“Go With the Winners” Algorithms

David Aldous *
Department of Statistics
University of California
Berkeley CA 94720

Umesh Vazirani †
Department of Computer Science
University of California
Berkeley CA 94720

1 Introduction

We can view certain randomized optimization algorithms as rules for randomly moving a particle around in a state space; each state might correspond to a distinct solution to the optimization problem, or more generally, the state space might express some other structure underlying the optimization algorithm. In this setting, a general paradigm for designing heuristics is to run several simulations of the algorithm simultaneously, and every so often classify the particles as “doing well” or “doing badly”, and move each particle that is “doing badly” to the position of one that is “doing well”. In this paper, we give a rigorous analysis of such a “go with the winners” scheme in the concrete setting of searching for a deep leaf in a tree. There are two relevant parameters of the tree: its depth d, and another parameter κ which is a measure of the imbalance of the tree. We prove that the running time of the “go with the winners” scheme (to achieve 99% probability of success) is bounded by a polynomial in d and κ. By contrast, the simple restart scheme: run several *independent* simulations and pick the deepest leaf encountered takes time exponential in κ and d in the worst-case. We also show that any algorithm that guarantees a constant probability of success must have worst-case running time at least κd.

Another way to interpret the result is as follows: suppose we are given a heuristic randomized optimization algorithm with only a small probability p of success. One way to boost the success probability to 99%, say, is to run the algorithm $\theta(1/p)$ times using independent coin-flips and select the best answer. Can one do better by introducing interactions between the different runs of the algorithm? Our results show that in the tree setting, we can boost the success probability by using as few as $\log 1/p$ runs of the optimization algorithm by introducing “go with the winners” interactions between the runs.

Our original motivation for studying this concrete tree setting was that it mimics the polynomial time behavior of simulated annealing (see section 3). The polynomial time behavior of simulated annealing is notoriously hard to analyze rigorously. Many early results such as [8] apply only to the situation where the running time is allowed to grow exponentially in the problem size. A notable exception is an analysis of the case when the function to be minimized has a special kind of fractal-like structure, where [10] proved that simulated annealing runs in polynomial time. There are now several results (see for example [6]) that analyze the Metropolis algorithm, which is the special case of running simulated annealing at a fixed temperature. These results use sophisticated methods to bound mixing rates of Markov chains. However, such techniques do not appear to be adequate to analyze the behavior of general simulated annealing which corresponds to running a sequence of Markov chains, one for each temperature on the annealing schedule. The tree model was designed to capture precisely the effect of lowering the temperature - walking towards the leaves corresponds to lowering the temperature (as explained in section 3). There has been extensive experimental study of variants of simulated annealing such as

*Research supported by N.S.F. Grants MCS 92-24857 and by the Miller Institute for Basic Research in Science
†Research supported by N.S.F. Grant CCR-9310214

random restarts (see for example [2], and [7]). Our results suggest that it might be fruitful to study the performance of a "go with the winners" implementation for simulated annealing.

"Go with the winners" schemes are obviously reminiscent of genetic algorithms [3] - they incorporate a "survival of the fittest" rule, but do not incorporate a "mating rule" for combining pairs of solutions to obtain a new one. Our results indicate that the fitness rule can be quite a powerful heuristic for designing search strategies in certain contexts. We should point out that our results hold even if the state space being explored is a layered directed graph instead of a tree. By contrast, a recent paper [1] on the quadratic dynamical systems gave evidence that the mating rule is not as powerful as had been assumed.

2 Finding deep vertices on a tree

In this section, we introduce a simple model for certain randomized optimization algorithms, which we shall view as a rule for moving a particle around a state space. In our model the state space is a rooted tree, and the particle traverses a path from the root to a leaf of the tree. For each vertex v in the tree, the randomized optimization algorithm assigns a probability distribution $p(v_1|v), p(v_2|v), \ldots, p(v_m|v)$ on its children $v_1, v_2, \ldots, v_m$; the particle chooses its path according to these probabilities. The objective of the algorithm is to pick a deep vertex in the tree: of depth at least d. By truncating the tree at depth d, the question "what is the chance the algorithm finds some vertex at depth $\geq d$?" becomes the question "what is the chance that the deepest vertex found by the algorithm is at the (unknown) maximal depth d?". For the applications that we have in mind, the number of vertices in the tree is exponential in d, and the probabilities $p(v_i|v)$ are not available explicitly. Instead we have access to a procedure that given v as input produces a child of v with the above probability distribution. Moreover, the names of the vertices are not unique, so if the procedure is called twice on the same input v it is not possible to tell whether the same child of v was ouput in the two cases.

In our model the randomized optimization algorithm can now be stated as follows:

Algorithm 0. Start at the root, repeatedly choose a child at random until reaching a leaf, then stop.

Write $a(i)$ for the chance that this algorithm reaches at least depth i. If we repeat K times, or equivalently if we follow this procedure with K particles simultaneously and independently, the chance of finding a vertex at maximal depth d is large iff $Ka(d)$ is large, so the condition for a maximal depth vertex to be found in a polynomial number of steps is simply

$$1/a(d) \text{ is polynomial in } d. \tag{1}$$

(A *step* is a single move of a single particle). Can we specify an interacting algorithm which requires only some weaker condition? Here is a natural candidate, which mimics the typical "go with the winners" schemes mentioned in section 1.

Algorithm 1. *"Go with the winners"* Repeat the following procedure, starting at stage 0 with B particles at the root.

> At stage i each of the B particles is at some vertex at depth i. If all the particles are at leaves, then stop. Otherwise some particles $j_1, \ldots, j_m$ are at non-leaves: spread the remaining $B - m$ particles evenly among the positions of these m particles. If $B - m$ is not a multiple of m then pick a random subset of the m positions each of which gets assigned an extra particle. Then let each of the B particles move from its current vertex to a child chosen at random.

We remind the reader that choosing a child of v "at random" means according to the given probabilities $p(\cdot|v)$; let us also declare that all different random choices in any algorithm are to be made independently. Before we discuss the performance of Algorithm 1, we must first define the parameter κ.

To define κ we need more notation. Recall that, in terms of Algorithm 0, $p(v)$ is the chance the particle visits vertex v, and $a(j) = \sum_{v \in V_j} p(v)$ is the chance the particle reaches depth j at least. Now fix v and consider Algorithm 0 with the particle started at v: write $p(w|v)$ for the chance the particle visits vertex w, and $a(j|v) = \sum_{w \in V_j} p(w|v)$ for the chance the particle reaches depth j at least. For $i < j$ define

$$\kappa_{i,j} = \frac{a(i)}{a^2(j)} \sum_{v \in V_i} p(v) a^2(j|v)$$

In other words, let W_i be the position of the Algorithm 1 particle at depth i, conditioned on getting to depth i: then

$$\kappa_{i,j} = \frac{E a^2(j|W_i)}{(E a(j|W_i))^2} \geq 1.$$

Finally define

$$\kappa = \max_{0 \leq i < j \leq d'} \kappa_{i,j}.$$

Informally, $\kappa_{i,j}$ measures the variability, amongst depth-i starting vertices, of the chance of getting to depth j. So κ is some measure of the "imbalance" of the tree. For example, if there is a single deep vertex at level d, then $\kappa_{i,d} = 1/q_i$, where q_i the the probability of being at the unique level i ancestor of the deep vertex conditioned on getting to level i. As another example, consider a random tree in which each vertex is a leaf with probability $\gamma < 1/2$ and otherwise has two children, condition on the tree reaching depth d at least, and truncate to depth d. Take the given $p(\cdot|v)$ to be uniform on the children of v. For a typical realization of such a tree, one can use classical results about Galton-Watson branching processes to show $\kappa = \Theta(1)$ while $1/a(d) = \Theta(c^d)$ for $c = 2(1-\gamma) > 1$. In this example, the standard depth-first search algorithm would work well. But now consider the tree in figure 1.

Figure 1 illustrates an example in which Algorithm 1 works well but where any simple "backtracking" single particle algorithm would work badly. And backtracking algorithms are less natural in the general randomized optimization algorithm context we are trying to mimic.

To analyze the performance of Algorithm 1, we have to make the technical assumption that for each i, $\frac{a(i)}{a(i+1)} \geq poly(d)$ for some polynomial $poly(d)$ to be specified. If this does not hold for the tree in question, then it is very easy to satisfy this condition by simply associating $poly(d)$ new leaves with each non-leaf node in the tree. A "Go with the winners" algorithm is easy to implement on the new tree, and the parameter κ for the new tree is simply $poly(d)$ times the value of κ for the old tree.

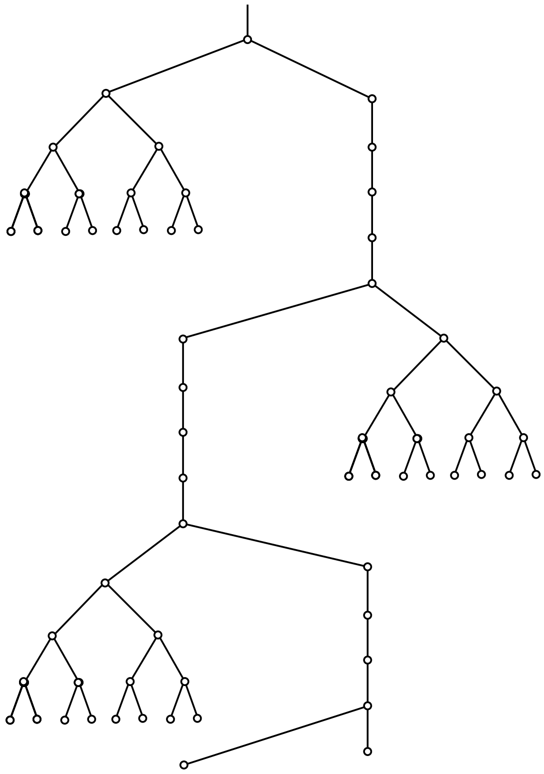

Figure 1: A bad tree for Algorithm 0

Theorem 1 *There is a polynomial $poly()$ such that if Algorithm 1 is run with $B = \kappa \times poly(d)$, then it fails to find the deepest leaf with probability at most $1/4$.*

Unfortunately Algorithm 1 is hard to analyze directly, and we defer its proof. To say why, let $p(v)$ be the chance that vertex v is on the path taken by Algorithm 0. And let V_j be the set of vertices at depth j. One might guess that in Algorithm 2, conditional on reaching depth j, the mean number of particles at vertices $v \in V_j$ would be proportional to $p(v)$. But this is false. One cannot give, inductively on j, an expression for the mean numbers of particles at vertices at depth j, because

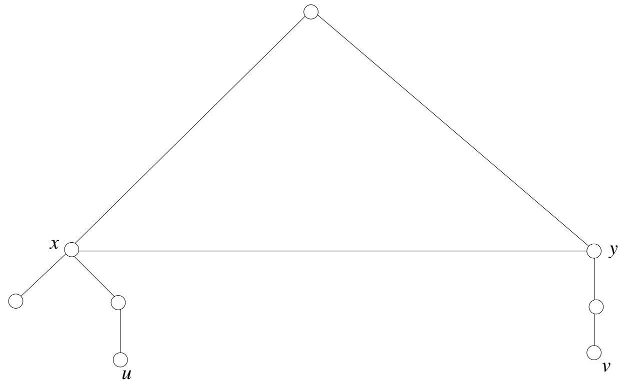

Figure 2: Conditioning in Algorithm 2

the dependence between positions of different particles comes into play. An example should make this clear. Consider the tree in figure 2, which consists of a large complete binary tree whose leftmost and rightmost leaves are extended as shown - the leftmost node x has two children, one is a leaf node and the other is a path down to node u. The rightmost node y has a path down to node v. Clearly $\frac{p(u)}{p(v)} = 1/2$. Now consider a "go with the winners" process with 2 particles. Since the complete binary tree is very large, we may as well assume that if both particles do not get stuck at the leaves of the complete binary tree, then at most one of them reaches x or y, and the other gets stuck at a leaf. But now the stuck particle is moved to the location of the other particle, and so both particles end up at x or at y with equal probability. If both particles are at x, with probability 3/4, both make it to u, since the only way the particles get stuck is if they both choose the leaf child of x. On the other hand conditioned on getting to y, both particles reach v with probability 1. So the mean number of particles reaching u is 3/4 times the mean number of particles reaching v, and these are not in the same ratio as $\frac{p(u)}{p(v)} = 1/2$.

We shall first analyze the following algorithm, Algorithm 2, which is conceptually similar to Algorithm 1, but mathematically more tractable. Our analysis of Algorithm 1 is based on the analysis of Algorithm 2. Algorithm 2 is defined in terms of the parameters θ_i for $1 \leq i \leq d-1$. Ideally we would like $\theta_i = \frac{a(i+1)}{a(i)}$, and our analysis will show that Algorithm 2 is efficient if the parameters θ_i are close to their ideal values. However, since we do not know the quantities $a(i)$ associated with the tree, Algorithm 2 is just a conceptual algorithm. For real $c \geq 0$ write $N(c)$ for a random variable with mean c and possible values $\{\lfloor c \rfloor, \lceil c \rceil\}$. That is, for non-integral c,

$$P(N(c) = \lfloor c \rfloor) = \lceil c \rceil - c, P(N(c) = \lceil c \rceil) = c - \lfloor c \rfloor. \quad (2)$$

Algorithm 2. Repeat the following procedure, starting at stage 0 with B particles at the root.

> At stage i there are a random number of particles all at level i. If all the particles are at leaves then stop. Otherwise, for each particle at a non-leaf, add at that particle's position a random number of particles, this random number having distribution $N(\theta_i^{-1} - 1)$. Finally move each particle from its current position to a child chosen at random.

The fact that different particles "reproduce" and "die" independently makes it possible to give moment estimates. The motivation for defining Algorithm 2 was to avoid the conditioning problems of Algorithm 1; we note that unlike Algorithm 1, Algorithm 2 does satisfy the property that conditional on reaching depth j, the mean number of particles at vertex $v \in V_j$ is proportional to $p(v)$.

Denote by S_i the number of particles at the start of stage i. Write

$$s_j = \prod_{i=1}^{j-1} \theta_i, j = 1, \ldots, d.$$

Lemma 2

$$E(S_i) = B\frac{a(i)}{s_i}, \ 0 \leq i \leq d.$$

$$\mathrm{var}(S_i) \leq \kappa B \frac{a^2(i)}{s_i^2} \sum_{j=0}^{i} s_j / a(j), \ 0 \leq i \leq d.$$

Note that if we choose $\theta_i = a(i+1)/a(i)$, then $E(S_d) = B$ and $\mathrm{var}(S_d) \leq \kappa B d$. So by Chebyshev's inequality the choice $B = 2\kappa d$ ensures that $S_d > 0$ with probability at least 1/2. As stated, Algorithm 2 cannot be implemented, since the θ_i's are not known quantities. However, we can iteratively use Algorithm 2 to obtain an algorithm that estimates the θ_i directly by sampling. This is what Algorithm 3 does.

Algorithm 3. Start at stage 0 with $B(d-1)$ particles at the root, divided into $d-1$ groups with B particles each. Move each particle to a randomly-chosen child of the root, ending stage 0. At the start of stage i ($1 \leq i \leq d-1$), groups 1 through $i-1$ have been discarded, and the remaining groups contain random numbers of particles, all at depth i. Count the proportion θ_i of group i particles which are at non-leaves. If $\theta_i = 0$ then output the vertex occupied by some group-i particle, and stop. Otherwise, discard all the particles in groups $i+1$ through $d-1$ which are at leaves. If no particles remain in group $i+1$, or if $i = d-1$, then choose some group-i particle at a non-leaf v, and output some child of v, and stop. Otherwise, discard the group-i particles. For each remaining particle v in groups $i+1$ through $d-1$, add at that particle's position a random number of additional particles, this random number having distribution $N(\theta_i^{-1} - 1)$, and include these additional particles in the same group as v. If any group now has more than $10B$ particles, then stop. Otherwise move each particle from its current vertex to a child chosen at random. End stage i.

The intuitive idea is that the number of particles in each group should remain approximately B. Because θ_i should be about $a(i+1)/a(i)$, and in the remaining groups the number of particles should change by a factor of approximately

$$\frac{a(i+1)}{a(i)} \times \theta_i^{-1} \approx 1$$

where the first term reflects particles at leaves being discarded, and the second term reflects the added particles.

The rule that the algorithm stops if any group acquires more than $10B$ particles clearly bounds the total number of steps (particle-moves) by

$$10B \sum_{i=0}^{d-1} (d-1-i) \; +1 \; \leq 5Bd^2. \qquad (3)$$

So the issue is to estimate the chance that the algorithm works, i.e. outputs some vertex at depth d, and this happens iff $\theta_{d-1} > 0$.

Our bound involves two parameters. The important parameter, κ, has already been discussed, and a less important one is

$$\beta \equiv \min_{0 \leq i < d'} \frac{a(i+1)}{a(i)}.$$

Theorem 3 *The chance that Algorithm 3 does not work is*

$$O\left(B^{-1}\kappa d^4 (1 + \frac{1}{\beta d})\right)$$

So under the reasonable assumption that $\beta = \Omega(1/d)$ it is enough to take $B = O(\kappa d^4)$ and so by (3) to use $O(\kappa d^6)$ steps. More crudely, if κ is polynomial in d then Algorithm 3 works in polynomial time.

3 Simulated annealing

The following discussion is of course oversimplified and inexact, but should make clear the motivation for our tree model.

Given a real-valued function f defined on a state space S, there is a natural way to define a tree such that leaves of the tree correspond to local minima of f. See figure 3. The essence of the correspondence is that each branch of the tree passing through height h corresponds to one connected component of $\{s : f(s) \leq h\}$. Although figure 3 pictures a one-dimensional S, all that is required of S is a notion of connectivity, and so the correspondence works for either continuous f on R^d with the topological notion of connectivity, or arbitrary f on a finite graph S with the graph-theoretic notion of connectivity.

If we run a Metropolis-type algorithm to try to simulate the distribution

$$\pi_T(s) = c_T \exp(-f(s)/T)$$

then what we actually get (in polynomial time) is π_T restricted to the connected component of $\{s : f(s) \leq h(T)\}$ containing the starting point, for some function $h(T)$. Identify this distribution with the corresponding point on the tree at height $h(T)$. Then we can view the progress of a polynomial-time simulated annealing algorithm, in which the

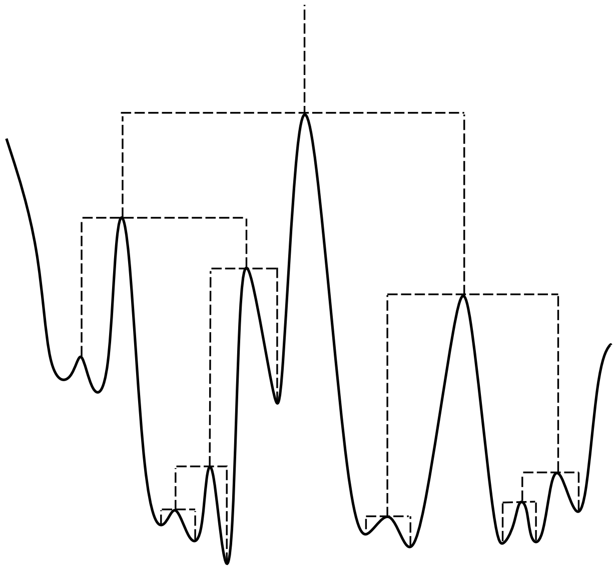

Figure 3: The tree corresponding to simulated annealing

temperature T is decreased with time, as a particle moving slowly down the tree, making irrevocable (non-uniform) random choices at branchpoints, ultimately ending at a leaf. (Of course theoretical results asserting logarithmic cooling schedules ultimately attain global minima have no relevance to polynomial-time behavior.) The branching probabilities in the description above depend on both size and depth of subtrees, and the fundamental requirement needed to make simulated annealing work well is that there be a strong association between size and depth of subtrees. This property has not been rigorously established for any nontrivial combinatorial optimization problem, and is unlikely to be established using current techniques for analyzing Markov chains. Instead, in our idealized tree model, we make a weaker assumption on the association between size and depth of subtrees, and show that it suffices to guarantee success of the "go with the winners" algorithm.

4 Proof of Lemma 2

The proofs of both Theorems 1 and 3 are based upon Lemma 2, which we prove in this section.

Recall the conditional variance formula

$$\mathrm{var}X = E\mathrm{var}(X|Y) + \mathrm{var}E(X|Y) \tag{4}$$

and the conditional covariance formula

$$\mathrm{cov}(X_1, X_2) = E\mathrm{cov}((X_1, X_2)|Y) +$$

$$\mathrm{cov}(E(X_1|Y), E(X_2|Y))$$

which hold for arbitrary square-integrable random variables (these formulas treat conditional expectations, etc, as random variables). We also need two simple facts about the distribution $N(c)$ defined at (2). First,

$$\mathrm{var}N(c) \le c \tag{5}$$

Second, if $N(c)$ balls are put independently into boxes $1, 2, \ldots$ with probabilities $q(1), q(2), \ldots$ then the numbers N_i in boxes i satisfy

$$\mathrm{cov}(N_i, N_j) \le 0, \quad j \ne i. \tag{6}$$

Proof of Lemma 2. Let X_v = number of particles reaching vertex v. So $S_n = \sum_{v \in V_n} X_v$. If w is a child of v and w is at depth n then

$$E(X_w|X_v) = p(w|v)\frac{s_{n-1}}{s_n}X_v \tag{7}$$

$$= \frac{p(w)/s_n}{p(v)/s_{n-1}}X_v.$$

So by induction along paths

$$EX_w = (p(w)/s_n)EX_0 = Bp(w)/s_n \tag{8}$$

and so

$$ES_n = Ba(n)/s_n.$$

For v, w as above let X_v^* be the number of particles at v after reproduction. Then

$$E(X_w|X_v^*) = p(w|v)X_v^*$$

$$\mathrm{var}(X_w|X_v^*) = p(w|v)(1 - p(w|v))X_v^*$$

and so by the conditional variance formula (4)

$$\mathrm{var}(X_w) = p(w|v)(1 - p(w|v))EX_v^*$$

$$+p^2(w|v)\mathrm{var}(X_v^*).$$

Now write

$$E(X_v^*|X_v) = (s_{n-1}/s_n)\ X_v$$

$$\mathrm{var}(X_v^*|X_v) \le (s_{n-1}/s_n)\ X_v \text{ by (5)}.$$

Applying the conditional variance formula (4) to the two expressions above gives

$$\mathrm{var}(X_v^*) \le EX_v^* + (s_{n-1}/s_n)^2 \mathrm{var}(X_v). \qquad (9)$$

Combining (4) and (9),

$$\mathrm{var}(X_w) \le p(w|v) s_{n-1}/s_n\ EX_v$$

$$+p^2(w|v)(s_{n-1}/s_n)^2 \mathrm{var}(X_v)$$

$$= EX_w + \left(\frac{p(w)/s_n}{p(v)/s_{n-1}}\right)^2 \mathrm{var}(X_v).$$

Then by induction, if $v(n)$ is a descendant of $v(0)$ through the line of descent $v(0), v(1), \ldots, v(n)$, then

$$\mathrm{var}(X_{v(n)}) \le \sum_{i=0}^{n} \left(\frac{p(v(n))/s_n}{p(v(i))/s_i}\right)^2\ EX_{v(i)}. \quad (10)$$

Now let $w(1), w(2)$ be vertices at depth n with last common ancestor v at depth $m < n$. Let $v(1), v(2)$ be the children of v which are ancestors of $w(1), w(2)$. Then

$$E(X_{w(1)}X_{w(2)}|X_{v(1)}, X_{v(2)})$$

$$= E(X_{w(1)}|X_{v(1)}) \times E(X_{w(2)}|X_{v(2)})$$

$$= \frac{p(w(1))/s_n}{p(v(1))/s_{m+1}}\ X_{v(1)} \times \frac{p(w(2))/s_n}{p(v(2))/s_{m+1}}\ X_{v(2)}$$

and this gives

$$\mathrm{cov}(X_{w(1)}, X_{w(2)})$$

$$= \frac{p(w(1))/s_n}{p(v(1))/s_{m+1}} \frac{p(w(2))/s_n}{p(v(2))/s_{m+1}} \mathrm{cov}(X_{v(1)}, X_{v(2)}).$$

Next consider how $X_{v(1)}$ and $X_{v(2)}$ relate to X_v.

$$\mathrm{cov}((X_{v(1)}, X_{v(2)})|X_v) \le 0 \text{ by (6)}$$

$$\mathrm{cov}(E(X_{v(1)}|X_v), E(X_{v(2)}|X_v))$$

$$= p(v(1)|v)p(v(2)|v)(s_m/s_{m+1})^2 \mathrm{var}(X_v) \text{ by (7)}$$

So by the conditional covariance formula (4)

$$\mathrm{cov}(X_{v(1)}, X_{v(2)})$$

$$\le p(v(1)|v)p(v(2)|v)(s_m/s_{m+1})^2 \mathrm{var}(X_v).$$

Substituting into (4)

$$\mathrm{cov}(X_{w(1)}, X_{w(2)}) \le \frac{s_m^2 p(w(1))p(w(2))}{s_n^2 p^2(v)} \mathrm{var}(X_v). (11)$$

We can now bound $\mathrm{var}(S_n)$.

$$\mathrm{var}(S_n) = \sum_{w(1)\in V_n} \sum_{w(2)\in V_n} \mathrm{cov}(X_{w(1)}, X_{w(2)})$$

Using (11) and the bound (10) for $\mathrm{var}(X_v)$, we get

$$\mathrm{var}(S_n) \le$$

$$\sum_{w(1)\in V_n} \sum_{w(2)\in V_n} \sum_{j=0}^{n} \sum_{u\in V_j} \frac{s_j^2 p(w(1))p(w(2))}{s_n^2 p^2(u)} EX_u$$

the final sum being restricted to those u which are common ancestors of $w(1)$ and $w(2)$

$$= B \sum_{w(1)\in V_n} \sum_{w(2)\in V_n} \sum_{j=0}^{n} \sum_{u\in V_j} \frac{s_j p(w(1))p(w(2))}{s_n^2 p(u)} \text{ using (8)}$$

$$= \frac{B}{s_n^2} \sum_{j=0}^{n} \sum_{u\in V_j} s_j a^2(n|u)p(u)$$

$$= \frac{B}{s_n^2} \sum_{j=0}^{n} s_j \kappa_{j,n} a^2(n)/a(j)$$

$$\le \kappa B \frac{a^2(n)}{s_n^2} \sum_{j=0}^{n} s_j/a(j).$$

5 Proof of Theorem 3

We first analyze Algorithm 3 without the condition "if any group has more than $10B$ particles, then stop". The key observation is that Algorithm 3 is equivalent to a sequential algorithm in which the particles of one group are moved down the tree and deleted before the particles in the next group are moved from the root. The behavior of the group-k particles depends on the previous groups' behaviors only via the quantities $\theta_1, \ldots, \theta_{k-1}$. Thus we start our analysis by conditioning on $\theta_1, \ldots, \theta_{k-1}$, and studying the conditional behavior of $B = S_0^{(k)}, S_1^{(k)}, \ldots, S_k^{(k)}, S_{k+1}^{(k)}$, where $S_i^{(k)}$ is the number of group-k particles at the start of stage i, interpreting $S_{k+1}^{(k)}$ as the number of particles at non-leaves at the start of stage k. Write

$$s_j = \prod_{i=1}^{j-1} \theta_i, j = 1, \ldots, k; \quad s_{k+1} = s_k.$$

Lemma 4 is a reformulation of Lemma 2 making the conditioning explicit. Lemmas 5 and 6 are proved below.

Lemma 4 *For* $1 \leq k \leq d' - 1$,

$$E(S_i^{(k)} | \theta_1, \ldots, \theta_{k-1}) = B\frac{a(i)}{s_i}, \ 0 \leq i \leq k+1.$$

$$\mathrm{var}(S_i^{(k)} | \theta_1, \ldots, \theta_{k-1}) \leq \kappa B \frac{a^2(i)}{s_i^2} \sum_{j=0}^{i} s_j / a(j),$$

$$0 \leq i \leq k+1.$$

Now write

$$b_d = \frac{1 + \frac{1}{2d}}{1 - \frac{1}{2d}}$$

and consider the event

$$A_j = \left\{ \frac{1}{b_d} \leq \frac{\theta_{j-1}}{a(j)} \leq b_d \right\}$$

Note that $b_d^{d-1} < e$ and hence for $j \leq d-1$

$$e^{-1} < \frac{1}{b_d^j} \leq \frac{s_j}{a(j)} \leq b_d^j < e \text{ on } A_1 \cap \ldots \cap A_j.$$

Recall that $\theta_k = S_{k+1}^{(k)} / S_k^{(k)}$. We shall estimate θ_k using Chebyshev's inequality and Lemma 2, and bound the variance terms inductively to get

Lemma 5 *For* $1 \leq k \leq d' - 1$,

$$P(A_{k+1}^c \text{ or } S_k^{(k)} \geq eB | \theta_1, \ldots, \theta_{k-1}, S_1^{(1)}, \ldots, S_{k-1}^{(k-1)})$$

$$\leq 8e\kappa B^{-1} d^3 (1 + \frac{1}{2\beta d})$$

on $A_1 \cap \ldots \cap A_k$.

Now summing over k gives

$$P(A_1, \ldots, A_d \text{ and } S^{(k)} < eB \text{ for } 1 \leq k \leq d-1)$$

$$\geq 1 - 8e\kappa B^{-1} d^4 (1 + \frac{1}{2\beta d}).$$

If A_d happens, the algorithm works, because $\theta_{d-1} > 0$. It remains to consider the chance that the algorithm stops because some group's size exceeds $10B$. Write $Q_{k,i}$ for the event that group k exceeds size $10B$ during stage i. We shall show that, for the algorithm continued without this stopping rule,

Lemma 6 *For* $1 \leq k \leq d-1$, $1 \leq i < k$,

$$P(S_k^{(k)} < eB | Q_{k,i}, A_1, \ldots, A_k) \leq 21\kappa d B^{-1}.$$

The conditional probability in Lemma 6 is larger than the corresponding joint probability, so summing over (k, i) gives

$$P(\text{ event (5), and } Q_{k,i} \text{ for some } (k,i)) \leq \frac{21}{2} \kappa d^3 B^{-1}.$$

Combining with (5) gives Theorem 3.

Proof of Lemma 5. Suppose events $A_1, \ldots, A_k$ happened. In order for A_{k+1} to happen it suffices that

$$\frac{1 - \frac{1}{2d}}{1 + \frac{1}{2d}} \leq \theta_k \frac{a(k)}{a(k+1)} = \frac{S_{k+1}^{(k)} a(k)}{S_k^{(k)} a(k+1)} \leq \frac{1 + \frac{1}{2d}}{1 - \frac{1}{2d}}.$$

Lemma 2 shows

$$ES_k^{(k)} = \frac{Ba(k)}{s_k}, \ ES_{k+1}^{(k)} = \frac{Ba(k+1)}{s_k}$$

so it suffices that

$$S_k^{(k)} \in (1 \pm \frac{1}{2d}) ES_k^{(k)} \text{ and} \tag{12}$$

$$S_{k+1}^{(k)} \in (1 \pm \frac{1}{2d}) ES_{k+1}^{(k)}. \tag{13}$$

Note also that when (13) occurs,

$$S_k^{(k)} \leq (1 + \frac{1}{2d}) \frac{Ba(k)}{s_k} \leq (1 + \frac{1}{2d}) B b_d^{k-1} < Be$$

and so to establish the Lemma it suffices to bound the probability that (13) does not happen. On $A_1 \cap$

$\ldots \cap A_k$ we have $s_j/a(j) \le e$ for $j \le k$ and so the variance bounds in Lemma 2 imply

$$\mathrm{var} S_k^{(k)} \le \kappa B \frac{a^2(k)}{s_k^2} ed$$

$$\mathrm{var} S_{k+1}^{(k)} \le \kappa B \frac{a^2(k+1)}{s_k^2} e \left(d + \frac{a(k)}{a(k+1)}\right)$$

$$\le \kappa B \frac{a^2(k+1)}{s_k^2} ed(1+1/\beta).$$

Applying Chebyshev's inequality separately to each event in (13), the chance the event does not happen is at most $\kappa B^{-1} ed \times (2d)^2$ or $\kappa B^{-1} e(d + 1/\beta) \times (2d)^2$, establishing the Lemma.

Proof of Lemma 6. We use the same kind of argument as in the proof of Lemma 5. Condition on $S_{i+1}^{(k)} = B^* \ge 10B$, for fixed B^* and i. Applying Lemma 2 to the group-k particles started with B^* particles at the beginning of stage $i+1$,

$$ES_k^{(k)} = B^* \frac{a(k)/a(i+1)}{s_k/s_{i+1}}.$$

$$\mathrm{var} S_k^{(k)} \le \kappa B^* \frac{a^2(k)}{s_k^2} \sum_{j=i+1}^{k} \frac{s_j/s_{i+1}}{a(j)/a(i+1)}.$$

If events $A_1, \ldots, A_k$ happened then

$$ES_k^{(k)} = B^* \prod_{j=i+2}^{k} \frac{\theta_{j-1}}{a(j)} \ge e^{-1} B^*$$

$$\mathrm{var} S_k^{(k)} \le \kappa B^* de^3.$$

Then Chebyshev's inequality gives the first inequality in

$$P(S_k^{(k)} < eB) \le \frac{\kappa B^* de^3}{(B^* e^{-1} - eB)^2} \le 21 \kappa d B^{-1}$$

where the second inequality uses $B^* \ge 10B$.

6 Sketch of Proof of Theorem 1

In this section, we outline the proof of Theorem 1. First consider running Algorithm 2 with parameters θ_i chosen so that the expected number of particles at level i, S_i, is $B(1 + 1/p_1(d))^i$ for some polynomial $p_1(d)$ to be chosen. By applying Chebyshev's inequality to the variance bound in Lemma 2, we can closely bound the probability that S_i deviates significantly from this expectation.

We will show that there are polynomials p_2, p_3, p_4, p_5 such that for $B = \kappa p_5(d)$:

- With probability at least $1 - 1/p_2(d)$, $|S_i - ES_i| \le B/p_3(d)$.
- With probability at least $1 - 1/p_2(d)$, for each i and each particle, the total number of descendants of this particle at level i is at most $B/p_4(d)$.

We now exclude from consideration the $2/p_2(d)$ fraction of the probability space in which the above two conditions are not satisfed. Our next observation is that it is possible to transform a run of Algorithm 2 into a run of Algorithm 1 in stages as follows:

> At stage i there are a random number C of particles at level i, which can be partitioned into D groups, each with $N(\theta_i^{-1})$ particles. If $C < B$ then abort. If $C \ge B$, then remove $C - B$ particles making sure that particles are removed from the groups with the largest number of particles first, and ties are broken randomly. Also, remove all descendants of particles that are removed.

Finally, to prove that Algorithm 1 reaches depth d with high probability, it suffices to show that the probability that we have to abort in the above transformation is very small. We carry out this argument for only those runs of Algorithm 2 that satisfy the two conditions above. First note that if the number of particles at each level was exactly equal to its expectation, and if the removal of particles from each level i caused exactly the expected number of descendants to be removed from levels $j > i$, then at the end of stage i, the number of particles remaining at level $j \le i$ is exaclty B and at level $j > i$ is $B(1 + 1/p_1(d))^{j-i}$. By applying Chernoff bounds, and an inductive argument to establish that the errors do not grow exponentially, but only polynomially, we can show that the number of particles remaining after stage i is with high probability close to the above numbers. Thus the process runs to completion with high probability.

7 Miscellaneous remarks

7.1 Approximate counting

There is a structural similarity between Theorem 3 and the topic of *approximate counting* [9]. To take the best-known example [5], consider how to estimate the volume of a convex set C in d dimensions, for large d, if we are told

$$B(1) \subset C \subset B(2)$$

where $B(r)$ is the ball of radius r. We can't just estimate the ratio $\mathrm{vol}(C)/\mathrm{vol}(B(2))$ by sampling at random from $B(2)$, because the ratio may be exponentially small (c.f. Algorithm 1 and (1)). But we can define subsets

$$C \subset C_1 \subset \ldots \subset C_m = B(2)$$

such that the ratios $\mathrm{vol}(C_i)/\mathrm{vol}(C_{i+1})$ are not small, and then estimate each ratio by sampling from C_{i+1}. Algorithm 3 has a similar "conditional sampling" flavor, and the parameter κ measures the non-uniformity of conditional probabilities.

7.2 Superprocesses

"Go with the winners" schemes are loosely related to a fashionable topic in theoretical probability. Consider K particles performing independent random walks on the d-dimensional lattice. Introduce an interaction by picking at each step one particle at random and moving it to the position of another randomly-chosen particle. Now the group of particles does not disperse as time increases but stays within $O(1)$ of its randomly-moving center of mass. With appropriate scaling, the $K \to \infty$ limit is a process of unit mass moving randomly in space, essentially the Fleming-Viot process [4], and this is one of a family of measure-valued diffusions studied under the name of *superprocesses*. There are however two important differences between our optimization algorithm setting and the theoretical probabilist's setting. First, we have an objective function that we use to choose which particle to move. Second, studying asymptotics of polynomially-many particles on exponentially-growing state spaces is intrinsically different from studying asymptotics on a *fixed* state space.

References

[1] S. Arora, Y. Rabani, and U. Vazirani. Quadratic dynamical systems. In *Proc. 34th Ann. Symp. on Foundations of Computer Science*, 1993.

[2] K. Boese, A. Kahng, and S. Maddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, page to appear, 1994.

[3] L. Davis, editor. Van Nostrand Reinhold, 1991.

[4] D. A. Dawson and K. J. Hochberg. Wandering random measures in the Fleming-Viot model. *Applied Probability*, 10:554–580, 1982.

[5] M. Dyer, A. Frieze, and R. Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *J. Assoc. Computing Machinery*, 38:1–17, 1991.

[6] M. Jerrum and G. Sorkin. Simulated annealing for graph bisection. In *Proc. 34th Ann. Symp. on Foundations of Computer Science*, 1993.

[7] D. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation part i, graph partitioning. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, 1990.

[8] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli. Convergence and finite-time behavior of simulated annealing. *Advances in Applied Probability*, 18:747–771, 1986.

[9] A. J. Sinclair. *Algorithms for Random Generation and Counting*. Birkhauser, 1993.

[10] G. Sorkin. Efficient simulated annealing on fractal energy landscapes. *Algorithmica*, 6:367–418, 1991.