BIOINFORMATICS

Vol. 24 ISMB 2008, pages i32–i40
doi:10.1093/bioinformatics/btn173

Optimal pooling for genome re-sequencing with ultra-high-throughput short-read technologies

Iman Hajirasouliha[1,†], Fereydoun Hormozdiari[1,†], S. Cenk Sahinalp[1,*] and Inanc Birol[2]

[1]Lab for Computational Biology, Simon Fraser University, Burnaby, and [2]BC Cancer Agency, Genome Sciences Center, Vancouver, BC, Canada

ABSTRACT

New generation sequencing technologies offer unique opportunities and challenges for re-sequencing studies. In this article, we focus on re-sequencing experiments using the Solexa technology, based on bacterial artificial chromosome (BAC) clones, and address an experimental design problem. In these specific experiments, approximate coordinates of the BACs on a reference genome are known, and fine-scale differences between the BAC sequences and the reference are of interest. The high-throughput characteristics of the sequencing technology makes it possible to multiplex BAC sequencing experiments by pooling BACs for a cost-effective operation. However, the way BACs are pooled in such re-sequencing experiments has an effect on the downstream analysis of the generated data, mostly due to subsequences common to multiple BACs. The experimental design strategy we develop in this article offers combinatorial solutions based on approximation algorithms for the well-known *max n-cut* problem and the related *max n-section* problem on hypergraphs. Our algorithms, when applied to a number of sample cases give more than a 2-fold performance improvement over random partitioning.

Contact: cenk@cs.sfu.ca

1 INTRODUCTION

After decades of research effort, the cost of sequencing an individual human genome via Sanger sequencing (Sanger *et al.*, 1977, 1980, 1982) has now been reduced to the order of 10-million dollars for a 10-fold coverage and require 10 000 instrument days (1 year with 30 instruments) (Solexa web server, 2008). In order to perform fast and cost effective genome sequence comparisons for many biological and medical applications, the cost needs to be further reduced to the order of a few thousand dollars (i.e. a few dollars per megabase) and the time frame of sequencing must be reduced to a few days per instrument. For example, the Solexa sequencing-by-synthesis technology of Illumina (Bennett, 2004) offers about 100-fold improvements over Sanger sequencing in both cost and throughput. Similarly, the pyrosequencing technology of 454 Life Sciences (Margulies *et al.*, 2005) promises massive parallelization of the sequencing process by the use of microchip sensors, improving the speed of Sanger sequencing by a factor of a few hundred. In addition to these, there are other commercial and pre-commercial products from Applied Biosystems, Helicos Biosciences and Visigen Biotechnology, among others that perform either clonal cluster sequencing or single molecule sequencing.

Unfortunately, the massive increase in the throughput offered by the above technologies comes with a shortened read length, and shorter the read length, the more problematic it is to work with a genome that has many longer repeats. While Sanger sequencing offers 500–1000 bps per read, the read lengths of new technologies range from ~25 (e.g. Solexa) to few hundred base pairs (e.g. 454) and as such, they are more suitable for re-sequencing studies. Although current developments in extending these technologies to produce longer, as well as paired end reads puts the *de novo* sequencing studies in the realm of the possible, for the rest of this report we concentrate on the problem of resequencing of individual genomes with short single end reads, and focus on the Solexa technology.

The type of re-sequencing problem we consider in this work is based on bacterial artificial chromosome (BAC) libraries. Benefits of using BACs in re-sequencing studies is 2-fold. (1) It is possible to cluster the short reads obtained from each BAC into small local sets that represent the sequence of one BAC where most of the repeat sequences have a unique copy. (2) One can a priori determine the genomic neighbourhood the BAC is coming from; thus having the reference sequence provides an essential backbone. This can be achieved, for instance, by using the results of fingerprinting or BAC end sequencing experiments. Although on the face of it, a BAC-library-based sequencing effort defeats the benefits of massive parallelization offered by new sequencing technologies, it enables directed sequencing studies possible, where the interest is not on whole genome (re)sequencing, but on investigating a region of interest. Furthermore, recent developments bring the throughput in fingerprinting technology on par with new sequencing technologies (Mathewson *et al.*, 2007).

Solexa sequencing experiments are run on a flow cell with eight lanes, each yielding in the order of 10^8 bps of sequence per run. A typical BAC we consider has a length ranging between 150 Kbps and 250 Kbps. Thus, if we sequence one BAC on each lane, a single run would produce about a 1000-fold coverage per BAC, which is far beyond necessary in a re-sequencing study. Therefore, in order to maximize the throughput of the Solexa technology, hence minimize its cost, it is of key importance to utilize each lane in a more sensible way, such as by sequencing more than one BAC per lane. However, sequencing multiple BACs per lane introduces major difficulties due to repeat sequences that are present in two or more BACs, as they would cause *tanglement* in their assemblies. In order to minimize the repeat elements that are present in multiple BACs, novel algorithmic techniques must be developed.

*To whom correspondence should be addressed.

†The authors wish to be known that, in their opinion, the first two authors should be regarded as joint First Authors.

Available algorithms for DNA fragment assembly (Pevzner *et al.*, 2001a,b; Roach *et al.*, 1995), especially those for short reads (Chaisson *et al.*, 2004; Chaisson and Pevzner, 2008; Warren *et al.*, 2006) all suffer from the presence of repeats within the genome region to be assembled (Margulies *et al.*, 2005). However, the high potential of high-throughput short-read technologies have promoted the development of novel protocols and algorithms such as the short read assembly protocol (SHARP; Sundquist *et al.*, 2007) that aim to address the shortcomings of short-read technologies. Our goal in this article is to help the available fragment assembly methods, especially in the context of genome re-sequencing, by providing (near) optimal utilization of the multiple lanes available by the Solexa technology, while keeping the cost of fragment assembly at a minimum. For this purpose, we present algorithms that partition the input BACs into multiple lanes in a way to minimize potential assembly errors due to repeat sequences shared among BACs in each lane. Our algorithms are quite efficient in practice and provide significant improvement to the cost of fragment assembly over random partitioning strategies.

2 PROBLEM DEFINITION AND GENERAL ALGORITHMIC APPROACH

Given a set of BACs sampling in some known specific genomic regions, and the reference sequence, our ultimate goal is to construct the sequence of each BAC using the results of optimally designed Solexa sequencing experiments. Consider a set of m BACs sequenced with a read length of k (typically 25–50 bps); the problem we address in this work is, how can we partition this set into n groups (or pools) of approximately $h=m/n$ BACs, such that individual reads can be attributed to the BACs they come from as correctly as possible. In other words, how can we minimize number of shared k-mers by multiple BACs in each pool in the overall configuration?

Strictly speaking, the problem does not have a solution, because before conducting any sequencing experiment, it is not possible to know how many shared k-mers the BACs in question would have. However, note that our focus is on re-sequencing studies, and if we have even a crude idea on the genomic coordinates of the BACs, we can approximate that missing information by using the reference sequence.

One other hurdle in designing a globally optimal experiment is the rapid proliferation of number of possible configurations. For instance, if we would like to pool $m=150$ BACs into groups of $h=10$, we would need to consider

$$\frac{\prod_{i=0}^{14}\binom{150-10i}{10}}{15!} > 10^{152} \tag{1}$$

configurations in an exhaustive search. Since it is infeasible to search all these configurations for finding the global optimum, we propose an algorithmic approach to guide us to an 'approximately' optimal setup.

Once again we note that our goal is to partition a given set of m BACs into pools of size h each, with the purpose of minimizing the number of potential assembly errors due to sequences that repeat in multiple BACs within a pool. Potential assembly errors due to sequences that repeat within a single BAC, on the other hand, are beyond the scope of this study and are not investigated here.

We define our problem more formally as follows.

2.1 Pooling problem

Given a set of m BACs that are to be placed into n pools of approximately $h=m/n$ BACs in each, let $C_{i,b}$ be the number of BACs in a pool P_i that share a particular k-mer b. If we denote the cost of b as $f(C_{i,b})$, we can write an overall cost function for a given configuration as

$$J=\sum_{i=1}^{n}\sum_{\forall b\in P_i} f(C_{i,b}) \tag{2}$$

and the problem becomes one of selecting the optimum partitioning $P^*=\{P_i^*\}$ that minimizes the cost J.

One can attribute alternative costs for shared k-mers b, two of which are

1. $f(C_{i,b})=\binom{C_{i,b}}{2}$;
2. $f(C_{i,b})=C_{i,b}-1$.

For the remainder of this article, we will restrict our attention to these two formulations for reasons explained below.

The pooling problem under the cost function $f(C_{i,b})=\binom{C_{i,b}}{2}$ is a minimization problem for the number of k-mers that are shared between pairs of BACs which are in the same pool. This can be reduced to a well-known combinatorial problem called, *n-clustering problem*, as follows: construct a complete graph G where each BAC B is represented with a unique vertex v_B and given any pair of BACs B and B', set the weight of the edge (V_B, V'_B) to the number of common k-mers in B and B'. The *n-clustering problem* is a problem of partitioning G into vertex sets, such that the sum of edge weights between vertices that belong to the same partition is minimized.

Unfortunately even obtaining a constant factor approximation to the n-clustering problem is NP-hard (Sahni and Gonzalez, 1976). Thus in Section 3.1, we first reduce this problem to another combinatorial problem known as the *max n-cut*. Although this problem is also NP-hard (Sahni and Gonzalez, 1976), we solve it by the use of a simple local search procedure within an approximation factor of $1-1/n$. For $n=15$, this implies an approximation factor of 0.93. Although this approximate solution to the *max n-cut* problem does not provide a guarantee on the approximation for the pooling problem, it gives good results in practice.

An extension to the pooling problem is the *balanced pooling* problem, where we seek to minimize the cost of partitioning BACs into pools, such that the number of BACs in each pool is *exactly* $h=m/n$. As can be expected, even approximating the balanced pooling problem within a constant factor is NP-hard. Thus, in Section 3.2 we reduce the balanced pooling problem to the *max n-section* problem, which is the balanced version of the *max n-cut* problem. We again describe an algorithm to approximately solve this problem within a factor of $1-1/n$. Although the latter algorithm does not provide a guarantee on the approximation factor it obtains for the balanced pooling problem, it yields good results in practice once again.

The pooling problem under the second cost function $f(C_{i,b})=C_{i,b}-1$ is a minimization problem for the number of BACs within a pool that shares each k-mer, summed over all k-mers. This is a generalized version of the pooling problem with the first cost function as will be explained below.

The pooling problem under the second cost function can be reduced to a *hypergraph partitioning problem* as follows. Let G be a hypergraph where each BAC B is represented with a unique vertex v_B and each subset S of at most d vertices are connected with a hyperedge e_S. In the most general case of the problem $d=m$. The weight of e_S, namely $w(e_S)$ is the number of k-mers that occur in *all* BACs in S and occur in no other BACs. Consider a partition of G into non-overlapping vertex sets. For a given subset S of vertices, let $\#(S)$ be the number of pools that have at least one vertex of S. Then the cost of e_S with respect to this partitioning is $w(e_S)\cdot(|S|-\#(S))$; here $|S|$ denotes the number of BACs in set S.

Our hypergraph partitioning problem defines a search for partitioning G into vertex sets/pools so as to minimize the total cost of the hyperedges with respect to this partition.

Unfortunately, the above hypergraph partitioning problem requires $O(\binom{m}{n})$ space to just represent all the hyperedges. As this represents faster than exponential growth with the number of BACs, even setting up an instance of the problem on a computer is not feasible for the parameter values we are interested in.

Notice that if we restrict the maximum number of vertices that can be incident to a hyperedge, d, to 2 (rather than m) then our hypergraph partition problem reduces to the *n-clustering problem* and thus to the pooling problem with the first cost function. Now we can consider versions of the hypergraph partition problem with $d=3, 4, \ldots$ as 'approximations' to our general hypergraph partitioning problem with $d=m$.

Our hypergraph partitioning problem with $d>1$ is NP-hard (Sahni and Gonzalez, 1976) even to approximate within a constant factor. We reduce it to another hypergraph partitioning problem in which the cost of a hyperedge e_S with respect to a partitioning is $w(e_S)\cdot(\#(S)-1)$ and the goal is to *maximize* the total cost of all hyperedges. In this article we show how to solve this problem approximately within a factor of $1-d/2n$. For $d=3$ and $n=15$, our algorithm provides a 0.9 approximation factor, again sufficiently close to 1. The algorithm employs a greedy approach and is quite efficient.

We also consider a balanced version of this hypergraph partitioning problem which asks for maximizing the total cost of all hyperedges with respect to a partition, provided that the number of vertices per each pool is exactly $h=m/n$. Again we provide a $(1-d/2n)$-approximation algorithm to this problem. This algorithm is quite involved, as further described in the next section, employing a solution to the minimum weighted bipartite matching towards a greedy selection of the vertices in each partition.

3 METHODS

In this section we give detailed descriptions of the approximation algorithms we use for solving the pooling problem, both balanced and unbalanced versions, under the two cost functions we presented earlier.

3.1 The pooling problem under $f(C_{i,b})=\binom{C_{i,b}}{2}$

The pooling problem (unbalanced version) under our first cost function can be formulated as the well-known max n-cut problem as follows.

Input: A weighted undirected graph $G(V,w)$, with the vertex set V representing the set $\mathcal{B}=\{B_1,B_2,\ldots,B_{150}\}$ of BACs, and the edge weights w. For any vertex pair v_B, v'_B, $w(v_B,v'_B)$ is the number of common k-mers in the corresponding BACs, B and B'.

Output: A partitioning of V into pools $P=\{P_1,P_2,\cdots,P_n\}$, $\bigcup_{i=1}^{n} P_i = V$, which maximizes the following objective function:

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\sum_{v_B\in P_i, v'_B\in P_j} w(v_B,v'_B).$$

3.1.1 *A local search algorithm for max n-cut (LSMnC)*

1. Randomly partition the vertex set V of the graph G into n different pools.
2. If there exists a vertex $v\in V$ such that it is assigned to pool P_i ($v\in P_i$) and there exists a pool P_j such that $\sum_{u\in P_i} w(v,u)\geq\sum_{x\in P_j} w(v,x)$, then move vertex v from the pool P_i into the pool P_j.
3. Repeat second step until no change can occur.

The above simple local search algorithm, when applied to the general max n-cut problem may take too much time before it terminates. However, for our specific problem, the running time of the above algorithm is guaranteed to be polynomial with m and the maximum length of a BAC as shown below.

PROOF. Let t be the total weight of the edges of the graph G, which is polynomial with m and the maximum number of k-mers in a BAC. It is clear that in each step of the local search algorithm, the total weight of edges going between pools increases by at least one. Therefore in the worst case, this algorithm terminates after t steps.■

In practice, the running time of the above local search algorithm is in second order for $m=150$, $n=15$ and maximum BAC length of $250K$. The approximation factor achieved by the above algorithm is $1-1/n$ as shown below.

PROOF. Consider an arbitrary vertex $v\in V$ and assume P_i is the cluster containing v after the termination of the local search. We have:

$$\forall 1\leq j\leq n:\ \sum_{\forall u\in P_i} w(v,u)\leq\sum_{\forall x\in P_j} w(v,x)\Rightarrow \tag{3}$$

$$\sum_{\forall u\in P_i} w(v,u)\leq\frac{1}{n}\sum_{j=1}^{n}\sum_{\forall x\in P_j} w(v,x)\Rightarrow \tag{4}$$

$$\sum_{j=1,j\neq i}^{n}\sum_{\forall u\in P_j} w(v,u)\geq\frac{n-1}{n}\sum_{j=1}^{n}\sum_{\forall x\in P_j} w(v,x) \tag{5}$$

The expression $\sum_{j=1,j\neq i}^{n}\sum_{\forall u\in P_j} w(u,v)$ in left-hand side of last equation represents the total weight of all edges in the 'cut' incident to vertex v. Also, the expression $\sum_{j=1}^{n}\sum_{\forall x\in P_j} w(v,x)$ in the right-hand side of the same equation represents the total weight of all edges incident to vertex v. Since v has been chosen arbitrary from the vertex set V, we have:

$$\sum_{\forall v\in V}\sum_{j=1,j\neq i}^{n}\sum_{\forall u\in P_j} w(v,u)\geq\sum_{\forall v\in V}\frac{n-1}{n}\sum_{j=1}^{n}\sum_{\forall x\in P_j} w(v,x) \tag{6}$$

According to the above inequality, the total weight of the edges which are incident to a pair of vertices that do not belong to the same pool is at least $(n-1)/n$ times the total weight of the edges in G, and thus the local search provides a $1-1/n$ approximation. ■

3.2 The balanced pooling problem under $f(C_{i,b})=\binom{C_{i,b}}{2}$

The balanced pooling problem asks to partition m BACs into n pools so as to minimize the above cost function, with the additional constraint that the number of BACs per each pool is exactly $h=m/n$. This is known as max n-section problem for which a local search algorithm by (Gaur *et al.*, 2007) guarantees an approximation factor of $1-1/n$.

For each vertex $u\in V$ and for each set of vertices belonging to a pool P_i, let $w(u,P_i)=\sum_{v\in P_i} w(u,v)$. The local search algorithm for the max n-section problem works as follows.

3.2.1 Local search algorithm for max n-section (LSMnS)

1. Initialization. Partition the vertices V into n pools $P_1, P_2, \ldots, P_n$ uniformly at random such that $|P_1| \leq |P_2| \leq \ldots \leq |P_n| \leq |P_1|+1$.
2. Iterative step. Find a pair of vertices $v \in P_i$ and $u \in P_j$ $(i \neq j)$, such that $w(v, P_i - v) + w(u, P_j - u) \geq w(v, P_j - u) + w(u, P_i - v)$. If such a pair exists move u to the cluster P_i and v to the cluster P_j.
3. Termination. If no such pair of vertices is found then terminate.

This algorithm is an simple extension to the local search algorithm described in Section 3.1 and it is easy to show that it terminates in time polynomial with m and the maximum length of a BAC. Furthermore, it was shown in (Gaur *et al.*, 2007) that this algorithm has a guaranteed approximation factor of $1-1/n$.

3.3 The pooling problem under $f(C_{i,b}) = C_{i,b} - 1$

We now focus on the cost function $f(C_{i,b}) = C_{i,b} - 1$. As we discussed in the problem definition (Section 2), the pooling problem under cost function $f(C_{i,b}) = C_{i,b} - 1$ can be reduced to a *hypergraph partitioning problem* as follows. Let G be a hypergraph where each BAC B is represented with a unique vertex v_B and each subset S of at most m vertices, are connected with a hyperedge e_S. The weight of e_S, namely $w(e_S)$ is the number of k-mers that occur in *all* BACs in S and occur in no other BACs.

Consider a partitioning of V, the vertex set of G, into non-overlapping pools $P = \{P_1, \ldots, P_n\}$. For a given subset S of vertices, let $\#(S)$ be the number of pools of P_i that have at least one vertex of S. Then the cost of e_S with respect to P is $w(e_S) \cdot (|S| - \#(S))$ and the goal of the hypergraph partitioning problem is to *minimize* the total cost of all hyperedges.

The 'dual' of this hypergraph partitioning would be another partitioning problem where the cost of e_S with respect to P is $w(e_S) \cdot (\#(S) - 1)$, and the goal is to *maximize* the total cost of all hyperedges.[1]

Input: A weighted hypergraph $G(V, w)$, with vertex set V, and weights $w(e_S)$ for each hyperedge e_S (which connects the set $S \subseteq V$ for $|S| \leq d$).

Output: A partitioning of vertices V into pools $P = \{P_1, P_2, \cdots, P_n\}$, $\bigcup_{i=1}^{n} P_i = V$, which maximizes the following objective function:

$$\sum_{S \subseteq V, |S| \leq d} w(e_S) \cdot (\#(S) - 1).$$

We give a greedy algorithm to solve the above hypergraph partitioning problem. The algorithm, at each iteration x randomly picks a vertex $v_x \in V'_x$, where V'_x is the set of vertices not processed so far, and adds it to one of the pools P_i as described below.

Before we provide further details, we give some definitions. Let $P_{x,i}$ be the set of vertices in pool P_i before iteration x and let $P_x = \{P_{x,1}, \ldots, P_{x,n}\}$. (Thus $V'_x = V - \bigcup_{i=1}^{i=n} P_{x,i}$.)

Given some $S \subset V$, let $\#_{P_x}(S)$ denote the number of pools $P_{x,i} \in P_x$ which include at least one vertex of S.

Also let $\widetilde{\#}_{P_{x,k}}(S)$ be a boolean function such that $\widetilde{\#}_{P_{x,k}}(S) = 1$ if $P_{x,k} \cap S = \emptyset$ and $\widetilde{\#}_{P_{x,k}}(S) = 0$ otherwise.

3.3.1 A greedy algorithm for hypergraph partitioning problem (GAHP)

1. As an initial step, set $V'_0 = V$ and $\forall_{i=1}^{n} P_{0,i} = \emptyset$.
2. In each iteration $x \in \{1, \ldots, m\}$, randomly pick a vertex $v_x \in V'_x$ and put v_x into the pool

$$k_x = \arg\max_k \sum_{S \subseteq V, v_x \in S} w(e_S) \cdot \widetilde{\#}_{P_{x,k}}(S).$$

(Thus, $\forall i \leq n, i \neq k_x, P_{x+1,i} = P_{x,i}$ and $P_{x+1,k_x} = P_{x,k_x} \cup \{v_x\}$).

[1]The two problems are equivalent as $|S|$ is a constant.

The above algorithm achieves an approximation factor of $1 - d/2n$ as shown below.

PROOF. First, we need to find a lower bound on the total cost $w(e_S) \cdot (\#(S) - 1)$ with respect to the pool set $P_{m+1} = \{P_{m+1,1}, P_{m+1,2}, \cdots P_{m+1,n}\}$ returned by the algorithm at the end of iteration m.

It is not hard to see that for any set of vertices S (for the remainder of the proof, all sets S we consider will satisfy $|S| \leq d$), for which $v_x \in S$:

$$w(e_S) \cdot \#_{P_{x+1}}(S) = w(e_S) \cdot \#_{P_x}(S) + w(e_S) \cdot \widetilde{\#}_{P_{x,k_x}}(S) \tag{7}$$

Thus,

$$w(e_S) \cdot \#_{P_{m+1}}(S) = \sum_{x=1, v_x \in S}^{m} w(e_S) \cdot \widetilde{\#}_{P_{x,k_x}}(S).$$

Now taking the sum of above equation for all possible hyperedges we would get:

$$\sum_{S \subseteq V} w(e_S) \cdot \#_{P_{m+1}}(S) = \sum_{S \subseteq V} \sum_{x=1, v_x \in S}^{m} w(e_S) \cdot \widetilde{\#}_{P_{x,k_x}}(S) = \sum_{x=1}^{m} \sum_{S \subseteq V, v_x \in S} w(e_S) \cdot \widetilde{\#}_{P_{x,k_x}}(S) \tag{8}$$

For bounding the left hand side of equation 8, we will consider an arbitrary iteration x.

$$n \cdot \sum_{S \subseteq V, v_x \in S} w(e_S) \cdot \widetilde{\#}_{P_{x,k_x}}(S) \geq \sum_{S \subseteq V, v_x \in S} \sum_{i=1}^{n} w(e_S) \cdot \widetilde{\#}_{P_{x,i}}(S). \tag{9}$$

By adding up the right hand side of equation 9 over all values of x we get:

$$\sum_{x=1}^{m} \sum_{S \subseteq V, v_x \in S} \sum_{i=1}^{n} w(e_S) \cdot \widetilde{\#}_{P_{x,i}}(S) = \sum_{S \subseteq V} \sum_{x=1, v_x \in S}^{m} \sum_{i=1}^{n} w(e_S) \cdot \widetilde{\#}_{P_{x,i}}(S). \tag{10}$$

For bounding equation 10 we first have to argue for any arbitrary $S \subseteq V$ we have,

$$\sum_{x=1, v_x \in S}^{m} \sum_{i=1}^{n} w(e_S) \cdot \widetilde{\#}_{P_{x,i}}(S) \geq w(e_S) \cdot (n + \cdots (n - |S| + 1))$$

Thus,

$$\sum_{S \subseteq V} \sum_{x=1, v_x \in S}^{m} \sum_{i=1}^{n} w(e_S) \cdot \widetilde{\#}_{P_{x,i}}(S) \geq \sum_{S \subseteq V} w(e_S) \cdot (n + \cdots (n - |S| + 1)) \tag{11}$$

Now using Equations 9–11 we will have:

$$\sum_{x=1}^{m} \sum_{S \subseteq V, v_x \in S} w(e_S) \cdot \widetilde{\#}_{P_{x,1},k_x}(S) \geq \frac{\sum_{S \subseteq V} w(e_S) \cdot (|S| \cdot n - (1 + 2 + \cdots (|S| - 1)))}{n}. \tag{12}$$

Utilizing Equations 12 and 8 we conclude:

$$\sum_{S \subseteq V} w(e_S) \cdot (\#_{P_{m+1}}(S) - 1) \geq \frac{\sum_{S \subseteq V} w(e_S) \cdot (|S| \cdot n - \frac{|S|(|S|-1)}{2})}{n} - \sum_{S \subseteq V} w(e_S) = \frac{n \cdot \sum_{S \subseteq V} w(e_S) \cdot (|S| - 1) - \sum_{S \subseteq V} w(e_S) \cdot \frac{|S|(|S|-1)}{2}}{n} \tag{13}$$

Now to find the approximation factor for this greedy algorithm we need to find an upper bound of the optimal solution. It is easy to see that for

even optimal partitioning $\sum_{e_S} w(e_S)\cdot(\#(S)-1) \leq \sum_{e_S} w(e_S)\cdot(|S|-1)$. Thus, the approximation factor α can be bounded as:

$$\alpha = \frac{\sum_{S\subseteq V} w(e_S)\cdot(\#_{P_{m+1}}(S)-1)}{\sum_{S\subseteq V} w(e_S)\cdot(\#_{P_{opt}}(S)-1)}$$

$$\geq \frac{\sum_{S\subseteq V} w(e_S)((|S|-1)n-\frac{|S|(|S|-1)}{2})}{n\cdot\sum_{S\subseteq V} w(e_S)\cdot(|S|-1)}$$

$$= 1-\frac{\sum_{S\subseteq V} w(e_S)\cdot\frac{|S|(|S|-1)}{2}}{n\cdot\sum_{S\subseteq V} w(e_S)\cdot(|S|-1)}$$

$$= 1-\frac{\sum_{S\subseteq V} w(e_S)\cdot(|S|\cdot(|S|-1))}{2n\cdot\sum_{S\subseteq V} w(e_S)\cdot(|S|-1)} \quad (14)$$

Now, as $|S|<d$ we can easily see that $\alpha \geq 1-d/2n$. ■

3.4 The balanced pooling problem under $f(C_{i,b})=C_{i,b}-1$

Our last algorithm deals with the *balanced* pooling problem under the cost function $f(C_{i,b})=C_{i,b}-1$, for which we give a greedy approximation algorithm. We remind the reader that there are $m=nh$ vertices to be assigned into n pools, and eventually each pool must have exactly h vertices.

3.4.1 A greedy algorithm for balanced hypergraph partitioning (GABHP) The algorithm starts with a set of n empty pools, $P=\{P_1,\ldots,P_n\}$, and at each iteration x, it selects a set of n arbitrary vertices, say $Y_x=\{y_{1,x},\ldots,y_{n,x}\}$, which are not assigned to any of the pools yet, and adds them to the pools such that each pool receives *exactly* one new vertex. Let the set of vertices in pool P_i at the beginning of iteration x be denoted by $P_{i,x}$ and let $P_x=\{P_{x,i},\ldots,P_{x,n}\}$. Thus, in iteration x, each $y_{j,x}$ is assigned to exactly one of the pools $P_{x,i}$.

For any set of vertices $S\in V$, let $\lambda(y_{j,x},P_{x,i},S)$ be a boolean function defined as follows.

$$\lambda(y_{j,x},P_{x,i},S)=\begin{cases}1 & \text{if } \exists\, y_{\ell,x}\neq y_{j,x}: y_{\ell,x}\in S,\ \ y_{\ell,x}\in P_{x,i}\\ 0 & \text{otherwise.}\end{cases}$$

Intuitively, for a given vertex $y_{j,x}$, a pool $P_{x,i}$, and a vertex set S, $\lambda(y_{j,x},P_{x,i},S)$ is equal to zero if and only if no other vertex $y_{\ell,x}$ incident to the hyperedge e_S has already been assigned to the pool $P_{x,i}$.

Then, for each pool $P_{x,i}$, we define the *marginal* cost function $\mu(y_{j,x},P_{x,i})$ with respect to the potential assignment of $y_{j,x}$ to $P_{x,i}$, as follows.

$$\mu(y_{j,x},P_{x,i})=\sum_{S\subseteq V, y_{j,x}\in S} w(e_S)\cdot\lambda(y_{j,x},P_{x,i},S)$$

We now construct a *new* complete bipartite graph H with vertex sets Y_x and P_x such that for any vertex $y_{j,x}\in Y_x$ and pool $P_{x,i}\in P_x$, there exists an edge in H with weight $\mu(y_{j,x},P_{x,i})$. Then we find a perfect *minimum* weighted matching for H, i.e. a perfect matching where the sum of weights of the edges in the matching has the minimum possible value by using the well-known Hungarian algorithm (Munkres, 1957).[2]

For each j, $1\leq j\leq n$, let $\pi(y_{j,x})$ be the pool which is matched to $y_{j,x}$ in the prefect minimum bipartite matching of the graph H. We add the vertex $y_{j,x}$ to the pool $\pi_{x,i}$.

We run the above iterative step for $x=1,\ldots,h$ so as to assign each one of the $m=nh$ vertices into one pool in a balanced manner.

The above algorithm gives an approximation to the balanced hypergraph partitioning problem within a factor of $1-d/2n$, where d is the maximum number of vertices that can be incident to a hyperedge. The proof for the approximation factor is similar to that for the unbalanced hypergraph partitioning problem and thus is omitted.

[2]We actually solve the dual, weight maximization problem after subtracting each edge weight from the maximum edge weight.

4 RESULTS AND DISCUSSION

We report results on two sets of BACs (with $m=150$) on which we tested our algorithms for both balanced and unbalanced pooling problem using both cost functions. We start by noting that for both data sets the results obtained by the balanced pooling algorithms turned out to be almost identical to those obtained by the unbalanced pooling algorithms for each of the two cost functions we used. In other words, the cost of the partition obtained by the LSMnC algorithm was almost identical to that of the LSMnS algorithm and the cost obtained by the GAHP algorithm was very similar to that of the GABHP problem. It is also interesting to note that the unbalanced pooling algorithms returned very balanced partitions.[3] We compare the performance of these algorithms with that of random partitioning of BACs into pools.

We measure the performance of our algorithms and that of random partitionings with respect to the general cost function $f(C_{i,b})=C_{i,b}-1$. The total cost of a particular partitioning of a set of m BACs into pools $P_1,\ldots,P_n$, is $J=\sum_{i=1}^{n}\sum_{\forall b\in P_i} f(C_{i,b})$. In order to compute the cost J for a partitioning, we construct, for each pool P_i, the *joint trie* of the k-mers (for this study $k=50$) of all BACs in P_i, denoted by T_i. The trie T_i can be constructed in time linear with the total lengths of the BACs in P_i as per the linear time algorithms for suffix tree construction (Mccreight, 1976; Sahinalp and Vishkin, 1994). During the construction of T_i, at each leaf corresponding to a k-mer b, we maintain the labels of the specific BACs that include b. By going through all leaves of each T_i, we compute J in time linear with the total lengths of the BACs.

We used two data sets in our experiments, each consisting of 150 BACs. The first set of clones were collected in the high-resolution analysis of *lymphoma genomes project* at the BC Genome Sciences Centre. They represent regions of interest in the genome of a tumor sample, where there are marked local deviations from the reference human genome. The BAC coordinates are deduced by aligning BAC fingerprints to the reference genome (Krzywinski *et al.*, 2007) and are confirmed by BAC end sequencing experiments.
The second set is a synthetic library of clones with a mean size of 182 kb and a standard deviation of 34 kb, representing a random sampling of the finished regions of the reference human genome, hg18.

4.1 Pooling experiments with LSMnC/LSMnS algorithms

We first compare the performance of our local search methods LSMnC/LSMnS with random partitions (denoted *ranPool*). Although these methods were designed to minimize the cost of pooling with respect to the cost function $f(C_{i,b})=\binom{C_{i,b}}{2}$, we report their performance with respect to the second cost function, $f(C_{i,b})=C_{i,b}-1$, as, ultimately that is the cost function we would like to minimize.

Note that ranPool is known to give an *expected* approximation factor of $1-1/n$ for the max n-cut problem. However, LSMnC will guarantee a *worst case* approximation factor of $1-1/n$ for the max n-cut problem.

In Figures 1(a) and 2(a), we depict how the cost of the ranPool method for 5000 independent trials, as well as the cost

[3] The number of BACs obtained by the unbalanced pooling algorithms were never less than 7 and never more than 12 in any pool.

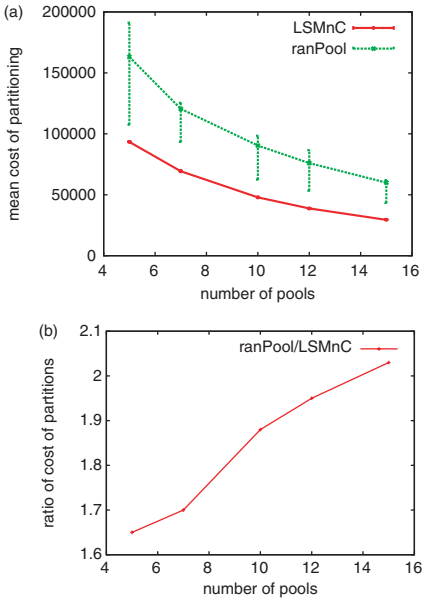

Fig. 1. A comparison of the cost of ranPool and LSMnC/LSMnS methods with respect to the number of pools on the lymphoma data set. (a) The actual cost of LSMnC/LSMnS method (red) and the cost of ranPool (green) with respect to the number of pools (mean, upper quartile and lower quartile results reported for 5000 independent runs) with respect to the number of pools. (b) The ratio of the average cost of ranPool (on 5000 independent runs) and the cost of LSMnC/LSMnS methods.

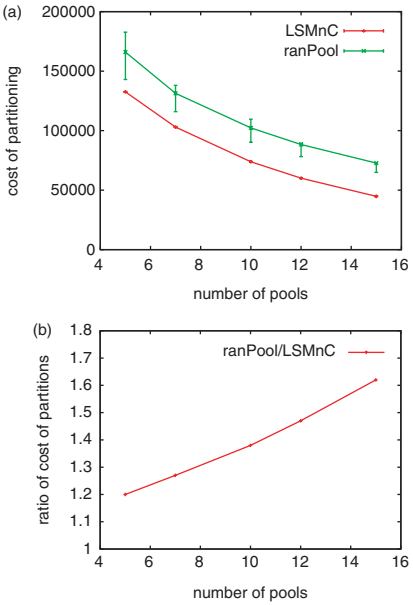

Fig. 2. A comparison of the cost of ranPool and LSMnC/LSMnS methods with respect to the number of pools on the synthetic data set. (a) The actual cost of LSMnC/LSMnS method (red) and the cost of ranPool (green) with respect to the number of pools (mean, upper quartile and lower quartile results reported for 5000 independent runs) with respect to the number of pools. (b) The ratio of the average cost of ranPool (on 5000 independent runs) and the cost of LSMnC/LSMnS methods.

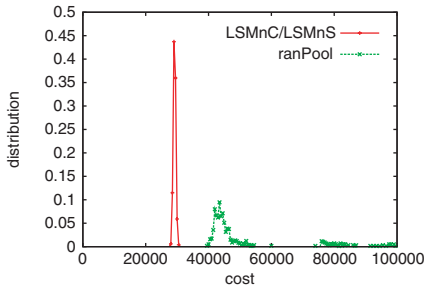

Fig. 3. The distribution of cost obtained by ranPool and LSMnC/LSMnS for $n = 15$ on the lymphoma data set for 5000 independent runs.

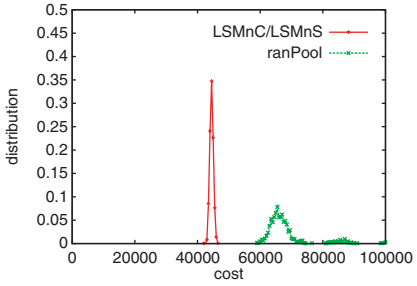

Fig. 4. The distribution of cost obtained by ranPool and LSMnC/LSMnS for $n = 15$ on the synthetic data set for 5000 independent runs.

of LSMnC/LSMnS method varies with respect to the number of pools on the *lymphoma* and the *synthetic* data sets (we show the mean, the highest and the lowest 25% cost for the ranPool method). We also depict how the ratio between the (average) cost of the ranPool and the cost of the LSMnC/LSMnS method changes with respect to the number of pools in Figures 1(b) and 2(b). We can observe that as the number of pools increases, the costs of both ranPool and LSMnC/LSMnS decrease for both data sets. Perhaps more interestingly, we can also observe that as the number of pools increases, the ratio between the (average) cost of ranPool and that of LSMnC/LSMnS increases as well.

For a specific number of pools ($n = 15$) we depict the distribution of the costs obtained in the 5000 independent runs of both ranPool and the LSMnC/LSMnS methods on both the lymphoma and the synthetic data sets in Figures 3 and 4, respectively. As can be seen, the costs obtained by independent trials of the LSMnC/LSMnS method (each starting with a unique random partitioning) are highly concentrated around the mean. Furthermore, even the minimum cost obtained by the ranPool method is well above the maximum cost returned by the LSMnC/LSMnS method.

We can conclude that the results obtained by the LSMnC/LSMnS approach is much better than those obtained by ranPool. For example, at $n = 15$ the LSMnC/LSMnS approach provides a factor 2 improvement over the average cost of 5000 independent runs of ranPool on the lymphoma data set. The cost improvement of LSMnC/LSMnS approach with respect to the minimum cost obtained by ranPool after 5000 independent trials is 1.4 on the same data set.

4.2 Pooling experiments with GAHP/GABHP algorithms

The local search algorithms LSMnC and LSMnS aim to 'minimize' the cost of pooling with respect to the cost function $f(C_{i,b})=\binom{C_{i,b}}{2}$; however, the cost obtained by these algorithms were considerably lower than that obtained by ranPool even with respect to the second cost function—which provides a more accurate performance measure.

Our second set of algorithms, GAHP/GABHP are designed to 'minimize' the cost with respect to the second cost function. They are flexible in the sense that one can set up the value of d as desired; for $d=n$, the optimal solution to the hypergraph partitioning indeed minimizes the cost function $f(C_{i,b})=C_{i,b}-1$. We tried the two algorithms for both $d=2$ and 3 in order to evaluate their advantage over the local search algorithms as well as random partitionings. The running time of both GAHP and the GABHP algorithms are exponential in d (the number of hyperedges grow exponentially with increasing d), thus it is of crucial importance to know up to which value of d, an improvement in performance could be expected. A significant performance improvement by GAHP/GABHP methods using $d=3$ over LSMnC/LSMnS methods (which solve the hypergraph partitioning problem for $d=2$) may imply that d should be increased to 4 or more for better performance.[4]

[4]Unfortunately the approximation factor achieved by the GAHP/GABHP methods deteriorate with increasing k. Note that for $d=3$, the approximation factor guaranteed by the GAHP/GABHP method to the hypergraph partitioning problem is $1-3/2n$ which is equal to 0.9 for $n=15$.

In Figures 5(a) and 6(a) we compare the (mean value as well as the highest and lowest 25%) costs obtained by 5000 independent runs of the ranPool method and that of the GAHP/GABHP approach on the *lymphoma* and the *synthetic* data sets. Figures 5(a) and 6(a) show how the costs change with respect to the increasing number of pools. We also show how the ratio between the average cost of the ranPool and the cost of GAHP/GABHP methods change with respect to the number of pools in Figure 7(a) and (b).

We investigate the effect of using hypergraphs with $d=3$ over the use of ordinary graphs with $d=2$ on the GAHP/GABHP methods in Figure 7(a) and (b). As can be observed, the performance improvement achieved by increasing d form 2 to 3 is negligible for both data sets. It may be possible to explain this observation by investigating the distribution of repeat sequences among the BACs in the two data sets. The number of k-mers which are repeated in exactly two BACs are 100–200 times more than those repeated in three BACs or more; see Figure 8(a) and (b) for the distribution of hyperedge weights in the two data sets. Thus, the total weight of hyperedges incident to three vertices or more is insignificant in comparison to edges that are incident to exactly two vertices. As a result, on the two data sets we used, the hypergraph partitioning algorithms largely 'ignore' the hyperedges whose contribution to the total cost is very small. We expect that the performance of the GAHP/GABHP methods for $d=3$ will be superior to that for $d=2$ if highly repetitive BACs are sequenced.

We finally compare the two alternative algorithmic methods we introduce in this article; LSMnC/LSMnS and GAHP/GABHP (for $d=2$). In Figure 9(a) and (b), a comparison of the two methods are

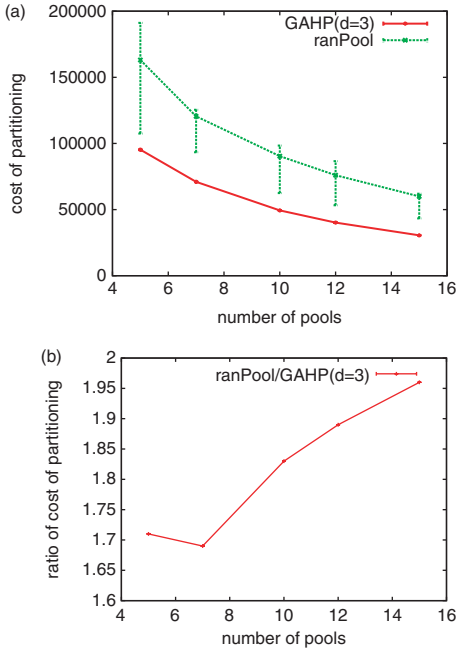

Fig. 5. A comparison of the cost of ranPool and GAHP/GABHP ($d=3$) methods with respect to the number of pools on the lymphoma data set. (a) The actual cost of LSMnC/LSMnS method (red) and the cost of ranPool (green) with respect to the number of pools (mean, upper quartile and lower quartile results reported for 5000 independent runs) with respect to the number of pools. (b) The ratio of the average cost of ranPool (on 5000 independent runs) and the cost of GAHP/GABHP methods.

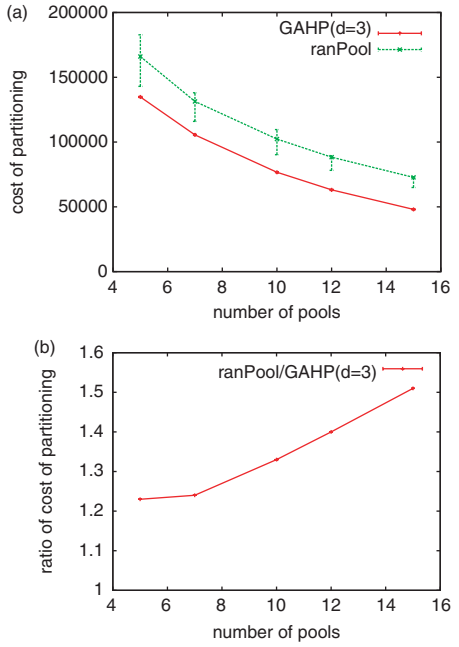

Fig. 6. A comparison of the cost of ranPool and GAHP/GABHP ($d=3$) methods with respect to the number of pools on the synthetic data set. (a) The actual cost of LSMnC/LSMnS method (red) and the cost of ranPool (green) with respect to the number of pools (mean, upper quartile and lower quartile results reported for 5000 independent runs) with respect to the number of pools. (b) The ratio of the average cost of ranPool (on 5000 independent runs) and the cost of GAHP/GABHP methods.

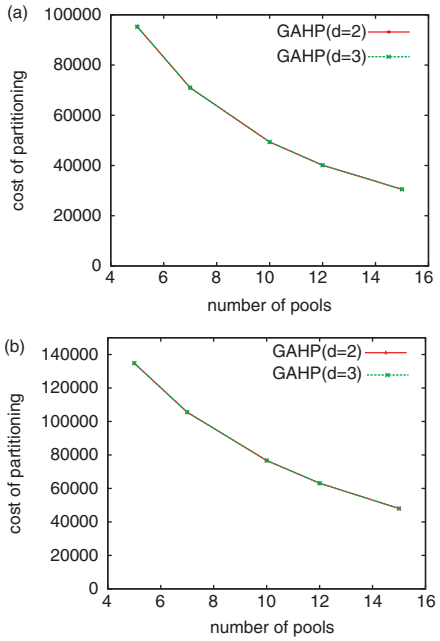

Fig. 7. A comparison of the cost of GAHP (GABHP) method for $d=2$ (red) and $d=3$ (green) with respect to the number of pools, (a) on the lymphoma data set, (b) on the synthetic data set.

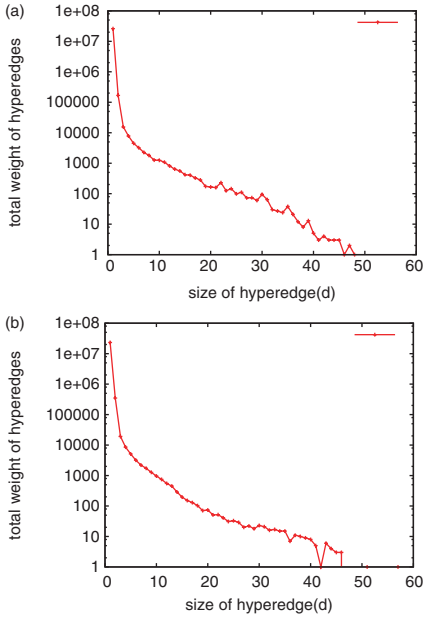

Fig. 8. The distribution of hyperedge weights (in log scale) among 5000 BACs, (a) from the lymphoma data set, (b) from the synthetic data set.

provided for both data sets. Interestingly enough, the performance of LSMnC/LSMnS method is slightly better than the GAHP/GABHP method for both data sets. This indicates that LSMnC/LSMnS (which finds a local optimum) outperforms a greedy based method (GAHP/GABHP), which does not necessarily find a local optimum. However, we believe that if regions of DNA with high repetitions

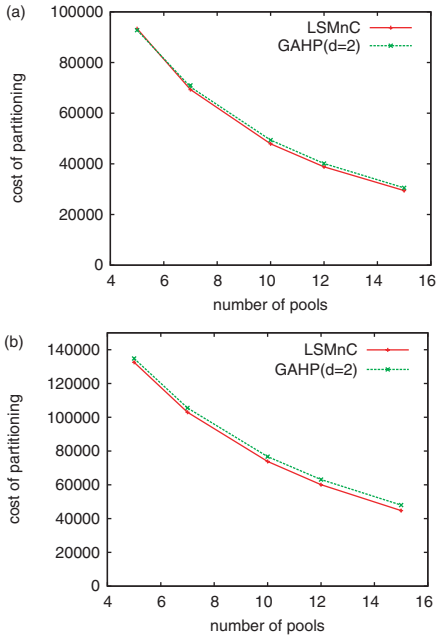

Fig. 9. A comparison of the cost of LSMnC/LSMnS method (red) and GAHP/GABHP method for $d=2$ (green) with respect to the number of pools, (a) on the lymphoma data set, (b) on the synthetic data set.

are used as the input, GAHP/GABHP (especially for $d>2$) would give better results compared to LSMnC/LSMnS and even GAHP/GABHP for $d=2$.

Conflict of Interest: none declared.

REFERENCES

Bennett,S. (2004) Solexa ltd. *Pharmacogenomics*, **5**, 433–438.

Chaisson,M. *et al.* (2004) Fragment assembly with short reads. *Bioinformatics*, **20**, 2067–2074.

Chaisson,M.J. and Pevzner,P. (2008) Short read fragment assembly of bacterial genomes. *Genome Res.*, **18**, 324–330, doi:10.1101/gr.7088808.

Gaur,D.R. *et al.* (2007) The capacitated max k-cut problem. *Math. Progr.*, doi:10.1007/s10107-007-0139-z.

Mathewson,C.A. *et al.* (2007) *Chapter 5: Large-Scale BAC Clone Restriction Digest Fingerprinting*. John Wiley and Sons, Hoboken, NJ, pp. 19.1–19.21.

Krzywinski,M. *et al.* (2007) A bac clone fingerprinting approach to the detection of human genome rearrangements. *Genome Biol.*, **8**, R224, doi:10.1186/gb-2007-8-10-r224.

Margulies,M. *et al.* (2005) Genome sequencing in open microfabricated high-density picoliter reactors. *Nature*, **437**, 376–380.

Mccreight,E.M. (1976) A space-economical suffix tree construction algorithm. *JACM*, **23**, 262–272.

Munkres,J. (1957) Algorithms for the assignment and transportation problems. *J. Soc. Ind. Appl. Math.*, **5**, 32–38.

Pevzner,P. *et al.* (2001a) A new approach to fragment assembly in dna sequencing. *RECOMB*, 256–265.

Pevzner,P.A. *et al.* (2001b) An eulerian path approach to dna fragment assembly. *Proc. Natl. Acad Sci.*, **98**, 9748–9753.

Roach,J. *et al.* (1995) Pairwise end sequencing: a unified approach to genomic mapping and sequencing. *Genomics*, **26**, 234–353.

Sahinalp,S.C. and Vishkin,U. (1994) Symmetry breaking for suffix tree construction. In *Proceedings of the STOC*, pp. 300–309.

Sahni,S. and Gonzalez,T. (1976) *P*-complete approximation problems. *J. ACM*, **23**, 555–565.

Sanger,F. *et al.* (1977) Dna sequencing with chain-terminating inhibitors. *Proc. Natl Acad. Sci.*, **74**, 5463–5467.

Sanger,F. *et al.* (1980) Cloning in single-stranded bacteriophage as an aid to rapid DNA sequencing. *J. Mol. Biol.*, **143**, 161–178.

Sanger,F. *et al.* (1982) Nucleotide sequence of bacteriophage lambda. *DNA*, **161**, 729–773.

Solexa web server (2008). http://www.solexa.com.

Sundquist,A. *et al.* (2007) Whole-genome sequencing and assembly with high-throughput, short read technologies. *PLoS ONE*, **2**, e484.

Warren, R. *et al.* (2006) Assembling millions of short DNA sequences using ssake. *Bioinformatics*, **23**, 500–501.