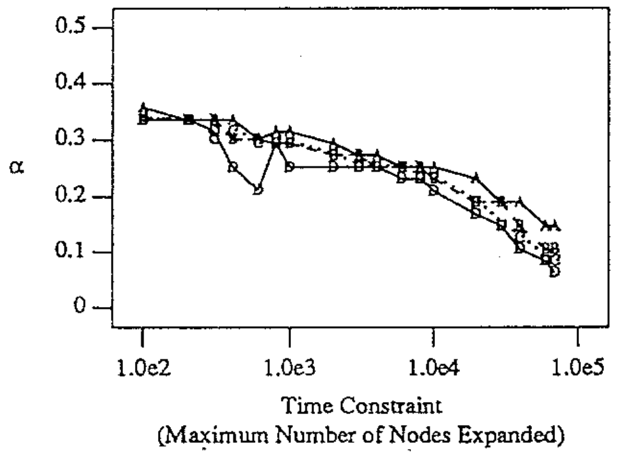

Figure 5.2. Performance profiles of *p*TCGD with c=0.6 for a TS problem instance of size 20. (Symbols A, B, C, and D indicate s=0.10, s=0.15, s=0.20, and s=0.25, respectively.)

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, *s*TCGD searches for less accurate solutions in the beginning; hence, time is consumed slowly in gradually approaching the maximum achievable degree of accuracy.

TCGD is also compared against TCA* in Figures 4.1. As expected, TCGD generally achieve worse degrees of approximation under a given time constraint, but achieves better degrees of approximation under a fixed CST cost.

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] L. G. Mitten, "Branch-and-Bound Methods: General Formulation and Properties," *Oper. Res.*, vol. 18, pp. 23-34, 1970.

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

[7] 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.