Alma Mater Studiorum
Università degli Studi di Bologna
DEIS

Tabu Search for the Founder Sequence Reconstruction Problem: A Preliminary Study

Andrea Roli and Christian Blum

January 10, 2009

DEIS Technical Report no. DEIS-LIA-004-09 *LIA Series no. 94*

Tabu Search for the Founder Sequence Reconstruction Problem: A Preliminary Study

Andrea Roli [1] and Christian Blum [2]

[1] *DEIS, Campus of Cesena*
University of Bologna
Cesena, Italy
andrea.roli@unibo.it

[2] *ALBCOM Research Group*
Universitat Politècnica de Catalunya
Barcelona, Spain
cblum@lsi.upc.edu

January 10, 2009

Abstract. The problem of inferring ancestral genetic information in terms of a set of founders of a given population arises in various biological contexts. In optimization terms, this problem can be formulated as a combinatorial string problem. The main problem of existing techniques, both exact and heuristic, is that their time complexity scales exponentially, which makes them impractical for solving large-scale instances. We developed a new constructive heuristic and a tabu search method with the explicit aim of providing solutions in a reduced amount of computation time. Experimental results show that when the number of founders grows, our algorithms have advantages over the ones proposed in the literature.

Keywords: *Metaheuristics, founder sequences reconstruction problem, bioinformatics*

Contents

1 Introduction

Technical advances in sequencing of genetic material has led to a rapid growth of available DNA sequences and haplotyped sequences. Given a sample of sequences from a population of individuals (for example, humans) one may try to study the evolutionary history of the chosen individuals. This is important, for example, for the discovery of the genetic basis of complex diseases. In case the population from which the sample sequences are taken has evolved from a relatively small number of founders, the evolutionary history can be studied by trying to reconstruct the sample sequences as fragments from the set of founder sequences. This genetic model, which is central to the problem tackled in this paper, was used, for example, in [5, 6]. Many findings from biological studies support the validity of this model, as, for example, [4]:

> The *Ferroplasma* type II genome seems to be a composite from three ancestral strains that have undergone homologous recombination to form a large population of mosaic genomes.

The major problem is that neither the number of founder sequences, nor the founder sequences themselves, may be known. Ukkonen [5] proposes a computational problem that, given the number k of founder sequences, consists in finding a set of k sequences such that the set of sample sequences, also called recombinants, can be reconstructed using as few fragments as possible. This problem is known as the *founder sequence reconstruction problem* (FSRP) or the *minimum mosaic problem* [6] and it is NP-complete [3]. A technical description of the problem is given in the following section.

The first algorithm that was developed for the FSRP is based on dynamic programming [5]. However, this algorithm does not scale well when the number of founders or the number/length of the recombinants grows. The authors of [6] proposed an exact solver based on tree search, called RecBlock. This solver can also be run as a heuristic with varying levels of sophistication. While the results of RecBlock are very good for rather small number of founders, it still does not scale well when, for example, the number of founders grows. This was our motivation for the work presented in this paper. With the goal of developing an algorithm that scales better than the existing techniques, we first developed a very fast constructive heuristic, and then a so-called tabu search method [2]. Tabu search is an artificial intelligence technique based on local search which belongs to the class of metaheuristic algorithms [1]. In this work we present a preliminary study on the performance of these algorithms.

The remainder of the paper is organized as follows. In Section 2 we technically introduce the FSRP. While Section 3 is devoted to the introduction of our algorithms, we present an experimental evaluation in Section 4. Finally, conclusions and an outlook to future work are given in Section 5.

1 1 0 1 1 0 1		b b b b b|c c
1 0 1 0 0 0 1	0 1 1 0 1 0 0	c c c c c c c
0 1 1 1 1 1 1	1 1 0 1 1 1 1	a a a|b b b b
0 1 1 0 1 0 0	1 0 1 0 0 0 1	a a a a a a a
1 1 0 0 0 1 1		b b b|c c|b b

(a) Set of recombinants $\mathcal{C}$ (b) Set of founders $\mathcal{F}$ (c) Decomposition

Figure 1. *(a) shows a set of 5 recombinants in matrix form. Assuming that the number of founders is fixed to 3, (b) shows a valid solution as a matrix of 3 founders. Denoting the first founder by "a", the second founder by "b", and the third one by "c", (c) shows a decomposition of the recombinants matrix into fragments taken from the founders. This decomposition produces the minimum number of breakpoints points, namely 4. Note that breakpoints are marked by vertical lines. This example is reproduced from [6].*

2 The Founder Sequence Reconstruction Problem

The founder sequence reconstruction problem (FSRP) can technically be described as follows. Given is a set of m *recombinants* $\mathcal{C} = \{C_1, \ldots, C_m\}$. Each recombinant C_i is a string of length n over a given alphabet Σ: $C_i = c_{i1}c_{i2}\ldots c_{in}$ with $c_{ij} \in \Sigma\ \forall\, j$. In this work we will consider a typical biological application where the recombinants are haplotyped sequences and $\Sigma = \{0, 1\}$. The symbols 0 and 1 encode the two most common alleles of each haplotype site.

A *candidate solution* to the problem consists of a set of k *founders* $\mathcal{F} = \{F_1, \ldots, F_k\}$. Each founder F_i is a string of length n over the alphabet Σ: $F_i = f_{i1}f_{i2}\ldots f_{in}$ with $f_{ij} \in \Sigma\ \forall\, j$. A candidate solution $\mathcal{F}$ is a *valid solution* if the set of recombinants $\mathcal{C}$ can be *reconstructed* from $\mathcal{F}$. This is the case when each $C_i \in \mathcal{C}$ can be decomposed into a sequence of $p_i \leq n$ fragments (that is, strings) $Fr_{i1}Fr_{i2}\ldots Fr_{ip_i}$, such that each fragment Fr_{ij} appears at the same position in at least one of the founders. Hereby, a decomposition with respect to a valid solution is called *reduced* if two consecutive fragments do not appear in the same founder. Moreover, for each valid solution $\mathcal{F}$ we can derive in polynomial time (see [6]) a so-called *minimal decomposition*. This is a decomposition where $\sum_{i=1}^{n} p_i - n$ is minimal. In the following we call this number the objective function value of $\mathcal{F}$ and denote it by $f(\mathcal{F})$. In biological terms, $f(\mathcal{F})$ is called the number of *breakpoints* of $\mathcal{C}$ with respect to $\mathcal{F}$.

The optimization goal considered in this paper is the following one. Given a fixed k, that is, a fixed number of founders, find a valid solution $\mathcal{F}^*$ that minimizes $f(\cdot)$. For an example, see Fig. 1.

Algorithm 1 Heuristic RecBlock

1: INPUT: a set $\mathcal{C}$ of m recombinants of length n, and k, the required number of founders
2: Let $\mathcal{F}$ be an empty matrix with k rows and n columns
3: The current number of breakpoints is 0, that is, $nbp = 0$
4: **for** $j = 1, \ldots, n$ **do**
5: Choose a binary string Col of length k for column j of $\mathcal{F}$ such that the number add of additional breakpoints is minimized
6: $nbp = nbp + add$
7: **end for**
8: OUTPUT: a solution $\mathcal{F}$ together with the total number of breakpoints nbp

3 Our Algorithm

We developed an algorithm that consists of a constructive heuristic with the subsequent application of tabu search. In the following we first focus on the description of the constructive heuristic, whereas tabu search is outlined afterwards.

3.1 Constructive Heuristic

To our knowledge, existing tree search methods for the FSRP, including constructive heuristics, all use the following way of constructing solutions. They regard a solution $\mathcal{F}$ as a matrix with k rows and n columns. In such a matrix (also denoted by $\mathcal{F}$ for simplicity reasons) row i represents founder i, for all $i = 1, \ldots, k$. Starting from an empty matrix, each construction step concerns filling the next empty column, starting from the first column. For filling a column, all possible binary strings of length k are considered and tested. For example, the RecBlock heuristic presented in [6] and pseudo-coded in Algorithm 1 chooses at each construction step $j = 1, \ldots, n$, the binary string that adds the least number of breakpoints to the partial solution under construction.[1] The disadvantage of this way of filling columns is the fact that the number of possibilities at each construction step is exponential in k. In our experiments (see Section 4) we will show that this makes the heuristic impractical for rather large values of k.

With this disadvantage of RecBlock in mind, we designed a new and fast way of filling founder matrix columns in the framework of the constructive mechanism used by RecBlock (see Algorithm 1). In the following we regard $\mathcal{C}$ (the set of recombinants) to be a matrix with m rows and n columns. The solution construction

[1] This description concerns the lightest heuristic version of RecBlock that is run with options -D0 and -C1.

process starts by filling the first column of $\mathcal{F}$, which is done as follows. First, we compute the fraction p of 0-entries in the first column of $\mathcal{C}$. Then we introduce two counters; counter n_0 for the 0-entries in the first column of $\mathcal{F}$, and counter n_1 for the 1-entries in the first column of $\mathcal{F}$. Both counters are initialized to 1 to ensure at least one 0-entry, respectively one 1-entry. Finally, $k-2$ times we draw a random number q from $[0,1]$, and we increase n_0 in case $q \leq p$, while we increase n_1 otherwise. The first column is then composed of n_0 0-entries, followed by n_1 1-entries.

After filling the first column, some data structures are initialized. For each row i of $\mathcal{C}$ we keep a variable cp_i that stores the position of the last breakpoint. These variables are initialized to 0, because no breakpoint exists yet. More specifically, $cp_i = 0$, for $i = 1, \ldots, m$. Moreover, we keep a variable rep_i that stores the index of the founder that represents row i of $\mathcal{C}$ after the last breakpoint cp_i. For all rows of $\mathcal{C}$ with a 0-entry in the first column this variable is initialized to 0, while for each row of $\mathcal{C}$ with a 1-entry the respective variable is initialized to $n_0 + 1$, that is, the first row of $\mathcal{F}$ with a 1-entry in the first column. More specifically, $rep_i = 0$ if $c_{i0} = 0$, and $rep_i = n_0 + 1$ otherwise.

All remaining columns are filled as follows. Let us assume that the first $j-1$ columns are already filled, which means that the column under consideration is column j. The positions of column j are filled one after the other, that is, starting from row 1. For filling position f_{ij} we first count the number n_0 of rows of $\mathcal{C}$ that are represented by founder i and that have a 0-entry in position j. More specifically, n_0 is the number of rows r of $\mathcal{C}$ with $rep_r = i$ and $c_{rj} = 0$. Correspondingly, n_1 is the number of rows r of $\mathcal{C}$ with $rep_r = i$ and $c_{rj} = 1$. In case $n_0 > n_1$, we set $f_{ij} = 0$. In case $n_1 > n_0$, we set $f_{ij} = 1$. Otherwise, that is, when $n_0 = n_1$, we choose a value for f_{ij} uniformly at random. Finally, we try to change the representant of the rows of $\mathcal{C}$ that, after assigning a value to f_{ij}, can not be represented anymore by their current representant. In case $f_{ij} = 0$, this concerns all rows r of $\mathcal{C}$ with $rep_r = i$ and $c_{rj} = 1$; similarly in case $f_{ij} = 1$. For all these rows r of $\mathcal{C}$ we search for a new representing founder l (where $i < l \leq k$) that can equally represent r starting from breakpoint cp_r, that is, we search for a row l in $\mathcal{F}$ (where $i < l \leq k$) such that $c_{rs} = f_{ls}$, for all $s = cp_r, \ldots, j-1$. In case such a founder l can be found, we set $rep_r = l$, and the search for an alternative representant for row r is stopped.

As a last step, after filling all the positions of a column j of $\mathcal{F}$, the variables cp_r and rep_r must be updated for all rows r of $\mathcal{C}$ for which $f_{rep_r j} \neq c_{rj}$. In such a case, we are looking for the founder i with the minimum l such that $c_{rs} = f_{is}$, for all $s = l, \ldots, j$. After identifying such a founder i, we set $cp_r = l$ and $rep_r = i$. This step concludes the description of our constructive heuristic.

3.2 Tabu Search

The tabu search phase has the goal of further reducing the number of breakpoints of the solution that was heuristically constructed. The search space explored by tabu search is the whole set of *valid* solutions (see Section 2). Before explaining the neighborhood that we chose, we remind that the transition from one fragment to the next one is called a breakpoint. This means that a decomposition $Fr_{i1}, \ldots, Fr_{id_i}$ of recombinant C_i contains $d_i - 1$ breakpoints. In the following we will denote the index of the founder from which fragment Fr_{ij} is taken by $F(j)$. A move in the neighborhood that we designed concerns the removal of exactly one of these breakpoints. In particular, for removing a breakpoint from decomposition $Fr_{i1}, \ldots, Fr_{id_i}$ we must change either founder $F(j)$ or $F(j+1)$ such that fragments Fr_{ij} and Fr_{ij+1} can be both taken from only one founder. This can be tried for $j = 1, \ldots, d_i - 1$. In the first case we must change the founder with index $F(j)$ such that $f_{F(j)r} = c_{ir}$ for $r = \sum_{s=1}^{j-1} |Fr_{is}| + 1, \ldots, \sum_{s=1}^{j} |Fr_{is}|$, and in the second case we must change the founder with index $F(j+1)$ such that $f_{F(j+1)r} = c_{ir}$ for $r = \sum_{s=1}^{j} |Fr_{is}| + 1, \ldots, \sum_{s=1}^{j+1} |Fr_{is}|$. Note that such a move is not guaranteed to reduce the total number of breakpoints, nor to produce a feasible solution. Nevertheless, in our experiments it proved to be quite effective.The neighborhood structure induced by the type of move described above was used inside a tabu search with dynamic tabu list length, which is varied at each iteration by randomly choosing a value in the range $[tl_{min}, tl_{max}]$, where tl_{min} and tl_{max} are parameters of the algorithm. For efficiency reasons, the neighborhood is restricted by considering only the breakpoints of one, randomly chosen, recombinant. The tabu list contains the most recently selected recombinants. Move evaluations are done incrementally, so as to make the search process faster. In the high level description of the algorithm, see Figure 2, we denote with $\mathcal{N}(\mathcal{F})|C_r$ the set of feasible solutions in the neighborhood of the current solution $\mathcal{F}$ with respect to the moves resulting from the decomposition of recombinant C_r .

4 Experimental Evaluation

We tested a multi-start version of our constructive heuristic[2] (denoted by *heuristic*) and the tabu search procedure (*TS*) against three variants of RecBlock: (a) the exact version (*rec-exact*), (b) a sophisticated heuristic variant (*rec-heuristic*), and (c) the lightest heuristic version corresponding to options -D0 -C1 (*rec-D0C1*). We implemented our algorithms in C++, compiled with -O3 option. All programs were run on nodes equipped with a 2.4 GHz AMD Opteron^{TM} Processor with 1GB of RAM. TS was run for 400 iterations and then restarted from a new heuristically

[2]Due to random decisions during the solution construction, the solution provided by our constructive heuristic is potentially different each time.

Algorithm 2 Tabu search for the FSRP

INPUT: a set $\mathcal{C}$ of m recombinants of length n, and k, the required number of founders
Let $\mathcal{F}_0$ be the initial solution provided by our constructive heuristic
InitializeTabuList(TL)
$\mathcal{F}_{best} \leftarrow \mathcal{F}_0$; $nbp_{best} \leftarrow f(\mathcal{F}_{best})$
while maximum number of iterations not reached **do**
  $r \leftarrow$ **RandomInt**(1,m)
  $\mathcal{N}_a(\mathcal{F}) \leftarrow \{\mathcal{F}' \in \mathcal{N}(\mathcal{F}) | C_r \text{ s.t. } r \notin TL \vee f(\mathcal{F}') < nbp_{best}\}$
  $\mathcal{F}'' \leftarrow \text{argmin}\{f(\mathcal{F}') \text{ s.t. } \mathcal{F}' \in \mathcal{N}_a(s)\}$
  UpdateTabuList(TL)
  $\mathcal{F} \leftarrow \mathcal{F}''$
  if $f(\mathcal{F}) < nbp_{best}$ **then**
    $\mathcal{F}_{best} \leftarrow \mathcal{F}$; $nbp_{best} \leftarrow f(\mathcal{F}_{best})$
  end if
end while
OUTPUT: solution $\mathcal{F}_{best}$

constructed solution until the time limit was reached. The range of the tabu list length was set to $[1, 10]$.

We used a benchmark set composed of randomly generated instances with m recombinants and $n = 2m$, $n = 3m$, or $n = 5m$ sites. We generated five instances per combination of m and n. The generated instances are valid and not reducible, i.e., no columns can be removed without affecting the optimal solution. Each instance was considered for several numbers of founders, more specifically, we considered $k \in \{3, \ldots, 10\}$. Each algorithm was applied to each instance and each k exactly once, with a maximum CPU time limit of one hour. Results are summarized in Tables 1 and 2 in which the average of the best solution values and the standard deviation are reported. Statistics are taken over the 5 instances per number of recombinants and sites; the values that are statistically better than the others[3] are marked by an asterisk.

Results show that RecBlock has a very good performance, but when instance size increases, in at least one dimension, also *rec-heuristic* can not return a solution. In these cases, our heuristic is slightly worse than *rec-D0C1* while TS is consistently better than the competitors. It is also interesting to consider the trend of execution time. In Figure 4 we plot the ratio of the time at which *heuristic* and *rec-D0C1* return the first solution, as a function of the number of founders and sites. As expected, *rec-D0C1* scales exponentially and the speedup we can achieve with our heuristic is up to 4 orders of magnitude.

[3]That is, for which the null hypothesis is rejected. The Mann-Whitney test was applied.

Table 1. *Instances with 30 recombinants. Best solution values returned by algorithms in 1 hour of CPU time, averaged over 5 random instances. The symbol '—' indicates that no solution was returned. In brackets, the standard deviation is reported. Statistically significantly better results are marked.*

30 recombinants sites , founders	**rec-exact**	**rec-heuristic**	**rec-D0C1**	**heuristic**	**TS**
60 , 3	573.8 (12.38) *	579.4 (11.5) *	604 (16.11)	594.2 (13.08)	583 (11.79) *
60 , 4	445.4 (5.59) *	450.2 (6.53) *	494.2 (18.27)	479.6 (9.18)	459.6 (7.5)
60 , 5	—	385.2 (7.85) *	425.4 (10.06)	412.2 (8.87)	395.8 (9.36)
60 , 6	—	340.6 (5.18) *	383.6 (5.13)	367.6 (6.88)	352 (6.6)
60 , 7	—	303.6 (5.64) *	353.8 (10.06)	335.2 (7.22)	318.2 (6.76)
60 , 8	—	274.6 (3.71) *	331 (8.75)	311.6 (5.77)	291.2 (4.38)
60 , 9	—	—	307.4 (10.29)	288.6 (6.47)	270.4 (4.51) *
60 , 10	—	—	294 (9)	268.4 (4.56)	251.8 (4.32) *
90 , 3	877.2 (2.95) *	885.2 (3.96)	917.8 (12.83)	910.8 (8.01)	892 (4.58)
90 , 4	684.2 (3.27) *	689.4 (4.34)	749.4 (5.81)	741.6 (7.16)	711.8 (4.02)
90 , 5	—	596.2 (4.49) *	653 (14.23)	645.6 (3.21)	618.6 (3.78)
90 , 6	—	525 (2.45) *	584.2 (7.85)	580.2 (4.32)	552.8 (4.76)
90 , 7	—	469.4 (3.91) *	542 (22.29)	529.8 (6.76)	500.4 (4.16)
90 , 8	—	424.4 (2.7) *	498.8 (17.47)	491 (4)	461.2 (2.17)
90 , 9	—	—	469.8 (6.1)	456.2 (4.92)	427.8 (3.9) *
90 , 10	—	—	438.2 (7.05)	427 (4.85)	398.8 (3.35) *
150 , 3	1468.8 (21.7) *	1482.6 (17.87) *	1533.4 (16.46)	1529 (16.12)	1500.6 (18.65)
150 , 4	1140.4 (9.42) *	1154.4 (5.18)	1249 (18.72)	1253.2 (12.77)	1200.8 (10.76)
150 , 5	—	991.6 (8.2) *	1083.8 (20.68)	1090.8 (9.88)	1041.6 (10.78)
150 , 6	—	876.2 (6.26) *	971.2 (3.49)	980 (4.8)	932 (9.14)
150 , 7	—	—	888.8 (12.03)	897 (4.47)	848.2 (6.42) *
150 , 8	—	—	819.2 (5.36)	831.8 (4.6)	783.2 (4.71) *
150 , 9	—	—	770.2 (12.64)	773 (3.39)	727.6 (3.71) *
150 , 10	—	—	715.2 (9.52)	724.8 (2.68)	676.6 (3.78) *

5 Conclusions and Outlook

In this paper we have proposed a constructive heuristic and a tabu search method for tackling large size instances of the FSRP. Results on random instances show that our tabu search method outperforms the heuristic version of RecBlock on large size instances. We are currently working on an enhanced version of the constructive heuristic with stochastic lookahead, and the design of an iterated local search metaheuristic is ongoing.

Table 2. *Instances with 50 recombinants. Best solution values returned by algorithms in 1 hour of CPU time, averaged over 5 random instances. The symbol '—' indicates that no solution was returned. In brackets, the standard deviation is reported. Statistically significantly better results are marked.*

50 recombinants sites , founders	**rec-exact**	**rec-heuristic**	**rec-D0C1**	**heuristic**	**TS**
100 , 3	1765.4 (16.96) *	1784.4 (14.64)	1837.8 (31.03)	1821.2 (18.02)	1789 (15.18)
100 , 4	1377.6 (10.88) *	1392.2 (9.39)	1481.8 (24.63)	1483.8 (8.23)	1425.2 (13.95)
100 , 5	—	1225.2 (14.72) *	1305 (17.36)	1301.2 (15.06)	1260.6 (14.43)
100 , 6	—	1095.8 (13.92) *	1177.6 (12.16)	1188.4 (15.08)	1140.2 (11.21)
100 , 7	—	997.8 (10.99) *	1087.8 (15.9)	1101.4 (9.89)	1049.4 (9.13)
100 , 8	—	920.4 (9.71) *	1026.8 (6.3)	1034.8 (9.78)	976 (9.62)
100 , 9	—	—	963.8 (14.82)	976.2 (13.59)	915 (11.73) *
100 , 10	—	—	918.8 (6.76)	928.4 (10.64)	868 (8.34) *
150 , 3	2631.2 (22.88) *	2660.6 (22.74) *	2740.8 (29.3)	2722.6 (23.99)	2677.4 (23.56)
150 , 4	2056.8 (5.72) *	2078.8 (6.91)	2194.2 (26.48)	2240.6 (6.88)	2148.2 (8.41)
150 , 5	—	1823.2 (8.32) *	1936.8 (12.74)	1965 (9.46)	1894.8 (8.35)
150 , 6	—	1635.8 (12.85) *	1759.6 (9.66)	1794.8 (6.8)	1717.8 (7.16)
150 , 7	—	1493.2 (11.19) *	1644 (12.53)	1668 (9.22)	1578.8 (10.18)
150 , 8	—	—	1528.8 (13.24)	1562.8 (10.01)	1475.2 (10.96) *
150 , 9	—	—	1443.8 (6.69)	1479.2 (14.74)	1386 (8.86) *
150 , 10	—	—	1376.8 (15.59)	1403.2 (11.56)	1314.8 (5.81) *
250 , 3	4421 (22.06) *	4466.2 (20.46)	4597.8 (33.69)	4601.6 (15.53)	4514.8 (11.95)
250 , 4	3448.67 (4.73) *	3490.8 (10.76)	3728.8 (8.53)	3813.6 (7.54)	3634.2 (13.88)
250 , 5	—	3071.4 (15.98) *	3258.4 (33.25)	3344 (21.12)	3218.8 (11.69)
250 , 6	—	2754.4 (14.17) *	2967.8 (24.77)	3046.8 (11.37)	2915.8 (17.31)
250 , 7	—	2510.6 (9.4) *	2735.6 (20.89)	2832 (13.82)	2686.6 (11.8)
250 , 8	—	—	2570.6 (22.06)	2648.8 (17.77)	2504.8 (12.93) *
250 , 9	—	—	2422 (30.24)	2505.8 (14.79)	2358 (9.67) *
250 , 10	—	—	2304.4 (28.06)	2378.8 (7.22)	2237.2 (7.6) *

Acknowledgements

This work was supported by grant TIN2007-66523 (FORMALISM) of the Spanish government. In addition, Christian Blum acknowledges support from the *Ramón y Cajal* program of the Spanish Ministry of Science and Technology of which he is a research fellow.

References

[1] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

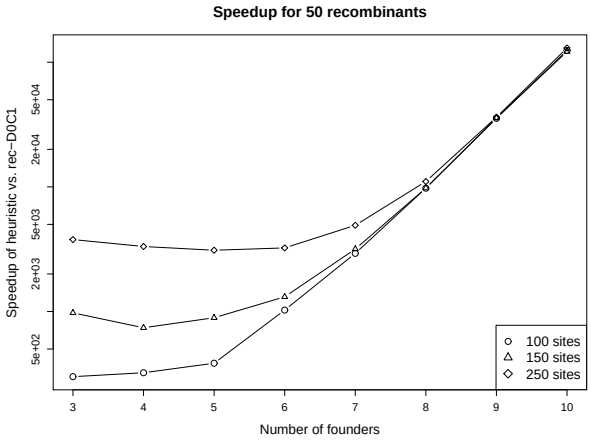

Figure 2. *Speedup achieved by our constructive heuristic procedure w.r.t.* rec-D0C1. *Along the y-axis, the ratio of the execution time of the heuristic and* rec-D0C1 *is reported.*

[2] Fred W. Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[3] P. Rastas and E. Ukkonen. Haplotype inference via hierarchical genotype parsing. In *Proceedings of WABI2007 – 7th Workshop on Algorithms in Bioinformatics*, 2007.

[4] G. W. Thyson, J. Chapman, P. Hugenholtz, E. Allen, R. Ram, P. Richardson, V. Solovyev, E. Rubin, D. Rokhsar, and J. Banfield. Community structure and metabolism through reconstruction of microbial genomes from the environment. *Nature*, (428):37–43, 2004.

[5] E. Ukkonen. Finding founder sequences from a set of recombinants. In R. Guigó and D. Gusfield, editors, *Proceedings of WABI 2002 – Proceedings of the 2nd Workshop on Algorithms in Bioinformatics*, volume 2452 of *Lecture Notes in Computer Science*, pages 277–286. Springer Verlag, Berlin, 2002.

[6] Y. Wu and D. Gusfield. Improved algorithms for inferring the minimum mosaic of a set of recombinants. In *Proceedings of CPM 2007 – Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching*, volume 4580 of *Lecture Notes in Computer Science*, pages 150–161. Springer Verlag, Berlin, 2008.