[24] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of sat problems. In *Proc. 10-th National Conf. on Artificial Intelligence (AAAI-92)*, pages 459–465, San Jose, CA, July 12-17 1992.

[25] C.H. Papadimitriou and P.C. Kanellakis. Flowshop scheduling with limited temporary storage. *J. ACM*, 27:533–549, 1980.

[26] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[27] J. Pearl. *Heuristics*. Addison-Wesley, Reading, MA, 1984.

[28] I. Pohl. Practical and theoretical considerations in heuristic search algorithms. In E.W. Elcock and D. Michie, editors, *Machine Intelligence 8*, pages 55–72. Wiley, New York, 1977.

[29] S.S Reddi and C.V. Ramamoorthy. On the flow-shop sequencing problem with no wait in process. *Operational Research Quarterly*, 23:323–331, 1972.

[30] A. Rényi. *Probability Theory*. North-Holland, Amsterdam, 1970.

[31] C.P. Williams and T. Hogg. Extending deep structure. In *Proc. 11-th National Conf. on Artificial Intelligence (AAAI-93)*, pages 152–157, Washington, DC, July 11-15 1993.

[32] W. Zhang. Truncated branch-and-bound: A case study on the asymmetric tsp. In *Working Note of AAAI 1993 Spring Symposium: AI and NP-Hard Problems*, pages 160–166, Stanford, CA, March 22-25 1993.

[33] W. Zhang and R.E. Korf. An average-case analysis of branch-and-bound with applications: Summary of results. In *Proc. 10-th National Conf on Artificial Intelligence (AAAI-92)*, pages 545–550, San Jose, CA, July 12-17 1992.

[34] W. Zhang and R.E. Korf. Depth-first vs. best-first search: New results. In *Proc. 11-th National Conf. on Artificial Intelligence (AAAI-93)*, pages 769–775, Washington, DC, July 11-15 1993.

[13] P.C. Kanellakis and C.H. Papadimitriou. Local search for the asymmetric traveling salesman problem. *Operations Research*, 28:1086–1099, 1980.

[14] R.M. Karp. A patching algorithm for the nonsymmetric traveling-salesman problem. *SIAM J. Comput.*, 8:561–573, 1979.

[15] R.M. Karp and J. Pearl. Searching for an optimal path in a tree with random costs. *Artificial Intelligence*, 99-117:21, 1983.

[16] W.H. Kohler and K. Steiglitz. Enumerative and iterative computational approaches. In Jr. (ed.) E.G. Coffman, editor, *Computer and Job-Shop Scheduling Theory*. John Wiley & Sons, New York, NY, 1976.

[17] V. Kumar. Search branch-and-bound. In S.C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 1468–1472. Wiley-Interscience, New York, 2 edition, 1992.

[18] T. Larrabee and Y. Tsuji. Evidence for a satisfiability threshold for random 3cnf formulas. In *Working Notes of AAAI 1993 Spring Symposium: AI and NP-Hard Problems*, pages 112–118, Stanford, CA, March 22-25 1993.

[19] E.L. Lawler, J.K. Lenstra, A.H.G Rinnooy Kan, and D.B. Shmoys. *The Traveling Salesman Problem*. John Wiley & Sons, Essex, 1985.

[20] E.L. Lawler and D.E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14:699–719, 1966.

[21] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[22] C.J.H. McDiarmid. Probabilistic analysis of tree searchdisorder. In G.R. Gummett and D.J.A. Welsh, editors, *Physical Systems*, pages 249–260. Oxford Science, 1990.

[23] C.J.H. McDiarmid and G.M.A. Provan. An expected-cost analysis of backtracking and non-backtracking algorithms. In *Proc. of the 12-th Intern. Joint Conf. on Artificial Intelligence (IJCAI-91)*, pages 172–177, Sydney, Australia, Auguest 1991.

[2] E. Balas and P. Toth. Branch and bound methods. In E.L. Lawler *et. al.*, editor, *Traveling Salesman Problem*, pages 361–401. John Wiley and Sons, Essex, Essex, 1985.

[3] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In *Proc. of the 12-th Intern. Joint Conf. on Artificial Intelligence, (IJCAI-91)*, pages 331–337, Sydney, Australia, Aug. 1991.

[4] J.M. Crawford and L.D. Auton. Experimental results on the crossover point in satisfiability problems. In *Proc. 11-th National Conf. on Artificial Intelligence, (AAAI-93)*, pages 21–27, Washington, DC, July 11-15 1993.

[5] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of a*. *JACM*, 32:505–536, 1985.

[6] A. Frieze, G. Galbiati, and F. Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Network*, 12:23–39, 1982.

[7] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, NY, 1979.

[8] J.G. Gaschnig. *Performance measurement and analysis of certain search algorithms*. PhD thesis, Computer Science Dept. Carnegie-Mellon University, Tech. Rep. CMU-CS-79-124, Pittsburgh, PA, 1979.

[9] B.A. Huberman and T. Hogg. Phase transitions in artificial intelligence systems. *Artificial Intelligence*, 33:155–171, 1987.

[10] T. Ibaraki. *Enumerative Approaches to Combinatorial Optimization - Part I*, volume 10 of *Annals of Operations Research*. Scientific, Basel, Switzerland, 1987.

[11] T. Ibaraki, S. Muro, T. Murakami, and T. Hasegawa. Using branch-and-bound algorithms to obtain suboptimal solutions. *Zeitchrift fur Operations Research*, 27:177–202, 1983.

[12] D.S. Johnson. Local optimization and the traveling salesman problem. In *Proc 17th Intern. Colloquium on Automata, Languages and Programming*, England, July 16-20 1990.

# 8 Conclusions

We have presented a new method, called $\epsilon$-transformation, that can be used by a search algorithm, such as branch-and-bound (BnB), to find approximate solutions to combinatorial problems. This method is designed to exploit the computational phase transitions of tree search problems. On a random tree, $\epsilon$-BnB, which is BnB using the $\epsilon$-transformation, runs in expected time that is at most cubic in the search depth, and finds a suboptimal goal node whose cost error is a small constant relative to the optimal goal cost. We also developed an iterative version of $\epsilon$-transformation that can be used to find both approximate and optimal solutions.

On random trees with large edge-cost ranges, large branching factors, and deep goal nodes, iterative $\epsilon$-DFBnB outperforms truncated DFBnB, finding better solutions sooner on average. On the asymmetric traveling salesman problem, $\epsilon$-DFBnB outperforms a local search method, and iterative $\epsilon$-DFBnB is superior to truncated DFBnB. Overall, we recommend that $\epsilon$-transformation be used in large search problems whose search trees have large edge-cost ranges and large branching factors.

To our knowledge, $\epsilon$-transformation is the first attempt to exploit phase transitions in order to solve combinatorial problems. Since phase transitions exist in many intelligent systems and combinatorial problems, we hope that the idea of $\epsilon$-transformation can be carried over to other problems and search methods.

# Acknowledgement

The authors are grateful to Colin McDiarmid and Judea Pearl for helpful discussions related to this research. Special thanks to Rich Korf for support, encouragement, discussions and comments.

# References

[1] S. Ashour. An experimental investigation and comparative evaluation of flow-shop scheduling techniques. *Operations Research*, 18:541–549, 1970.

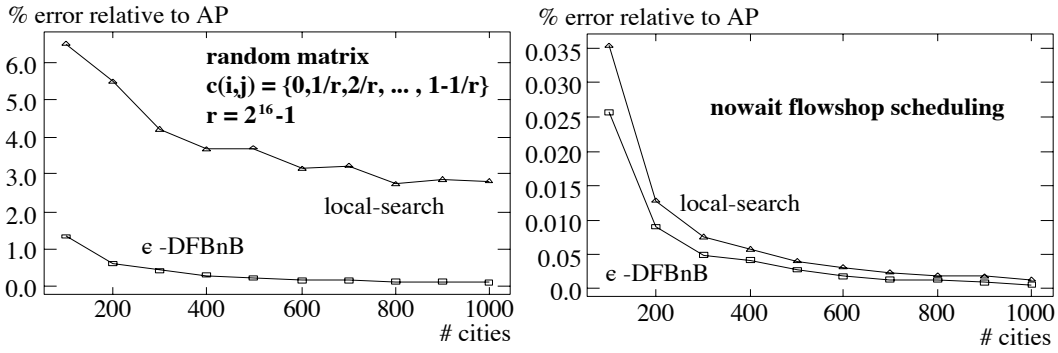



Figure 7: $\epsilon^*$-DFBnB vs. local search on the ATSP.

gent systems. Cheeseman *et al.* [3] empirically showed that phase transitions exist in many NP-hard combinatorial optimization problems. In particular, the phase transitions of constraint-satisfaction problems have attracted the attention of many researchers [4, 18, 24, 31].

In their seminal paper, Karp and Pearl [15] also proposed an algorithm that finds a suboptimal goal node of a tree most of the time, but may fail sometimes, and runs in expected time linear in the tree depth. McDiarmid and Provan [22, 23] extended Karp and Pearl's approximation algorithm to a general random tree. They further introduced dead-end nodes, which are leaf nodes located at a shallower depth than optimal goal nodes. They showed that backtracking is not necessary when there are no dead-end nodes. A serious problem with Karp and Pearl's algorithm is that it is incomplete, meaning that it is not guaranteed to find a goal node. Furthermore, the algorithm uses parameters that are based on the optimal goal cost, which is generally unknown, and hence is difficult to apply in practice.

It is well known in the operations research area that approximate solutions can be obtained by prematurely terminating DFBnB, taking the best solution found to that point as an approximation. This method was also referred to as truncated DFBnB, which we adopted in this paper. The earliest study of this method that we found was made by Ashour [1, 16]. Ibaraki *et al.* [10, 11] systematically studied approximation methods based on BnB, which they called *suboptimal BnB algorithms*. Their experimental results indicate that a good cost function is crucial to realize a successful suboptimal BnB algorithm.

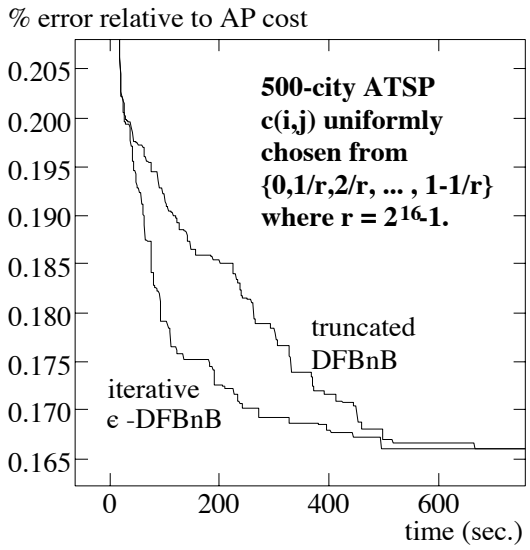



Figure 6: Iterative $\epsilon$-DFBnB vs. truncated DFBnB on 500-city random ATSPs.

DFBnB for the problem instances we considered, because we chose to use 5 initial tours for local search. Figure 7 shows our results on solution quality. The horizontal axes are the numbers of cities, ranging from 100 to 1000 cities in increments of 100. The vertical axes are the tour quality, expressed as the average tour cost error of these algorithms relative to the AP lower bounds. Each data point is averaged over 100 trials. The results show that $\epsilon^*$-DFBnB outperforms local search: it finds better solutions than local search on average even though local search uses more computation.

The results indicate that $\epsilon$-DFBnB and iterative $\epsilon$-DFBnB are effective on search trees with large branching factors and large edge-cost ranges.

# 7 Related Work

The existence of phase transitions in heuristic search was originally revealed by Karp and Pearl [15]. Their results have been extended by McDiarmid and Provan [22, 23], and Zhang and Korf [33, 34] to random trees with arbitrary branching factors and real-valued edge costs. Huberman and Hogg [9] systematically demonstrated that phase transitions are universal in large intelli-

defines a cost between each pair of cities, the ATSP is to find a minimum-cost tour that visits each city exactly once and returns to the starting city. The ATSP can be optimally solved by branch-and-bound (BnB), using the solution cost of the related assignment problem (AP) [26] as a monotonic node cost function. See [2] for a summary and detailed description of the method. Briefly, the state space of the ATSP under BnB is a tree without duplicate nodes.

In our implementation of $\epsilon$-DFBnB and iterative $\epsilon$DFBnB, we used the sampling method described in Section 4 to estimate the branching factor and edge cost distribution. The value of $\epsilon$ for the first iteration was the learned value of $\epsilon^*$, and the value of $\epsilon$ for each subsequent iteration was set to $fv_{max}/2$. From our experiments, $\epsilon$-DFBnB without actual-value pruning does worse than $\epsilon$-DFBnB with actual-value pruning. Hence, we only present the results of $\epsilon$-DFBnB with actual-value pruning.

We used many different cost matrices in our experiments. From our experiments, iterative $\epsilon$-DFBnB finds better solutions sooner than truncated DFBnB on average. Figure 6 shows one example on 500-city random ATSPs, where costs $c_{i,j}$ are uniformly chosen from $\{0, 1, 2, \cdots, 2^{16} - 1\}$. The results are averaged over 100 trials. The horizontal axis is the CPU time on a Sun4/sparc460 workstation, and the vertical axis is the average error of tour costs from these two algorithms relative to the AP lower bounds. This result is similar to the result on random trees in Figure 4.

We also compared $\epsilon^*$-DFBnB with a local search method [25] which was applied five different times for each problem instance in our experiments. The five initial tours were generated by the nearest-neighbor method [6], nearest insertion farthest insertion, greedy algorithms, and the patching algorithm [6, 14]. We used random cost matrices and matrices converted from no-wait flowshop scheduling for 4 machines, which is NP-hard [13]. No-wait flowshop scheduling involves determining a sequence for processing a set of jobs where each job must be handled by a set of machines in the same preset order. The objective is a sequence that minimizes a cost function, *i.e.*, total completion time cost which was used in our experiments. The no-wait constraint additionally requires the next machine to be available when a job is ready for it. The scheduling problem instances were generated by uniformly choosing the processing time of a job on a machine from $\{0, 1, 2, \cdots, 2^{16} - 1\}$. We then converted them into the ATSPs using the method in [29].

The average running time of local search is much longer than that of $\epsilon^*$-

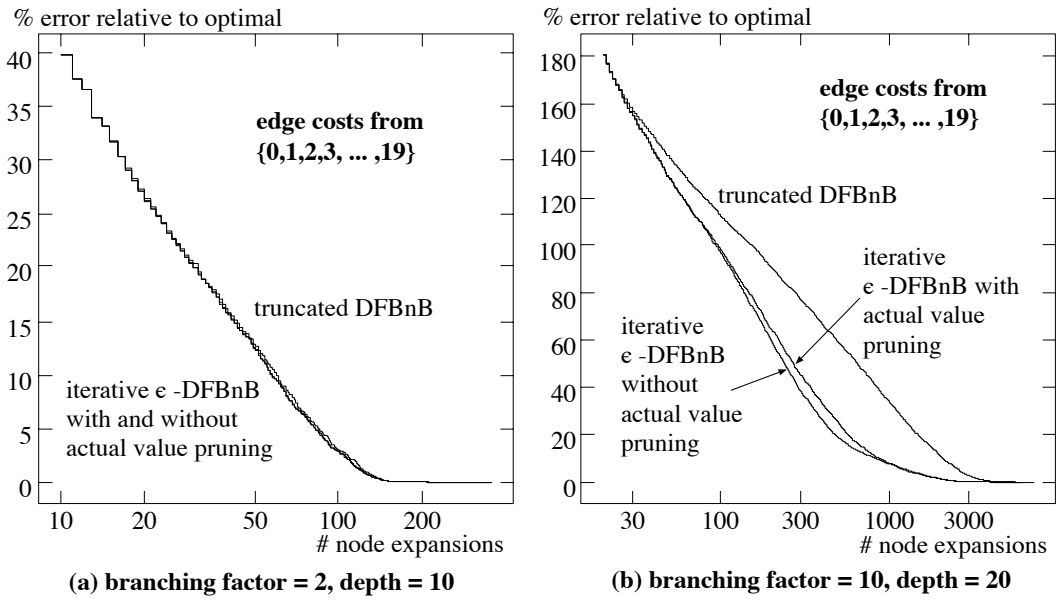



Figure 5: Iterative $\epsilon$-DFBnB vs. truncated DFBnB, small edge-cost range.

to 4(b)), iterative $\epsilon$-DFBnB further outperforms truncated DFBnB.

The edge-cost distribution also affects iterative $\epsilon$-DFBnB. Specifically, its relative advantage over truncated DFBnB decreases when the probability of zero-cost edges increases. Figure 5 shows one example on uniform random trees $T(b = 2, d = 10)$ and $T(b = 10, d = 20)$, with the edge costs uniformly chosen from $\{0, 1, 2, 3, \cdots, 19\}$. The horizontal and vertical axes are the same as those in Figure 4. The results are averaged over 1000 trials. Compared to Figure 4(a), Figure 5(a) shows that on random trees with a relatively large probability of zero-cost edges and a shallow depth, iterative $\epsilon$-DFBnB does not outperform truncated DFBnB. Although iterative $\epsilon$-DFBnB is still superior to truncated DFBnB when the branching factor and tree depth are increased (Figure 5(b)), its performance improvement over truncated DFBnB is less, as seen by comparing Figure 5(b) to Figure 4(b).

## 6.2 The Asymmetric Traveling Salesman Problem

The asymmetric traveling salesman problem (ATSP) [19] is an NP-hard [7] combinatorial problem. Given $n$ cities and an *asymmetric* matrix $(c_{i,j})$ that

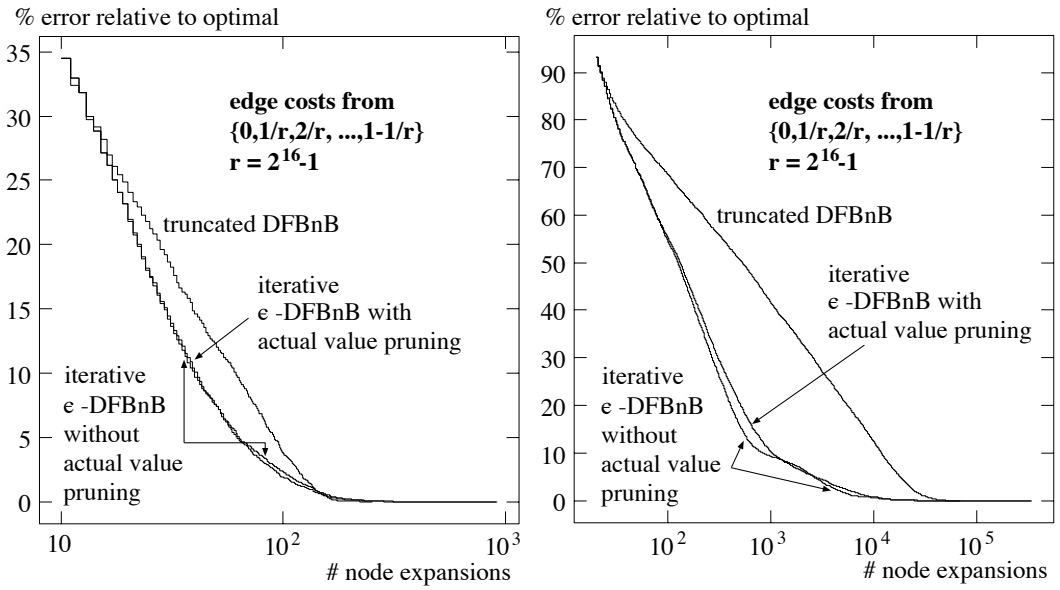



Figure 4: Iterative $\epsilon$-DFBnB vs. truncated DFBnB, large edge-cost range.

algorithm updated its current solution. We then recorded the average solution cost for a given number of node expansions as the performance measure, since expanding a node is the primary operation. In our experiments, the new $\epsilon$ value is set to $fv_{max}/2$.

Figure 4 shows our results on uniform random trees $T(b = 2, d = 10)$ and $T(b = 10, d = 20)$. The edge costs are uniformly chosen from $\{0, 1/r, 2/r, \cdots, 1 - 1/r\}$, where $r = 2^{16} - 1$. The results are averaged over 1000 trials. The horizontal axes, on a logarithmic scale, are the average node expansions, and the vertical axes are the average goal cost errors of both algorithms relative to the optimal goal costs. Figure 4 indicates that iterative $\epsilon$-DFBnB without actual-value pruning is slightly better than with actual-value pruning. Compared to truncated DFBnB, iterative $\epsilon$-DFBnB finds a better solution with the same average number of node expansions. For instance, on random trees $T(b = 10, d = 20)$ (Figure 4(b)), at 1000 node expansions, the relative error of solution costs from iterative $\epsilon$-DFBnB is 10.4%, while the relative error of solution costs from truncated DFBnB is 40.4%. Figure 4 also shows that when the branching factor and tree depth increases (from Figure 4(a)

$fv_{max}$. The most conservative way is to set $\epsilon = fv_{max}$. It can be easily shown that if edge costs are integers bounded by a constant, then iterative $\epsilon$-BnB that uses $\epsilon = fv_{max}$ in the next iteration expands asymptotically the same number of nodes as $\epsilon$-BnB that uses the exact value of $\epsilon$ for finding a required solution. In general, a small reduction in the value of $\epsilon$ may only cause a few new nodes to be explored in the subsequent iteration, which in turn may lead to a large number of iterations, and consequently a large node-regeneration overhead. Alternatively, we may decrease the value of $\epsilon$ by a larger amount, such as $\epsilon = fv_{max}/2$.

# 6 Experimental Study

The purpose of this section is to identify the conditions when $\epsilon$-transformation and iterative $\epsilon$-transformation are effective. To this end, we compare $\epsilon$-DFBnB and iterative $\epsilon$-DFBnB with other approximation algorithms.

Iterative $\epsilon$-DFBnB can be used in the same way as truncated DFBnB [11, 32] to find approximate and optimal solutions. Truncated DFBnB is a DFBnB that terminates prematurely when the total available computation has been exhausted. The best solution found up to that point can then be taken as an approximation. The main difference between these two algorithms is that the territory explored by iterative $\epsilon$-DFBnB is generally smaller than the territory explored by truncated DFBnB, although iterative $\epsilon$-DFBnB may re-expand a node many times.

Local search [12, 13, 21] is a well-known approximation method for many difficult combinatorial problems. Starting at an initial solution, such as one generated by a polynomial-time approximation algorithm, local search continuously improves the current solution by local perturbations, until no further improvement can be made. This process may be invoked many times with different initial solutions if more computation is available. A serious drawback of local search is that without enumerating all possible starting solutions, it cannot determine if the best solution found so far is optimal.

## 6.1 Random Trees

We ran both iterative $\epsilon$-DFBnB and truncated DFBnB on the same set of random trees, and recorded the total number of node expansions when either

may not be improved since a node $n$ may have a higher actual value than $u'$, but may have a lower face value than the face-value upper bound $u$. The optimal goal node of an $\epsilon$-tree $T_\epsilon(b,d)$ may be located underneath node $n$. Therefore, actual-value pruning reduces the opportunity to update $u$, consequently causing some nodes with higher face value to be expanded, which are not visited by $\epsilon$-BnB without actual-value pruning. Overall, $\epsilon$-BnB with and without actual-value pruning explore different parts of the search tree. Whether or not actual-value pruning increases or decreases the running time depends upon the relative pruning power of these two different pruning methods for a given problem. For the same reason, the solution quality they produce may differ as well. Our results on random trees show that $\epsilon^*$-DFBnB with actual-value pruning runs longer but finds better solutions than $\epsilon^*$-DFBnB without actual-value pruning.

# 5 Iterative $\epsilon$-Transformation

It may be the case that we need a better solution than can be guaranteed by $\epsilon$-transformation, *i.e.*, we may be asked to generate a solution whose cost is within a few percentage points of optimal. Although $\epsilon^*$-transformation is not guaranteed to find such suboptimal solutions, we can use an $\epsilon$ that is less than $\epsilon^*$ to find a better solution. Unfortunately, the previous analyses [15, 22, 23, 33, 34] do not provide a guideline on how to set this smaller $\epsilon$ in order to guarantee a given error bound.

We suggest an algorithmic approach to address this issue, which is called *iterative $\epsilon$-BnB*. Iterative $\epsilon$-BnB performs a sequence of BnB searches with a series of $\epsilon$-transformations, where the value of $\epsilon$ is reduced over successive iterations. The first iteration performs $\epsilon^*$-BnB. Within each iteration, BnB keeps track of the largest actual edge cost among all those that are set to zero. Call this value $fv_{max}$. At the end of an iteration, if the cost of the solution found is less than the required solution cost by comparing it to some lower bound, then the algorithm stops. Otherwise, the algorithm calculates a new value of $\epsilon$, and executes another $\epsilon$-BnB iteration. Iterative $\epsilon$-BnB can be executed until a satisfactory solution is found. DFBnB is preferable because it only requires space that is linear in the search depth.

There are many ways to update the value of $\epsilon$. To explore more of the trees in the next iteration, the new value of $\epsilon$ must be less than or equal to

$C_{\epsilon^*} = C + \delta k \leq C + \delta d = \delta d + \log\log d + o(\log\log d)$ as $d \to \infty$ (by (2)). This result and (2) give the relative error of $C_{\epsilon^*}$ as

$$\begin{aligned}
\lim_{d\to\infty} \frac{C_{\epsilon^*} - C^*}{C^*} &\leq \lim_{d\to\infty} \frac{\delta d + \log\log d + o(\log\log d) - (\alpha d + o(d))}{\alpha d + o(d)} \\
&= \lim_{d\to\infty} \frac{(\delta - \alpha)d}{\alpha d + o(d)} + \lim_{d\to\infty} \frac{\log\log d + o(\log\log d) - o(d)}{\alpha d + o(d)} \\
&= \delta/\alpha - 1
\end{aligned}$$

which is a constant, since $\delta \leq \epsilon^*$, and both $\epsilon^*$ and $\alpha$ are constants. □

A useful feature of $\epsilon$-transformation is that a tradeoff can be made between the expected complexity of $\epsilon$-BnB and the solution quality. Solutions with higher costs can be produced with less computation by using a larger value of $\epsilon$. Solutions with lower costs can be produced with greater computation by using a smaller value of $\epsilon$.

# 4 Learning $\epsilon$ and Actual-Value Pruning

The value of $\epsilon$ is calculated based on the branching factor $b$ and the edge-cost distribution $F$. Unfortunately, for practical problems, $b$ and $F$ are generally not available. Nevertheless, they can be estimated by sampling the nodes in a search tree while searching for solutions. In other words, the value of $\epsilon$ can be learned on-line during search. Consider DFBnB as an example. If DFBnB examines the children of the current node in best-first order of their face vales (node ordering), and breaks ties in favor of a node with a less actual node cost, then the first leaf node reached is the same whether $\epsilon$-transformation is used or not. DFBnB can take all nodes generated in the process of reaching the first leaf node as samples, and use them to estimate $b$ and $F$. As the search proceeds, the estimates of $b$ and $F$ can be refined and used to update the value of $\epsilon$.

BnB using $\epsilon$-transformation, $\epsilon$-BnB, can also use *actual-value pruning*. This pruning prevents BnB from exploring an interior node if its actual value exceeds the actual value of the best goal node found up to that point, which is called the *actual-value upper bound* $u'$. The purpose is to prune an interior node that cannot lead to a goal node with an actual value less than $u'$. Intuitively, one might expect actual-value pruning to improve the efficiency of BnB, and not to affect the solution quality. However, the search efficiency

chosen such that $bp_\epsilon \geq 1$. To maximize the solution quality, we select the minimum $\epsilon$ that satisfies $bp_\epsilon \geq 1$. That is, we choose

$$\epsilon^* = \min\{\epsilon | bp_\epsilon \geq 1\}, \quad \text{where} \quad p_\epsilon = F(\epsilon). \tag{1}$$

When $\epsilon = \epsilon^*$, we refer to $\epsilon$-transformation as $\epsilon^*$-transformation.

# 3 Properties of $\epsilon$-Transformation

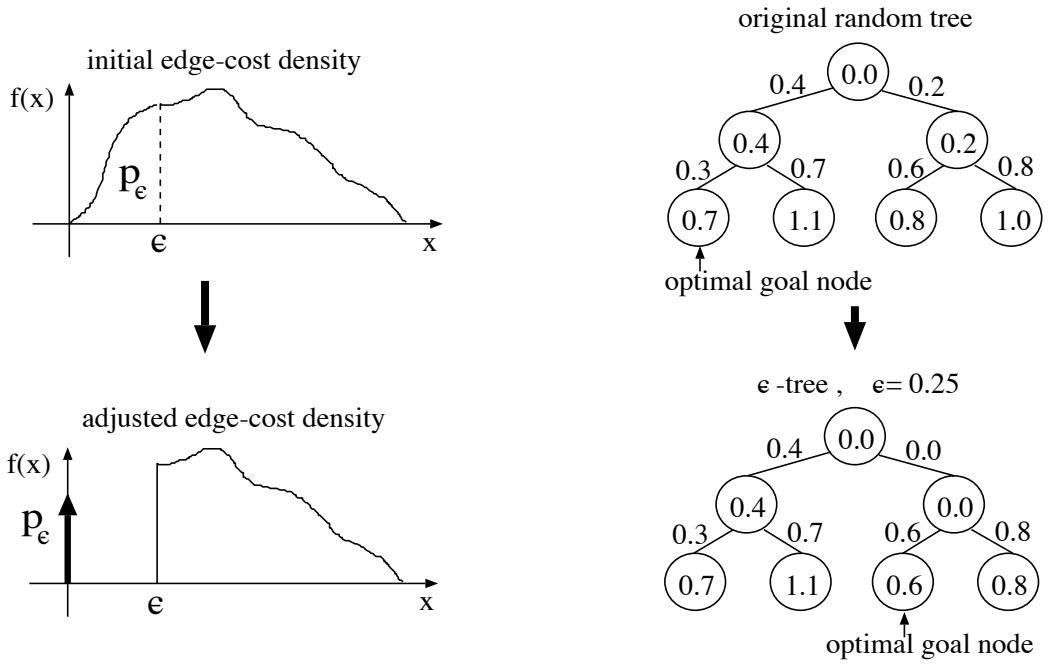
**Theorem 3.1** *On a random tree $T(b,d)$ with $bp_0 < 1$, as $d \to \infty$, $\epsilon^*$-BnB runs in expected time that is at most cubic in $d$, and finds a goal node whose error of solution cost relative to the optimal goal cost is almost surely a constant less than or equal to $(\delta/\alpha - 1)$, where $\alpha$ is a constant such that the optimal goal cost $C^*$ satisfies $C^*/d \to \alpha$ (Lemma 1.1), $\delta = E[x|x \in [0, \epsilon^*]]$, and $\epsilon^*$ is defined in Equation (1).*

*Proof*: $\epsilon^*$ is chosen by Equation (1) so that the expected number of same-cost children in $T_{\epsilon^*}(b,d)$ is one. Thus, by Lemma 1.1, the expected number of nodes expanded by $\epsilon^*$-BnB is at most cubic in $d$.

Let $C^*$ be the cost of an optimal goal node of $T(b,d)$. By Lemma 1.1, $C^*/d \to \alpha$ almost surely as $d \to \infty$, where $\alpha$ is a constant. Let $C$ be the face value of an optimal goal node of $T_{\epsilon^*}(b,d)$. By Lemma 1.1 and since $\epsilon^*$ satisfies (1), we have $C/(\log\log d) \to 1$ almost surely as $d \to \infty$. This means that $\alpha d$ and $\log\log d$ are the dominating terms in $C^*$ and $C$, respectively, as $d \to \infty$, *i.e.*,

$$C^* = \alpha d + o(d); \quad C = \log\log d + o(\log\log d) \quad \text{(almost surely as } d \to \infty\text{)} \tag{2}$$

Let $C_{\epsilon^*}$ be the *actual value* of the solution returned by $\epsilon^*$-BnB. $C_{\epsilon^*}$ consists of two parts. The first part is the sum of edge costs on the solution path which are greater than $\epsilon^*$. The face and actual values of these edges are the same and give rise to the face value $C$ of the optimal goal node in $T_{\epsilon^*}(b,d)$. The second part is the sum of the actual values of the edges on the solution path whose face values are set to zero. The actual values of these edges must be less than or equal to $\epsilon^*$. Let $\delta$ be the actual value of an edge that has a zero face value, and $k$ be the number of zero-cost edges on the solution path in $T_{\epsilon^*}(b,d)$, where $k \leq d$. Since edge costs are *i.i.d*, we then have




**(a) transforming an edge-cost density** **(b) transforming a random tree**

Figure 3: Adjusting an edge-cost density, and modifying a random tree.

probability that an edge has cost less than or equal to $\epsilon$ is $F(\epsilon) = \int_0^\epsilon f(t)dt$, which is also the probability $p_\epsilon$ that an edge of $T_\epsilon(b,d)$ has cost zero. Figure 3(a) illustrates an edge-cost density function and its adjusted density function for a given $\epsilon$. Figure 3(b) shows a $T(2,2)$ and its corresponding $T_\epsilon(2,2)$ with $\epsilon = 0.25$, where the numbers in the nodes and on the edges are node costs and edge costs, respectively. The optimal goal node of an $\epsilon$-tree is not necessarily the optimal goal node of its original random tree (see Figure 3(b) for an example). This is why $\epsilon$-transformation is not guaranteed to find an optimal goal node.

After the transformation, best-first search (BFS) or depth-first branch-and-bound (DFBnB) can be adopted to find an optimal goal node in the $\epsilon$-tree $T_\epsilon(b,d)$, and *the actual value* of this goal node. For simplicity, we call BnB, BFS, or DFBnB using $\epsilon$-transformation $\epsilon$-BnB, $\epsilon$-BFS, or $\epsilon$-DFBnB, respectively.

In order for $\epsilon$-BnB to run in polynomial average time, the value of $\epsilon$ is

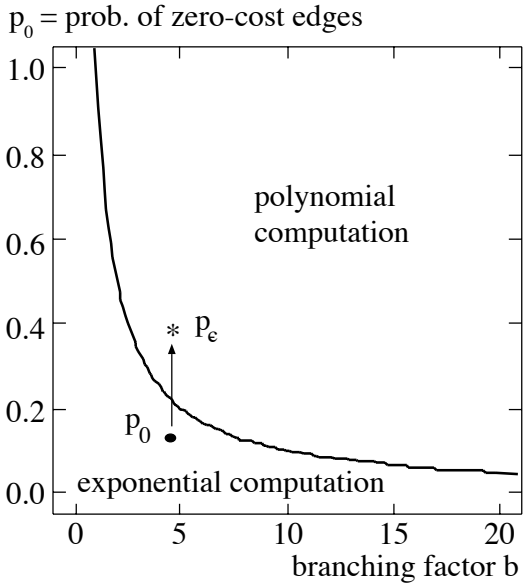



Figure 2: Transforming a difficult problem to an easy one.

tion to the problem of finding a suboptimal solution with polynomial average computation by *artificially* increasing $p_0$. This is illustrated by figure 2.

$p_0$ can be increased by setting some non-zero edge costs to zero. To reduce the amount of information lost on edge costs, and to improve the expected solution quality, we only set to zero those edge costs that are below a particular value $\epsilon$. This is why we call our method $\epsilon$-transformation. $\epsilon$ is set to the smallest value such that a suboptimal goal node can be found in polynomial average time.

**Definition 2.1** *For a given constant $\epsilon$, an* $\epsilon$-tree $T_\epsilon(b,d)$ *of a random tree $T(b,d)$ is the same as $T(b,d)$, except that those edge costs in $T(b,d)$ that are less than or equal to $\epsilon$ are set to zero in $T_\epsilon(b,d)$. The edge and node costs of $T(b,d)$ are referred to as* actual values*, and the edge and node costs of $T_\epsilon(b,d)$ are called* face values*.*

An $\epsilon$-tree is still a random tree, but with an adjusted edge-cost distribution, *i.e.*, with an increased probability of zero-cost edges. Let $f(x)$ be the density function and $F(x)$ be the distribution of edge costs. Then the

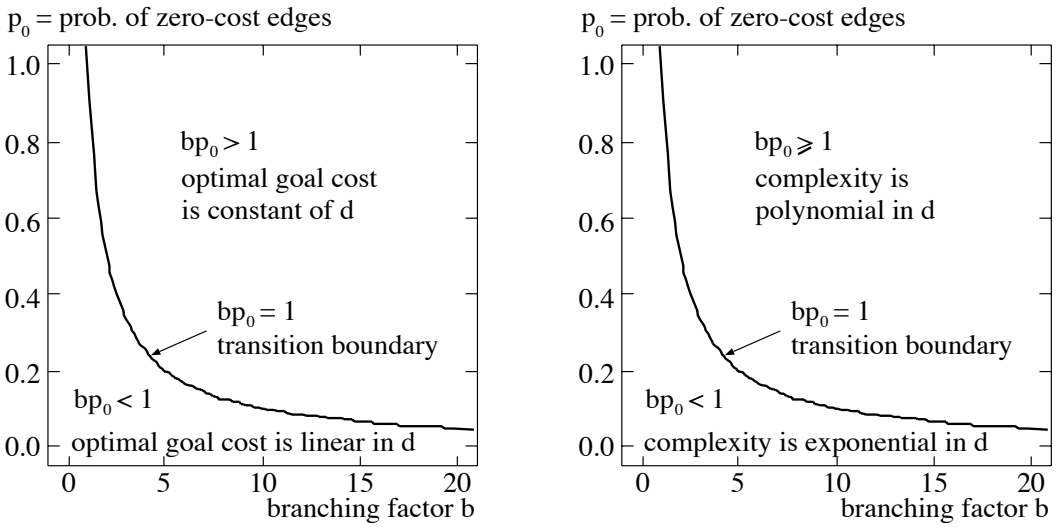



Figure 1: Phase transitions of tree search problems.

iterative version of $\epsilon$-transformation for finding both suboptimal and optimal goal nodes (Section 5). We identify the conditions when both methods are effective on random trees and on the asymmetric traveling salesman problem [19] (Section 6). Related work is discussed in Section 7, and our conclusions appear in Section 8.

# 2 $\epsilon$-Transformation

$\epsilon$-transformation is based on the following very simple observation of Figure 1(b). For a random tree $T(b, d)$, if we can increase the expected number of same-cost children of a node so that $bp_0 \geq 1$, then the expected complexity of finding an optimal goal node becomes polynomial in $d$. This can be accomplished by raising the probability $p_0$ of zero-cost edges, since the branching factor $b$ is usually fixed by the structure of state space. However, increasing $p_0$ means obtaining a better node-cost function, which requires more information about the problem, and is generally impractical. If we are willing to sacrifice the quality of solutions, however, we are able to transform the problem of finding an optimal solution with exponential average computa-

is less than $u$, or prunes the node if its cost is greater than or equal to $u$. Whenever a new leaf node is found whose cost is less than $u$, $u$ is revised to the cost of this new node.

It turns out that the cost of an optimal goal node of $T(b,d)$, and the expected complexity of BFS and DFBnB on $T(b,d)$ experience phase transitions. The order parameter that determines these transitions is the expected number of children of a node whose cost is the same as their parent, which are called *same-cost children*. If $p_0$ is the probability that an edge has cost zero, then $bp_0$ is the expected number of same-cost children of a node. When $bp_0$ increases from less than one to greater than one, the expected cost of the optimal goal node of $T(b,d)$ changes from a *linear function* of $d$ to a *constant*, and the expected complexity of BFS and DFBnB decreases from *exponential* in $d$ to at most *cubic* in $d$. Specifically, these phase transitions are summarized by the following lemma, and illustrated by Figure 1.

**Lemma 1.1** [22, 23, 34] *Let $C^*$ be the optimal goal cost of a random tree $T(b,d)$ with $b > 1$, and $N_B$ and $N_D$ be the expected numbers of nodes expanded by BFS and DFBnB on $T(b,d)$ respectively. As $d \to \infty$, (1) when $bp_0 < 1$, $C^*/d \to \alpha$ almost surely[1], where $\alpha$ is a constant, and $N_B = N_D = \theta(\beta^d)$, for some constant $\beta$; (2) when $bp_0 = 1$, $C^*/(\log\log d) \to 1$ almost surely, $N_B = \theta(d^2)$, and $N_D = O(d^3)$; and (3) when $bp_0 > 1$, $C^*$ is almost surely bounded, $N_B = \theta(d)$, and $N_D = O(d^2)$.* □

Many tree search problems that we encounter in practice, such as planning and scheduling, usually require computation exponential in the search depth, even in the average case. In other words, they are located in the exponential region of Figure 1(b). However, we usually do not need optimal solutions, but rather ones that have a satisfactory quality and can be found quickly. Therefore, the development of approximation algorithms is particularly important.

In this paper, we develop a new method, called $\epsilon$-transformation, that can be used by a search algorithm, such as BnB, to find suboptimal solutions quickly (Section 2). This method makes use of the phase transitions in Figure 1. We analyze its performance on a random tree $T(b,d)$ (Section 3), and consider an improvement and a variation (Section 4). We also present an

[1] A sequence $X_n$ of random variables is said to converge *almost surely* (with probability one) to $X$ if $P(\lim_{n\to\infty} X_n = X) = 1$ [30].

# 1 Introduction

It has been observed that phase transitions exist in many intelligent systems [9] and combinatorial optimization problems [3, 4, 15, 18, 22, 23, 24, 31, 33, 34]. A *phase transition* is a dramatic change to some problem property as some *order parameter* changes across a critical point. The simplest phase transition example is that water changes from a solid phase to a liquid phase when the order parameter, temperature, rises from below a critical point, the freezing point, to above that point.

The earliest evidence of computational phase transitions was the phase transition of a tree search problem [15], which has recently been studied in detail [22, 23, 33, 34]. The problem is to find an optimal goal node of the following random tree.

**Definition 1.1** [22, 23, 34] *A random tree* T(b,d) *is a tree with depth* d*, and independent and identically distributed (i.i.d) random branching factors with mean* b*. Nonnegative edge costs are bounded i.i.d. random variables. The cost of a node is the sum of the edge costs along the path from the root to that node. An* optimal goal node *is a node of minimum cost at depth* d*.*

Compared to the conventional tree model that assumes independence of node costs [8, 27, 28], this random tree is more realistic because it naturally introduces dependences among node costs. The costs of two nodes are dependent on each other if they share common edges on their paths to the root. This model has been applied to analyze practical problems, such as the traveling salesman problem [33, 34].

Best-first search (BFS) and depth-first branch-and-bound (DFBnB) can be used to search these random trees. Both are special cases of branch-and-bound (BnB) [17, 20, 26], which is a general technique for problem solving. BFS maintains a partially expanded state space, and at each cycle expands a minimum-cost node among all those generated but not yet expanded, until an optimal goal node is chosen for expansion. BFS is optimal among all algorithms that are guaranteed to find an optimal goal node, and use the same cost function, up to tie-breaking [5]. Hence, the complexity of BFS is also the complexity of finding an optimal goal node.

DFBnB uses an upper bound $u$ on the cost of an optimal goal. Starting at the root node, it chooses a deepest node, and expands this node if its cost

# $\epsilon$-Transformation: Exploiting Phase Transitions to Solve Combinatorial Optimization Problems*

*Weixiong Zhang* and *Joseph C. Pemberton*
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024
Email: zhang@cs.ucla.edu, pemberto@cs.ucla.edu
Phone: (310)206-3643

January 24, 1994

**Abstract**

It has been shown that there exists a transition in the average-case complexity of tree search problems, from exponential to polynomial in the search depth. We develop a new method, called $\epsilon$-transformation, which makes use of this complexity transition, to find a suboptimal solution. With a random tree model, we show that the expected number of nodes expanded by branch-and-bound (BnB) using $\epsilon$-transformation is at most cubic in the search depth, and that the error of the solution cost found relative to the optimal solution cost is a small constant. We also present an iterative version of $\epsilon$-transformation that can be used to find both optimal and suboptimal goal nodes. Depth-first BnB (DFBnB) using iterative $\epsilon$-transformation significantly improves upon truncated DFBnB on random trees with large branching factors and deep goal nodes, finding better solutions sooner on average. Our experiments on the asymmetric traveling salesman problem show that DFBnB using $\epsilon$-transformation outperforms a well-known local search method, and DFBnB using iterative $\epsilon$-transformation is superior to truncated DFBnB.

*This research was supported by NSF Grant No. IRI-9119825, a grant from Rockwell International, a GTE graduate fellowship (1992-93), and a UCLA Chancellor's Dissertation Year Fellowship (1993-94).