Journal of Combinatorics
Volume 1, Numbers 3–4, 265–283, 2010

On randomizing two derandomized greedy algorithms*

Kevin P. Costello†, Asaf Shapira‡ and Prasad Tetali§

We consider the performance of two classic approximation algorithms which work by scanning the input and greedily constructing a solution. We investigate whether running these algorithms on a random permutation of the input can increase their performance ratio. We obtain the following results:

1. Johnson's approximation algorithm for MAX-SAT is one of the first approximation algorithms to be rigorously analyzed. It has been shown that the performance ratio of this algorithm is 2/3. We show that when executed on a random permutation of the variables, the performance ratio of this algorithm is improved to $2/3 + c$ for some $c > 0$. This resolves an open problem of Chen, Friesen and Zhang [3].

2. Motivated by the above improvement, we consider the performance of the greedy algorithm for MAX-CUT whose performance ratio is 1/2. Our hope was that running the greedy algorithm on a random permutation of the vertices would result in a $1/2 + c$ approximation algorithm. However, it turns out that in this case the performance of the algorithm remains 1/2. This resolves an open problem of Mathieu and Schudy [11].

AMS 2000 subject classifications: Primary 68W20; secondary 68W25, 05C80.
Keywords and phrases: Randomized algorithms, approximation algorithms, random graphs.

1. Introduction and statement of main results

The "greedy" approach is probably the earliest and most widely used paradigm in designing algorithms. Some of the most well studied exact algorithms

*An extended abstract of this paper is appearing in the Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA) 2011.

†Supported by NSF grant DMS-0902968.
‡Supported in part by NSF grant DMS-0901355.
§Supported in part by NSF grants DMS-0701043, CCR-0910584.

are based on this approach. This method has also turned out to be useful when designing approximation algorithms. See [14] for more details and several concrete examples. Our main goal in this paper is to consider variants of two classic approximation algorithms that were based on this approach. In both cases we analyze the performance ratio of an algorithm which executes the greedy algorithm on a random permutation of the input.

1.1. Johnson's algorithm for maximum satisfiability

The first problem we examine is the (weighted) MAX-SAT problem: Given an input set of clauses, together with a weight function $w(C)$ for each clause C, find an assignment so as to maximize the total weight of satisfied clauses. Since MAX-SAT is the archetypal NP-hard problem, approximation algorithms for this problem where studied as early as the early 70's. Probably the first algorithm was Johnson's algorithm [9], which was shown by its namesake to have an approximation ratio of at least 1/2 (this was later improved to 2/3). For many years this was the best approximation algorithm, until Yannakakis [16] obtained a 3/4 approximation algorithm, which was later simplified by Goemans and Williamson [7]. The best approximation algorithm is due to Avidor, Berkovitch and Zwick [1] which builds upon several previous algorithms (see [1] for details), all of which apply semi-definite programming.

While the above algorithms certainly outperform Johnson's algorithm, they all rely on sophisticated techniques such as Semi-definite programming, linear programming or Max-flow algorithms. On the other hand Johnson's algorithm can be stated as the following simple algorithm. Let $x_1, \ldots, x_n$ be an arbitrary ordering of the variables. Starting from x_1, assign the variables the value true/false using the following rule; when trying to assign x_i a value, we give it the value that maximizes the conditional expectation of the weight of satisfiable clauses, where the conditional expectation is taken over a uniform assignment to the variables $x_{i+1}, \ldots, x_n$ (while fixing the assignments to $x_1, \ldots, x_i$). If both values have equal expectation, one is chosen arbitrarily. One can easily see that this algorithm is just a simple derandomization (using the standard method of conditional expectations) of the algorithm that assigns $x_1, \ldots, x_n$ random values. Since in expectation, a random assignment satisfies at least half of the clauses (noting each clause has at least one literal), we immediately get that Johnson's algorithm has performance ratio of at least 1/2. This was the best result concerning the performance ratio of Johnson's algorithm, until Chen, Friesen and Zheng [3] obtained a tighter analysis showing that the performance ratio was actually

2/3 (see also [5] for a streamlined version of their analysis). More precisely they proved the following.

Theorem 1. [3] *The weight w_{sat} of clauses satisfied by Johnson's algorithm satisfies*

$$w_{sat} \geq \frac{w_{tot} + w_{opt}}{3} \geq \frac{2}{3} w_{opt} \,,$$

where w_{tot} is the total weight of all clauses and w_{opt} is the weight satisfied by an optimal assignment.

It was also observed in [3] that the 2/3 ratio in Theorem 1 is tight. For example, on the clause set

$$\{(x_1 \vee \neg x_2), (x_1 \vee \neg x_3), \neg x_1\},$$

with all clauses having weight 1 and all variables defaulting to true in case of ties, the algorithm begins by assigning x_1 to "true" (as both expected values are 2) and fails to satisfy the third clause. Note, however, that this was in some sense due to an unfortunate ordering of variables; if the algorithm had instead first considered x_2 or x_3, it would have set those variables "false". In either case x_1 would then have been set to false and all clauses would have been satisfied.

Because of this, [3] suggested the following *randomized* version of Johnson's Algorithm:

Algorithm 1. First choose an ordering of the variables uniformly at random, then run Johnson's Algorithm with the variables considered in this new order.

The following open problem was raised in [3]:

Question 1. [3] Does Algorithm 1 give a performance ratio better than 2/3?

Our first main result answers this question in the affirmative:

Theorem 2. *There is an absolute constant $c > 0$ such that for any weighted maximum satisfiability instance, Algorithm 1 satisfies in expectation at least a $(2/3 + c)$ fraction of the optimal assignment.*

We currently do not know the optimum value for c in Theorem 2. Poloczek and Schnitger ([12], see also discussion following Corollary 1 below) have constructed an example where Algorithm 1 with high probability only achieves a $2\sqrt{15} - 7 \approx 0.746$ performance ratio. In particular, this method cannot outperform the more sophisticated algorithms that apply semi-definite programming.

1.2. The greedy algorithm for MAX-CUT

A second problem for which we consider the performance of randomized greedy algorithms is MAX-CUT: Given an input graph G, we aim to find the partition (L, R) of the vertices of G which maximizes the number of edges crossing between L and R. Here the greedy algorithm (which dates back to Erdős [6]) is even simpler: vertices are considered in turn and each x_i is assigned to whichever of L and R creates more crossing edges between x_i and x_j, $j < i$. Ties are broken arbitrarily. This algorithm can be viewed as a derandomization of the uniform random cut, and as such has an approximation ratio of at least $1/2$. Again, this ratio is tight for the original algorithm: If the algorithm starts with a complete bipartite graph on $L_0 \times R_0$ and initially divides L_0 uniformly between the two sides, then no assignment of R_0 can cut more than $1/2$ the edges. Again, however, this example seems to be an artifact of a poor choice of vertex order. Mathieu and Schudy [11] proposed the following randomized variant of the greedy algorithm to avoid such examples:

Algorithm 2. Choose a random ordering of the vertices $\{x_1, \ldots, x_n\}$. Then perform the above greedy algorithm considering the vertices in this ordering.

In [11], the authors showed that repeated applications of such a randomized algorithm can be used to construct an approximation with an expected difference of ϵn^2 edges from the optimal cut size in time $n^2 + 2^{O(\epsilon^{-2})}$. In particular, for *dense* graphs it provides a very good approximation to the maximum cut size. Motivated by this, they asked:

Question 2. [11] Does the randomized greedy algorithm provide better than a $1/2-$approximation for general graphs?[1]

Given Theorem 2, which shows that taking a random permutation before running a similar greedy algorithm gives an improved performance ratio for MAX-SAT, it is tempting to assume that the answer to Question 2 would be positive. However, as we show in Theorem 3, the answer turns out to be negative: The randomized greedy partition does not provide better than a $(\frac{1}{2} + \epsilon)$ approximation to MAX-CUT in general for any $\epsilon > 0$.

Theorem 3. *For any $\epsilon > 0$ there are* **bipartite** *graphs $G = (V, E)$ on n vertices for which Algorithm 2 outputs a cut with at most $(\frac{1}{2}+\epsilon)|E(G)|$ edges*

[1] Showing *any* combinatorial algorithm giving a better than $1/2-$approximation was a long-standing open problem that was only recently solved by Kale and Seshadhri [10].

with probability $1 - \exp(-\Omega_\epsilon(\log^3 n))$. *In particular, repeating the algorithm a polynomial number of times is unlikely to give a* $(1/2+\epsilon)$*-approximation.*[2]

It is an interesting question at precisely which edge densities Algorithm 2 outperforms a simple random cut. The analysis of Mathieu and Schudy [11] establishes that Algorithm 2 gives a $1 - o(1)$ approximation for MAX-CUT on graphs with $\Omega(n^2)$ edges, and it is also not difficult to see that the algorithm provides a non-negligible improvement in the case of graphs with $O(n)$ edges (for example, because the algorithm gains half an edge over random on average every time it has to split an odd number of edges, and this adds up to a positive fraction of the total edges). Conversely, the details of the proof of Theorem 3 reveal that the performance ratio can be no better than $1/2 + o(1)$ on graphs with $O(n\sqrt{\log n})$ edges. We (hesitantly) make the following conjecture:

Conjecture 1. *For any* $0 < c < 1$ *there are bipartite graphs on* n^{2-c} *edges for which Algorithm* 2 *with high probability only cuts a* $\frac{1}{2} + o(1)$ *fraction of the edges.*

Finally, let us return to the analysis of Johnson's algorithm and derive the following upper bound on the performance of Algorithm 1 from our upper bound on the performance of Algorithm 2.

Corollary 1. *The performance ratio of Algorithm* 1 *is at most* $3/4$*, that is* $0 < c \leq 1/12$. *The same conclusion holds even when restricted to satisfiable formulas.*

There are ***satisfiable*** *formulas on which both Algorithm* 1 *and a uniform random assignment with high probability only satisfy* $3/4$ *of the clauses.*

Proof. Note that we can view MAX-CUT as a special case of MAX-SAT: Given a graph G, we construct a formula Ψ_G, where each edge $(x_i, x_j) \in E(G)$ is represented by two clauses, $(x_i \vee x_j)$ and $(\neg x_i \vee \neg x_j)$, each of weight 1 in Ψ_G. Observe that if we take a bipartition of G into L, R and set the variables in L to "true" and the variables in R to "false", then a cut edge satisfies two clauses and an uncut edge one. Now, it is not hard to see that Algorithm 1 specializes in this case to Algorithm 2. Hence, if we execute it on the formulas Ψ_G, which are derived from the graphs G of Theorem 3, the $\frac{1}{2} + o(1)$ upper bound on the fraction of cut edges due to Theorem 3 implies

[2]The problem raised by Mathieu and Schudy [11] considered executing Algorithm 2 some constant number of times, but we rule out the possibility of getting a $1/2+\epsilon$ performance ratio even when executing the algorithm a polynomial number of times.

that Algorithm 1 will satisfy only $\frac{3}{4}+o(1)$ of the clauses. Moreover, since the graphs G are bipartite, the formulas are completely satisfiable and so the performance ratio is bounded by 3/4 even if the formula is satisfiable. □

Remark 1. It is interesting to compare this result with the upper bound obtained by Poloczek and Schnitger in [12]. Their construction is a mixture of clauses of length 1 and length 2. Although Algorithm 1 provides a substantial improvement over a uniform random assignment in their construction, the uniform random assignment performs poorly enough that even this improvement is not enough to increase the approximation ratio to 3/4. Our construction is 2−uniform, so that the random assignment already satisfies 3/4 of the clauses. However, Algorithm 1 provides no further improvement in this case.

1.3. Paper overview

Theorem 2, regarding the performance of the modified Johnson algorithm, is proven in Section 2. The proof applies certain martingale and stopping time arguments along with LP-duality. We believe the combination of these techniques may be of independent interest and may be applicable to the analysis of other randomized approximation algorithms. Theorem 3, regarding the performance of the randomized MAX-CUT algorithm, is proven in Section 3. The main idea is to apply martingale arguments in order to analyze the performance of the algorithm on sparse random graphs.

2. The proof of Theorem 2

We may assume without loss of generality that the original optimal assignment had every variable set to "true" by swapping the roles of some x_i and $\neg x_i$ as necessary.

At any point in the algorithm we will denote by S those clauses which have already been satisfied by the assigned variables. We will refer to a clause as **negative** if it has not yet been satisfied and every variable within it occurs in the form $\neg x_i$, and denote the set of negative clauses by N. We will refer to an unsatisfied clause as **open** if it contains at least one unassigned variable, and **closed** otherwise. The measure of a clause, $\mu(C)$, will be defined as $w(C)2^{-|C|}$, where $|C|$ is the number of unassigned variables in C. In terms of this notation, the two expected values in Johnson's algorithm can be thought of as comparing the measure of the clauses containing "x_i" to that of the clauses containing "$\neg x_i$".

A key lemma in the analysis of [3] was the following:

Lemma 1. *At every step in Johnson's algorithm and for every i we have*

$$\Delta_i w(S) \geq 2\Delta_i \mu(N),$$

where Δ_i denotes the change in each quantity if variable x_i is the next to be assigned.

Even if we replace Johnson's Algorithm by Algorithm 1, there are situations where Lemma 1 will be tight. For example, consider the clause set

$$\{(x_1 \vee \neg x_2), (x_2 \vee \neg x_1)\}$$

with both clauses having weight 1 and both variables defaulting to "false" in case of ties. No matter which variable is chosen, one clause will be satisfied (so $\Delta w(S) = 1$) and one negative clause of length 1 will be created (so $\Delta\mu(N) = 1/2$). However, in this example we already know that a uniform random assignment, and hence Johnson's algorithm, satisfies on average at least a 3/4 fraction of the clauses.

The crux of our argument will be to show that, in the early stages of the algorithm, this dichotomy always holds: either the random assignment already significantly outperforms the 2/3 bound, or Lemma 1 holds on average with a ratio strictly larger than 2. Let δ_0, δ_1,and δ_2 be small positive constants to be determined later. Our conditions, which will be expressed in terms of the δ_i and will be discussed further following the lemma, can be thought of as describing those instances for which Johnson's algorithm could perform badly on some ordering of the variables.

Lemma 2. *Suppose that the initial set of clauses satisfied $w_{opt} \geq (1-\delta_0)w_{tot}$.*

Suppose furthermore that the following two conditions are simultaneously satisfied at some point in our randomized Johnson's algorithm:

- *The weight of S, the set of currently satisfied clauses, satisfies $w(S) \leq \delta_1 w_{tot}$.*
- *A random assignment which independently sets each unassigned variable to true or false with probability $\frac{1}{2}$ each satisfies in expectation weight at most $(\frac{2}{3} + \delta_2)w_{open}$ of the open clauses, where w_{open} denotes the total weight of open clauses.*

Then the next step in the algorithm has the property that

$$\mathbf{E}(\Delta w(S)) \geq r\mathbf{E}(\Delta\mu(N)),$$

where

$$r = r(\delta_0, \delta_1, \delta_2) := \frac{6(1 - \delta_0 - 2\delta_1)}{2 + 3\delta_0 + \delta_1 + 12\delta_2 - 12\delta_1\delta_2}.$$

Note that $r(0, 0, 0) = 3$, but that r drops below 2 as the δ_i increase (Lemma 1 still applies in this case even in the randomized version though). In some sense this is a reflection about how our initial "nearly satisfiable" condition gives us more control over the behavior of the algorithm at the beginning than further along.

The motivation for this lemma and its proof can be thought of as how knowledge that the approximation ratio of the original Johnson's Algorithm was near 2/3 can be exploited to provide a good amount of information about the (initial) structure of the set of clauses.

The use of δ_0 is motivated by the observation that Johnson's algorithm performs better on sets of long clauses than on sets of short ones. The algorithm it derandomizes (assigning all variables uniformly at random) satisfies a clause of length k with probability $1 - 2^{-k}$. If the expected fraction of clauses satisfied by the original algorithm is significantly larger than 2/3, we are already done since Johnson's algorithm also satisfies this fraction. So we assume this is not the case, which among other things implies that a positive fraction of the initial weight of clauses were singleton.

The use of δ_2 is motivated by how the original bound of $w_{sat} \leq 2(w_{sat} + w_{opt})/3$ in [3] is strongest when w_{sat} and w_{opt} are separated. In improving the overall approximation ratio, then, we can assume that the original set of clauses (including those singleton clauses we know exist) was nearly satisfiable. Note that this does nothing to prevent contradictory clauses from being produced later on in the algorithm, and we have less and less control over such created contradictions as time goes on. We introduce δ_1 as a cutoff to reflect this lack of control.

The proof of Lemma 2 involves viewing the lemma's constraints, along with an additional constraint coming from the definition of Johnson's Algorithm, as effectively defining a linear program[3]. The statement of the lemma now becomes that this program has a non-negative minimum, which can be directly verified.

2.1. The proof of Lemma 2

By rescaling, we may assume that $w_{tot} = 1$. We may also assume that $r \geq 2$, as otherwise the Lemma follows immediately from Lemma 1.

[3]We would like to thank Alex Samorodnitsky for suggesting this linear programming interpretation, which replaced a much clunkier original analysis.

Let A_1 denote the set of unassigned variables which, if selected by Johnson's algorithm, would be set to true in the current step, and A_2 denote those unassigned variables that would be set to false. Let $f(\alpha, \beta, \gamma, \delta)$ be the total weight of unsatisfied clauses with

- α instances of "x_i", with $x_i \in A_1$,
- β instances of "$\neg x_i$", with $x_i \in A_1$,
- γ instances of "x_i", with $x_i \in A_2$,
- δ instances of "$\neg x_i$", with $x_i \in A_2$.

In particular, $f(0,0,0,0)$ represents the total weight of closed clauses. We have the following five inequalities relating the f. First, the total weight of the clauses must equal w_{tot}, which we write as

$$-w(S) - \sum f(\alpha, \beta, \gamma, \delta) = -1. \tag{1}$$

Here and elsewhere all sums are assumed to be over all nonnegative values of variables in the summand. Next, we note that the definition of A_2 and that of Johnson's algorithm guarantees that for any variable $x_i \in A_2$ the measure of clauses containing $\neg x_i$ must be at least as large as the measure of clauses containing x_i. Adding up over all variables in A_2 gives

$$\sum \frac{\delta - \gamma}{2^{\alpha+\beta+\gamma+\delta}} f(\alpha, \beta, \gamma, \delta) \geq 0. \tag{2}$$

By the assumptions of the lemma we have

$$-w(S) \geq -\delta_1. \tag{3}$$

By assumption, we also know that $\mu(N)$ was initially at most $\frac{1}{2}\delta_0$ (since all clauses initially had length at least one), and by Lemma 1 we know the change in $\mu(N)$ so far is at most $\frac{1}{2}w(S)$. This gives an upper bound on $\mu(N)$, which we write as

$$\sum \frac{-1}{2^{\beta+\delta}} f(0, \beta, 0, \delta) \geq -\frac{\delta_0 + \delta_1}{2}. \tag{4}$$

Finally, since a random assignment fails for at least a $\frac{1}{3} - \delta_2$ fraction of the unsatisfied clauses, we have

$$\sum_{(\alpha,\beta,\gamma,\delta) \neq (0,0,0,0)} \frac{f(\alpha, \beta, \gamma, \delta)}{2^{\alpha+\beta+\gamma+\delta}} + \left(\frac{1}{3} - \delta_2\right)(w(S) + f(0,0,0,0)) \geq \frac{1}{3} - \delta_2. \tag{5}$$

We view equations (1)–(5) as a set of constraints defining a linear program on the $f(\alpha,\beta,\gamma,\delta)$ and $w(S)$. Our goal is to show that the objective function

$$(6)\quad \sum(\alpha+\delta)f(\alpha,\beta,\gamma,\delta)-r\left(\sum\frac{\beta-\delta}{2^{\beta+\delta}}f(0,\beta,0,\delta)+\sum\frac{1}{2^{\beta+\delta}}f(0,\beta,1,\delta)\right)$$

has a non-negative minimum value.

Consider the inequality formed by taking

$$c_1\cdot(1)+c_2\cdot(2)+c_3\cdot(3)+c_4\cdot(4)+c_5\cdot(5),$$

where

$$(c_1,c_2,c_3,c_4,c_5)=\left(r-1,2(r+1),\frac{1}{3}(3-r(1+6\delta_2)),2+r,2r\right).$$

(It can be checked that $c_3>0$.)

The right hand side of this inequality is

$$(r-1)(-1)-\frac{1}{3}(3-r(1+6\delta_2))\delta_1-(2+r)\frac{\delta_0+\delta_1}{2}+2r\left(\frac{1}{3}-\delta_2\right)=0\,,$$

(where r was chosen so that this was true). By LP-Duality, we are therefore done if we can show that for each $(\alpha,\beta,\gamma,\delta)$ the coefficient of $f(\alpha,\beta,\gamma,\delta)$ in

$$(7)\qquad (6)-(c_1\cdot(1)+c_2\cdot(2)+c_3\cdot(3)+c_4\cdot(4)+c_5\cdot(5))$$

is non-negative.

We will show this by the following case analysis:

Case 0: $\alpha+\beta+\gamma+\delta\le 5$. These follow from direct computation, which we omit here.

Case 1: $\alpha>0$ or $\gamma>1$ and $\alpha+\beta+\gamma+\delta>5$ In this case the contribution of $f(\alpha,\beta,\gamma,\delta)$ to $\Delta\mu(N)$ is 0, and (7) evaluates to

$$\begin{aligned}&\alpha+\delta+r-1-\frac{2(\delta+r+\delta r-\gamma(1+r))}{2^{\alpha+\beta+\gamma+\delta}}\\&\quad\ge\alpha+\delta+1-\frac{6}{2^{\alpha+\beta+\gamma+\delta}}-\frac{8\delta}{\alpha+\beta+\gamma+\delta}\\&\quad\ge\alpha+\delta+1-\frac{6}{64}-\frac{48}{64}\ge 0\end{aligned}$$

using the inequalities $2\le r\le 3$.

Case 2: $\alpha = 0$, $\gamma = 1$, and $\beta + \delta \geq 5$. Now (7) evaluates to

$$\begin{aligned} & r + \delta - 1 - \frac{\delta r + r + \delta - 1}{2^{\beta+\delta}} \\ & \geq \delta + 1 - \frac{4\delta + 2}{2^{\beta+\delta}} \\ & \geq \delta + 1 - \frac{4\delta + 2}{32} \geq 0. \end{aligned}$$

Case 3: $\alpha = 0, \gamma = 0$, and $\beta + \delta \geq 6$. Now (7) evaluates to

$$\begin{aligned} & r + \delta - 1 - \frac{r(\beta + \delta + 1) + 2\delta - 2}{2^{\beta+\delta}} \\ & \geq 1 + \delta - \frac{3(\beta + \delta + 1)}{2^{\beta+\delta}} - \frac{2\delta - 2}{2^{\beta+\delta}} \\ & \geq 1 + \delta - \frac{21}{64} - \frac{5}{32} \geq 0. \end{aligned}$$

2.2. Leveraging Lemma 2 into a better bound

By Theorem 1, in the case where $w_{opt} \leq w_{tot}(1 - \delta_0)$ we have

$$\begin{aligned} w_{sat} & \geq \frac{w_{tot} + w_{opt}}{3} \\ & \geq w_{opt} \frac{\frac{1}{1-\delta_0} + 1}{3} \geq w_{opt} \left(\frac{2}{3} + \frac{\delta_0}{3} \right). \end{aligned}$$

Now let us assume that $w_{opt} \geq (1 - \delta_0) w_{tot}$. Let S^0 be the set of satisfied clauses and N^0 be the set of negative clauses at the beginning of the first step where the hypotheses of Lemma 2 do not all apply, and let B be the event that $w(S^0) \geq \delta_1 w_{tot}$.

By Lemma 2, it follows that $w(S) - r(\delta_0, \delta_1, \delta_2)\mu(N)$ is a submartingale at each step prior to this point. Since this difference is clearly bounded for any fixed set of clauses, it follows from Doob's optional stopping theorem that the stopped difference is also a submartingale, that is to say that

$$\mathbf{E}(w(S^0)) \geq r(\delta_0, \delta_1, \delta_2)\mathbf{E}(\mu(N^0)). \tag{8}$$

We then have from Lemma 1 that after our modified analysis stops, the rate of increase of $w(S)$ remains at least twice that of $\mu(N)$, that is to

say

$$w(S) - w(S^0) \geq 2(\mu(N) - \mu(N^0)),$$

which would give us the usual 2/3 bound at the end of the algorithm.

However, we can do better in the case where B did not occur. Since Johnson always does at least as well as random, we know that, conditioning on $\neg B$, we have

$$w(S) - w(S^0) \geq \left(\frac{2}{3} + \delta_2\right) w_{open},$$

while

$$\mu(N) - \mu(N^0) \leq w(N) - w(N^0) \leq \left(\frac{1}{3} - \delta_2\right) w_{open}.$$

Comparing, we have at the end of the algorithm:

$$\begin{aligned} w(S) - w(S^0) &\geq 2(\mu(N) - \mu(N^0)) + 3\delta_2 w_{open} \\ &\geq 2(\mu(N) - \mu(N^0)) + 3\delta_2 \left(1 - \frac{3}{2}\delta_1 - \frac{1}{2}\delta_0\right) w_{tot}\,. \end{aligned}$$

Combining this with (8), we have (again at algorithm's end),

$$\begin{aligned} \mathbf{E}(w_{sat}) &= \mathbf{E}(w(S^0)) + \mathbf{E}(w_{sat} - w(S^0)) \\ &\geq 2\mathbf{E}(\mu(N^0) - \mu(N_{init})) + \left(1 - \frac{2}{r(\delta_0, \delta_1, \delta_2)}\right) \mathbf{E}(w(S^0)) \\ &\quad + 2\mathbf{E}(\mu(N) - \mu(N^0)) + 3\delta_2 \left(1 - \frac{3}{2}\delta_1 - \frac{1}{2}\delta_0\right) w_{tot}\mathbf{P}(\neg(B)) \\ &= 2\mathbf{E}(\mu(N) - \mu(N_{init})) + \left(1 - \frac{2}{r(\delta_0, \delta_1, \delta_2)}\right) (\delta_1 w_{tot}\mathbf{P}(B)) \\ &\quad + 3\delta_2 \left(1 - \frac{3}{2}\delta_1 - \frac{1}{2}\delta_0\right) w_{tot}\mathbf{P}(\neg(B)) \\ &\geq 2\mathbf{E}\left((w_{tot} - w_{sat}) - \frac{1}{2}(w_{tot} - w_{opt})\right) \\ &\quad + \min\{\left(1 - \frac{2}{r(\delta_0, \delta_1, \delta_2)}\right) \delta_1, 3\delta_2 \left(1 - \frac{3}{2}\delta_1 - \frac{1}{2}\delta_0\right)\} w_{tot}\,. \end{aligned}$$

Solving and using $w_{opt} \leq w_{tot}$ gives

$$\mathbf{E}(w_{sat}) \geq \left(\frac{2}{3} + \frac{1}{3}\min\left\{\left(1 - \frac{2}{r(\delta_0, \delta_1, \delta_2)}\right) \delta_1, 3\delta_2 \left(1 - \frac{3}{2}\delta_1 - \frac{1}{2}\delta_0\right)\right\}\right) w_{opt}\,.$$

Combining this with the case where w_{opt} is small, we have an improvement in the ratio of

$$\frac{1}{3}\min\left\{\delta_0, \left(1-\frac{2}{r(\delta_0,\delta_1,\delta_2)}\right)\delta_1, 3\delta_2\left(1-\frac{3}{2}\delta_1-\frac{1}{2}\delta_0\right)\right\}.$$

This is positive so long as δ_0, δ_1, and δ_2 are all sufficiently small (e.g. it is equal to 0.003653 at $(\delta_0,\delta_1,\delta_2) = (0.01096, 0.06802, 0.004094)$).

Remark 2. The proof of Theorem 1 from Lemma 2 is deterministic — any algorithm satisfying the bounds in Lemma 2 achieves a $2/3+c$ performance ratio. However, it is not clear how to derandomize Lemma 2, as the optimal assignment (and hence $\mu(N)$) is not known in advance.

In the case where all clauses have length at most 2, it was shown by Poloczek and Schnitger [12] that there is always some ordering of the variables for which Algorithm 1 returns the *optimal* assignment. So finding the best ordering of variables for Algorithm 1 is no easier than solving Maximum satisfiability when all clauses have length 2, which is NP-Hard even to approximate with a factor of $21/22+\epsilon$ [8]. The same applies to Algorithm 2, since this also reduces to a $2-$satisfiability problem.

3. The proof of Theorem 3

Let $p = \frac{\sqrt{\log n}}{n}$, and consider the bipartite graph G formed by first assigning each vertex into either R^0 or L^0 uniformly and independently at random, then connecting each pair of vertices $(x,y) \in R^0 \times L^0$ independently with probability p. We will refer to G as **ϵ-pseudo-random** if it satisfies the following three properties

- P1: $||L^0| - \frac{n}{2}| < n^{3/4}$.
- P2: $||E(G)| - \frac{pn^2}{4}| \leq \epsilon\frac{pn^2}{40}$.
- P3: There is no cut (L,R) in G that simultaneously satisfies

$$|E(L,R)| \geq \left(\frac{1}{2}+\epsilon\right)|E(G)|$$

and

$$\left||L^0 \cap L| - \frac{|L^0|}{2}\right| \leq \epsilon n/10. \tag{9}$$

Lemma 3. *The probability that G is $\epsilon-$pseudo-random is at least* $1-\exp(-\Omega(\epsilon^2 n\sqrt{\log n}-n)) - \exp(-\Omega(\sqrt{n}))$.

Proof. We repeatedly use the following special case of Chernoff's bound for binomial variables ([4], see [13] for this particular version):

Theorem 4. *Let $X = X_1 + \cdots + X_m$ be the sum of iid Bernoulli variables, each with success probability q. Then*

$$\mathbf{P}(|X - mq| > t\sqrt{mq}) \leq 2\max\{e^{-t^2/4}, e^{-t\sqrt{mq}/2}\}.$$

Applying this bound with $q = 1/2$, $m = n$, and $t = n^{1/4}$ gives that the probability P_1 fails to occur is $\exp(-\Omega(n^{1/2}))$.

Applying this bound with $q = p$, $m = n^2/4 - n^{3/2}$, and $t = \epsilon n\sqrt{p}$ gives

$$\mathbf{P}(P_1 \wedge \neg P_2) = \exp(-\Omega(n^2 p\epsilon^2)) = \exp(\omega(\epsilon^2 n\sqrt{\log n})).$$

For P_3, we apply the union bound over all cuts satisfying (9). There are at most 2^n such cuts, and each cut satisfying (9) has at most $m := \frac{n^2(1+\epsilon/10)}{2}$ pairs (x, y) of vertices on opposite sides of the cut. Applying Chernoff with this m, $q = p$, and $t = \epsilon\sqrt{mp}/2$ gives that

$$\mathbf{P}(E(X, Y) > mp(1 + \epsilon/2)) \leq \exp(-\Omega(\epsilon^2 mp/16)) = \exp(-\Omega(\epsilon^2 n\sqrt{\log n}))$$

by our choice of p. The probability that P_3 occurs but not P_2 or P_1 is at most 2^n times this large. □

A natural way of thinking of condition P_3 is that a pseudo-random graph only has one good cut — the only cuts which cut significantly more than half of the edges are those which correlate with the original bipartition. It is therefore enough to show that the cut output by Algorithm 2 usually does *not* exhibit such correlation, that is to say

Lemma 4. *For a random G with this edge probability, Algorithm* 2 *satisfies* (9) *with probability* $1 - \exp(-\Omega(\log^3 n))$.

Combining the above two lemmas, we see that the probability Algorithm 2 succeeds on a random graph is $\exp(-\Omega(\log^3 n))$.

Proof. By symmetry, it suffices to bound the probability L is too large. To do so, we view both G and the cut given by the Algorithm as being exposed simultaneously. In other words, we first expose the ordering $\{x_1, \ldots, x_n\}$ of the vertices, then successively expose for each x_i its assignment to either L^0 and R^0, followed by its edges to those x_j with $j < i$.

We define

- $L_1(t)$ as the number of vertices in $\{x_1, \ldots, x_t\}$ assigned to L^0 in G, and then to L by our greedy algorithm,
- $L_2(t)$ as the number of vertices in $\{x_1, \ldots, x_t\}$ assigned to L^0 in G, and then to R by our greedy algorithm,
- $R_1(t)$ as the number of vertices in $\{x_1, \ldots, x_t\}$ assigned to R^0 in G, and then to L by our greedy algorithm,
- $R_2(t)$ as the number of vertices in $\{x_1, \ldots, x_t\}$ assigned to R^0 in G, and then to R by our greedy algorithm.

We also define

$$g_1(t) = L_1(t) - L_2(t) \text{ and } g_2(t) = R_2(t) - R_1(t)$$

(note the asymmetry in the definition). Our intuition from before is encoded in the following lemma, which states that g_1 and g_2 cannot have much drift unless they are already large.

Claim 1. *At any point in Algorithm* 2*, we have*

$$\mathbf{E}(g_1(t+1)-g_1(t)) \leq \max\{0, pg_2(t)\} \text{ and } \mathbf{E}(g_2(t+1)-g_2(t)) \leq \max\{0, pg_1(t)\}$$

Proof. We focus on the first expectation (the proof for the second is identical). Clearly $g_1(t+1) = g_1(t)$ unless $x_{t+1} \in L^0$. If $x_{t+1} \in L^0$ and $g_2(t) \leq 0$ (meaning that R^0 currently has at least as many vertices assigned to L as to R), then clearly the probability x_{t+1} is assigned to L is at most $1/2$, giving the 0 upper bound on the expectation in this case. So now suppose $g_2(t) > 0$, and let T be an arbitrary collection of $g_2(t)$ vertices in R_2. Let E be the event that x_{t+1} has at least one neighbor in T. We have by symmetry (after throwing out the vertices in T both sides have an equal number of vertices) that

$$\mathbf{E}(g_1(t+1) - g_1(t) | \neg E) = 0,$$

so it follows that

$$\mathbf{E}(g_1(t+1) - g_1(t)) \leq \mathbf{E}(g_1(t+1) - g_1(t) | E)\mathbf{P}(E)$$

The first term is at most one, while the second is at most $pg_2(t)$. □

This claim suggests that $g_1(t)$ and $g_2(t)$ are (in a rough sense) "dominated on average" by the solutions to the differential equation $g'(t) = pg(t)$,

$g(0) = 1$. Lemma 4 corresponds to how the solution to this differential equation, e^{pt}, is still $o(n)$ even at $t = n$ (recall that p was taken to be $o(\log n/n)$).

Now define $f(t) = \frac{20e^{10pt}\log^{3/2} n}{\sqrt{p}}$, and let $f_1(t) = g_1(t) - f(t)$ and $f_2(t) = g_2(t) - f(t)$. Consider the stopping process which halts when either all n vertices are assigned or the larger of $f_1(t)$ and $f_2(t)$ exceeds $t^{1/2}\log^{3/2} n$.

Claim 2. *Both $f_1(t)$ and $f_2(t)$ are supermartingales throughout the stopping process.*

Remark 3. $f(t)$ was essentially chosen in such a way as to make this lemma true. The general idea we are using here (turning the quantities we are tracking into (super)martingales, then using stopping time arguments and concentration inequalities to show that these martingales do not become too large anywhere in the process) was motivated by the use of similar arguments in the differential equations method for random graph processes [15].

Proof. Again we focus only on f_1, since f_2 is identical.

$$\begin{aligned}\mathbf{E}(f_1(t+1) - f_1(t)) &= \mathbf{E}(g_1(t+1) - g_1(t)) - (f(t+1) - f(t)) \\ &\leq \max\{0, p(f(t) + t^{1/2}\log^{3/2} n)\} - (f(t+1) - f(t)),\end{aligned}$$

since we are assuming the process has not yet stopped. This is negative if the first term is 0, since f is increasing. Otherwise we have

$$\begin{aligned}\mathbf{E}(f_1(t+1) - f_1(t)) &\leq p\left(\frac{20e^{10pt}\log^{3/2} n}{\sqrt{p}} + t^{1/2}\log^{3/2} n\right) \\ &\quad - \frac{20e^{10pt}\log^{3/2} n}{\sqrt{p}}(e^{10p} - 1) \\ &\leq p\left(\frac{20e^{10pt}\log^{3/2} n}{\sqrt{p}} + t^{1/2}\log^{3/2} n\right) \\ &\quad - \frac{20e^{10pt}\log^{3/2} n}{\sqrt{p}}(5p) \\ &= \log^{3/2} n\sqrt{p}(\sqrt{pt} - 80e^{10pt})\end{aligned}$$

The claim now follows from the inequality $\sqrt{z} \leq 80e^{10z}$ for nonnegative z. □

Our final claim is that this process likely does not stop until all vertices have been assigned.

Claim 3. *The probability that the process stops before $t = n$ is at most* $\exp(-\Omega(\log^3 n))$.

Proof. By the above, $f_1(t)$ is a submartingale, and we have

$$\begin{aligned}|f_1(t+1) - f_1(t)| &\le 1 + f(t+1) - f(t)\\ &= 1 + \frac{20\log^{3/2} n e^{10pt}}{\sqrt{p}}(e^{10p} - 1)\\ &\le 1 + \frac{400p\log^{3/2} n e^{10pn}}{\sqrt{p}}\\ &\le 1 + \frac{400\log^{7/4} n e^{10\sqrt{\log n}}}{\sqrt{n}}\\ &\le 2,\end{aligned}$$

for n sufficiently large. It follows from Azuma's inequality [2] that for any t

$$\mathbf{P}(f_1(t) \ge t^{1/2}\log^{3/2} n) \le e^{-\log^3 n/8}.$$

Taking the union bound over all t, we see the probability of stopping before $t = n$ due to f_1 becoming too large is $\exp(-\Omega(\log^3 n))$. The same holds for f_2. □

We now can put everything together. If the process does not stop, it follows that $f_1(n) \le n^{1/2}\log^{3/2} n$, from which it follows that

$$\begin{aligned}g_1(n) &\le f(n) + n^{1/2}\log^{3/2} n\\ &\le 20n^{1/2}e^{10\sqrt{\log n}}\log^{3/2} n + n^{1/2}\log^{3/2} n\\ &\le n^{2/3}\end{aligned}$$

for n sufficiently large. □

References

[1] A. Avidor, I. Berkovitch and U. Zwick, Improved approximation algorithms for MAX NAE-SAT and MAX SAT, *Proc. of WAOA* 2005, 27–40. MR2267879

[2] K. Azuma, Weighted sums of certain dependent random variables, *Tohoku Math. J.* 19 (1967), 357–367. MR0221571

[3] J. Chen, D. K. Friesen, and H. Zheng. Tight bound on Johnson's algorithm for maximum satisfiability, *J. Comput. System Sci.* 58(3) (1999), 622–640. MR1705085

[4] H. Chernoff. A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations, *Ann. Math. Stat.* 23(4) (1952), 493–507. MR0057518

[5] L. Engebretsen. Simplified tight analysis of Johnson's algorithm, *Inform. Process. Lett.* 92 (2004), 207–210. MR2094860

[6] P. Erdős. On bipartite subgraphs of a graph, *Matematika Lapok* 18 (1967), 283–288.

[7] M. X. Goemans, D. P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem, *SIAM J. Discrete Math.* 7 (1994), 656–666. MR1299093

[8] Johan Håstad. Some optimal inapproximability results, *J. ACM* 48(4) (2001), 798–859. MR2144931

[9] D. S. Johnson. Approximation algorithms for combinatorial problems, *J. Comput. System Sci.* 9(3) (1974), 256–278. MR0449012

[10] S. Kale, C. Seshadhri. Combinatorial approximation algorithms for MaxCut using random walks. To appear, *ICS* 2011. arXiv:1008.3938v1.

[11] C. Mathieu, W. Schudy. Yet another algorithm for dense max cut: go greedy, *SODA* (2008), 176–182. MR2485302

[12] M. Poloczek, G. Schnitger. Randomized variants of Johnson's algorithm for MAX SAT. To appear, *SODA* 2011.

[13] T. Tao, V. Vu. *Additive Combinatorics*, Cambridge Studies in Advanced Math. **105**, Cambridge University Press, Cambridge, 2006. MR2289012

[14] V. Vazirani. *Approximation Algorithms*, Springer-Verlag, Berlin, 2001. MR1851303

[15] N. Wormald. The differential equation method for random graph processes and greedy algorithms, in *Lectures on Approximation and Randomized Algorithms*, PWN, Warsaw, 1999, pp. 73–155.

[16] M. Yannakakis. On the approximation of maximum satisfiability, *J. Algorithms* 17(3) (1994), 475–502. MR1300260

Kevin P. Costello
School of Mathematics
Georgia Institute of Technology
Atlanta, GA 30332-0160
USA
E-mail address: kcostell@math.gatech.edu

Asaf Shapira
School of Mathematics and School of Computer Science
Georgia Institute of Technology
Atlanta, GA 30332-0160
USA
E-mail address: asafico@math.gatech.edu

Prasad Tetali
School of Mathematics and School of Computer Science
Georgia Institute of Technology
Atlanta, GA 30332-0160
USA
E-mail address: tetali@math.gatech.edu

Received August 26, 2010