Available online at www.sciencedirect.com
ScienceDirect
Discrete Optimization 5 (2008) 74–87
DISCRETE OPTIMIZATION
www.elsevier.com/locate/disopt

Asynchronous Teams for probe selection problems☆

Cláudio N. Meneses, Panos M. Pardalos*, Michelle Ragle

Department of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, FL 32611, USA

Received 2 September 2005; received in revised form 18 November 2007; accepted 25 November 2007
Available online 8 January 2008

Abstract

The selection of probe sets for hybridization experiments directly affects the efficiency and cost of the analysis. We propose the application of the Asynchronous Team (A-Team) technique to determine near-optimal probe sets. An A-Team is comprised of several different heuristic algorithms that communicate with each other via shared memories. The A-Team method has been applied successfully to several problems including the Set Covering Problem, the Traveling Salesman Problem, and the Point-to-Point Connection Problem, and lends itself well to the Probe Selection Problem. We designed and developed a C++ program to run instances of the Minimum Cost Probe Set and Maximum Distinguishing Probe Set problems. A program description and our results are presented in the paper.

Keywords: Probe selection; Asynchronous Team; Near-optimal solution; Heuristic

1. Introduction

Scientists in the past have relied upon culture methods to study microbial communities. The amount of information they could derive from such studies was minimal given the tremendous complexity and diversity of microbial communities. It is believed that more than 99% of most microbial communities consist of uncultured microorganisms [5]. The development of rRNA-based analysis methods has resulted in the identification of thousands of these previously unknown microorganisms. One such method is that of oligonucleotide fingerprinting which uses radioactively labeled DNA probes to identify a large number of clones through a series of hybridization experiments [1,8].

The design and selection of probe sets plays a vital role in hybridization experiments. In this paper we propose the use of the Asynchronous Team (A-Team) technique to determine near-optimal probe sets for the given sets of clones. We focus our attention on two probe selection problems called the Maximum Distinguishing Probe Set (MDPS) and the Minimum Cost Probe Set (MCPS) [1].

The A-Team method was proposed by Souza and Talukdar [3]. An A-Team is comprised of several different heuristic algorithms, called agents, that communicate with each other by means of shared memories. The shared

☆ This work has been partially supported by NSF, NIH and CRDF grants.
* Corresponding author.
E-mail addresses: claudio@ufl.edu (C.N. Meneses), pardalos@ufl.edu (P.M. Pardalos), raglem@ufl.edu (M. Ragle).

1572-5286/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.disopt.2007.11.006

memories store solutions generated by agents. Each agent can make its own decisions about inputs, scheduling and resource allocation. The A-Team method lends itself well to the solution of combinatorial problems.

In the next section we describe the problems in detail with examples. In Section 3, we give a brief description of the A-Team technique. Sections 4 and 5 outline the A-Team for MDPS and MCPS respectively, with detailed algorithms. In Section 6, we discuss issues related to the implementation. Computational results are given and discussed in Section 7, and closing remarks are stated in Section 8.

2. Probe selection problems

The analysis of microbial communities gives rise to interesting combinatorial problems. Given a population $\mathcal{C}$ of clones, to analyze $\mathcal{C}$ we need to choose a set S of oligonucleotide probes of a given length l from a large set of candidate probes. Clones and probes are represented as sequences over the alphabet {A,C,G,T}.

A probe p is said to *distinguish* a pair of clones c and d if p is a substring of exactly one of c or d. In some applications clones have length approximately 1500 and probes have length between 6 and 10. In a hybridization experiment the fluorescence response is linear with respect to the number of occurrences of the probe in a clone up to a certain threshold R. As a result there are different versions of the *distinguishability criteria*. We will consider two cases: $R = 1$ and $R = 4$. Below we define the S-fingerprint. In the case where $R = 1$, the S-fingerprint will always be binary, while any case where $R \neq 1$ will result in a non-binary S-fingerprint. In this paper, we thus refer to the case where $R = 1$, as binary, and where $R = 4$, as non-binary.

Example 2.1. Let $\mathcal{C} = \{AAACCTGA, AAACATAAA, ACTAACG\}$ and $\mathcal{P} = \{CCT, ACT, AAA, GCTA, ACG\}$. If $R = 1$, then $S = \{CCT, ACT\}$ is a smallest set of probes such that two distinct clones c and d from $\mathcal{C}$ are distinguished by at least one probe in S. That is, probe CCT distinguishes clones AAACCTGA and AAACATAAA, and clones AAACCTGA and ACTAACG, but it does not distinguish clones AAACATAAA and ACTAACG; The probe ACT distinguishes clones AAACATAAA and ACTAACG.

If $R = 4$, then $S = \{AAA\}$ is a smallest set of probes that distinguishes any two distinct clones in $\mathcal{C}$.

Let $occ(p, c)$ denote the number of occurrences of probe p in clone c. Given a finite set S of probes, the S-fingerprint of clone c, denoted by $fingerprint_S(c)$, is the vector of values $\min\{R, occ(p, c)\}$ over all $p \in S$.

Example 2.2. Let $\mathcal{C}$ and $\mathcal{P}$ be as in Example 2.1, and $R = 1$. If $S = \{CCT, AAA\}$, then $occ(CCT, AAACCTGA) = 1, occ(AAA, AAACCTGA) = 1, fingerprint_S(AAACCTGA) = [1, 1]$. Similarly, $fingerprint_S(AAACATAAA) = [0, 1], fingerprint_S(ACTAACG) = [0, 0]$.

A set S of probes *distinguishes* two clones c and d if $fingerprint_S(c) \neq fingerprint_S(d)$. Let $\mathcal{C}^2$ denote the set of all pairs of different clones from $\mathcal{C}$, that is, $\mathcal{C}^2 = \{(c, d) \mid c, d \in \mathcal{C}, c < d\}$, where "$<$" is an arbitrary (e.g., lexicographic) ordering of $\mathcal{C}$. We denote by $\Delta_S \subseteq \mathcal{C}^2$ the set of pairs of clones that are distinguished by set S.

In this paper we study two probe selection problems, namely:

MAXIMUM DISTINGUISHING PROBE SET (MDPS)
Instance: A set $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ of clones, a set $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ of probes, and an integer k.
Solution: A subset $S \subseteq \mathcal{P}$, with $|S| = k$.
Measure: $|\Delta_S|$, to be maximized.

MINIMUM COST PROBE SET (MCPS)
Instance: A set $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ of clones and set $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ of probes.
Solution: A subset $S \subseteq \mathcal{P}$ such that $\Delta_S = \mathcal{C}^2$.
Measure: $|S|$, to be minimized.

Both problems MDPS and MCPS are NP-hard when the length of probes is unbounded [1]. In [1], the authors tackle MCPS using a Lagrangian relaxation-based heuristic, and the MDPS using a Simulated Annealing algorithm.

In this paper we present a technique called A-Team to find near-optimal solutions to MDPS and MCPS. An A-Team is an organization of agents that communicate with each other by means of shared memories. Each agent is a heuristic strategy that can make its own choices about its inputs, scheduling and resource allocation. Shared memories are repositories of solutions generated by agents. This technique was proposed by Souza and Talukdar [3] and has been applied successfully to the Traveling Salesman Problem [3], Flow Shop Scheduling Problem, Job Shop Scheduling Problem, Set Covering Problem, and Point-to-Point Connection Problem [4].

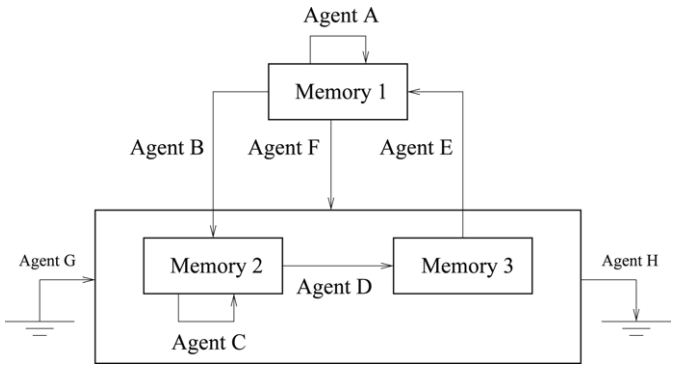

Fig. 1. Example of an A-Team. Arrows represent agents and rectangles represent memories.

3. Asynchronous Team

In this section we give a brief description of what an A-Team is. In Fig. 1 an A-Team is shown, where arrows and rectangles represent, respectively, agents and shared memories. Agents can read from and write to shared memories. Shared memories can contain other shared memories. In Fig. 1 Agent A reads from Memory 1 and writes to Memory 1, Agent B reads from Memory 1 and writes to Memory 2. Agent F reads from Memory 1 and writes to both memories 2 and 3. While Agent G is responsible to fill with solutions the memories 2 and 3, Agent H is responsible for the elimination of solutions from memories 2 and 3.

An A-Team is characterized predominantly by three features:

- **Autonomous agents**: They make their own choice about their input selection, scheduling, and resource allocation policy.
- **Asynchronous communications**: Agents can read and write information in shared memories without any synchronization among them.
- **Cyclic dataflow**: Agents retrieve, modify, and store information continuously in the shared memories.

In Sections 4 and 5, we describe the A-Teams agents that we designed for the MDPS and MCPS problems. In the process of developing the A-Teams approach for both problems, many agents (or algorithms) were developed and tested on a collection of test datasets that did not include any of the data whose results are presented in Section 7. Only those that produced the best results consistently were included in the A-Teams. The test datasets were also used to identify the best parameter settings.

4. A-Team for the MDPS

In this section we design an A-Team to find near-optimal solutions to MDPS instances. Throughout this section the set S is used to represent the solution, where S is a set of k probes that distinguishes a maximum number of clone pairs. We start by defining the heuristic agents (see Fig. 2).

Construction agent C1:

This agent generates a feasible solution to an MDPS instance by selecting a random k-subset S from $\mathcal{P}$.

1. Create a random k-subset S from $\mathcal{P}$
2. Compute S-fingerprint of each clone in $\mathcal{C}$
3. return S, $ObjectiveFunctionValue(S)$.

Algorithms 1–4 detail the functions required for Agent C1. Algorithm 3 gives a naive implementation of the calculation of the objective function value, $|\Delta_S|$, where the S-fingerprints for each pair of distinct clones are compared in $O(m^2k)$ time. In Section 6 we give a $O(mk)$ time algorithm for computing $|\Delta_S|$ [1]. Note that $|\Delta_S|$ represents the number of pairs of clones distinguished by the solution set S.

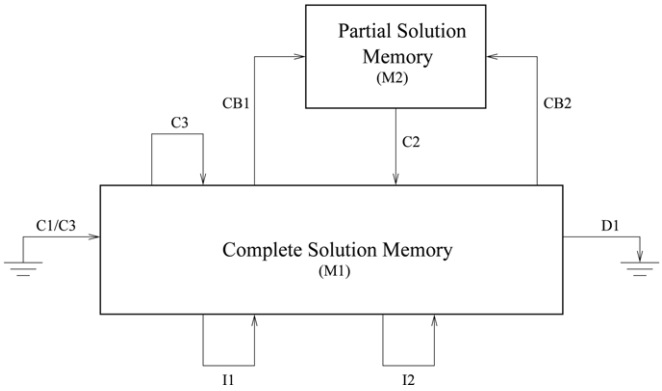

Fig. 2. A-Team for the MDPS: Agents and shared memories.

Input: $\mathcal{P} = \{p_1, \ldots, p_n\}, k$
Output: S
$I \leftarrow \{1, 2, \ldots, n\}$;
$S \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** k **do**
  $j \leftarrow$ randomly choose an element from the index set I;
  $S \leftarrow S \cup \{p_j\}$;
  $I \leftarrow I - \{j\}$;
end
return S

Algorithm 1: Create a random k-subset S from $\mathcal{P}$.

Input: $\mathcal{C} = \{c_1, \ldots, c_m\}, \mathcal{P} = \{p_1, \ldots, p_n\}, S, R$
Output: $fingerprint_S(c)$ for all $c \in \mathcal{C}$
for $i \leftarrow 1$ **to** m **do**
  for *each* $p_j \in S$ **do**
    compute $occ(p_j, c_i)$;
    $fingerprint_S(c_i)[j] \leftarrow \min\{R, occ(p_j, c_i)\}$;
  end
end
return $\{fingerprint_S(c) : c \in \mathcal{C}\}$

Algorithm 2: Compute S-fingerprint of each clone in $\mathcal{C}$.

Input: $\mathcal{C} = \{c_1, \ldots, c_m\}, S$
Output: $|\Delta_S|$
Compute S-fingerprint of each clone in $\mathcal{C}$;
$d \leftarrow 0$;
for $i \leftarrow 1$ **to** $m - 1$ **do**
  for $j \leftarrow i + 1$ **to** m **do**
    if $fingerprint_S(c_i) \neq fingerprint_S(c_j)$ **then**
      $d \leftarrow d + 1$;
    end
  end
end
return d;

Algorithm 3: Simple way to compute $|\Delta_S|$. This algorithm runs in $O(m^2k)$ time.

```
Input: probe p, clone c
Output: number of occurrences of probe p in clone c
t ← 0;
for i ← 1 to length(c) − length(p) + 1 do
    if (p = substring(c, i, i + length(p) − 1)) then
        t ← t + 1;
    end
end
return t;
```

Algorithm 4: Compute $occ(p, c)$. $Substring(c, i, j)$ returns the substring in c that starts at position i and ends at position j.

Consensus-based agent CB1:

This agent works together with agent C2.

1. Randomly select r solutions, say S^i for $i = 1, \ldots, r$, from Memory M1
2. $S' \leftarrow \cap_{i=1}^{r} S^i$
3. return S'.

Consensus-based agent CB2:

This agent works together with agent C2.

1. Randomly select two solutions, say S^1 and S^2, from Memory M1
2. If objective function value of S^1 is better than the objective function value of S^2, then $S' \leftarrow S^1 \setminus S^2$; else $S' \leftarrow S^2 \setminus S^1$
3. return S'.

Construction agent C2:

This agent works together with agents CB1 and CB2.

1. Select a solution, say S^1, from Memory M2
2. $t \leftarrow |S^1|$
3. Randomly select $k - t$ probes from $\mathcal{P}$, say $p'_1, p'_2, \ldots, p'_{(k-t)}$, such that $p'_1, p'_2, \ldots, p'_{(k-t)}$ do not belong to S^1
4. $S \leftarrow S^1 \cup \{p'_1, p'_2, \ldots, p'_{(k-t)}\}$
5. return S, $ObjectiveFunctionValue(S)$.

Construction agent C3:

This agent constructs a feasible solution by selecting probes that distinguish a large fraction of the pairs of clones. The intuition behind this agent is that it is unlikely for probes distinguishing only a small fraction of the pairs of clones to appear in a good solution. Algorithm 5 gives the details for this construction agent.

1. Randomly select a solution, say S^1, from Memory M1
2. $S \leftarrow \emptyset$
3. For each $p \in S^1$, compute the fraction of pairs from $\{fingerprint_{\{p\}}(c) : c \in \mathcal{C}\}$ that are distinct. If this fraction is greater than the parameter *ratio*, then insert probe p into S
4. If $|S| < k$, then randomly select $k - |S|$ probes from the set $\mathcal{P} \setminus S$ and insert them into S.
5. return S, $ObjectiveFunctionValue(S)$.

Input: $\mathcal{C} = \{c_1, \ldots, c_m\}$, $\mathcal{P} = \{p_1, \ldots, p_n\}$, k, R
Output: S, *ObjectiveFunctionValue*(S)

Randomly select a solution, say S^1, from Memory M1;
$S \leftarrow \emptyset$;
$i \leftarrow 0$;
$Q \leftarrow (m^2 - m)/2$;
Stop $\leftarrow$ *false*;
for *probe* $p \in S^1$ **and** *Stop* = *false* **do**
  $t \leftarrow 0$;
  for *each pair of clones* $(c, d) \in \mathcal{C}^2$ **do**
    if $(fingerprint_{\{p\}}(c) \neq fingerprint_{\{p\}}(d))$ **then**
      $t \leftarrow t + 1$;
      if $(\frac{t}{Q} > ratio)$ **then**
        $S \leftarrow S \cup \{p\}$;
        $i \leftarrow i + 1$;
        break;
      end
    end
  end
  if $(i = k)$ **then**
    Stop $\leftarrow$ *true*;
  end
end
if $(i < k)$ **then**
  for $j \leftarrow i + 1$ **to** k **do**
    Randomly select a probe, say p, from the set $\mathcal{P} \setminus S$;
    $S \leftarrow S \cup \{p\}$;
  end
end
return S, *ObjectiveFunctionValue*(S);

Algorithm 5: Detailed algorithm for Agent C3.

In the MDPS each feasible solution is a set of k probes. We say that two sets of probes are t-neighbors if they can be obtained from each other by substituting exactly t of the probes, for $t = 1, \ldots, k$. One can use the idea of generating neighbors of a given solution to try to improve a solution. The next two agents, I1 and I2, are based on this approach.

Improvement agent I1:

1. Randomly select a solution, say S^1, from Memory M1
2. Randomly generate a 1-neighbor of S^1, say S. If S is better than S^1, then replace S^1 by S and stop. If S is not better than S^1, then repeat the process until either a better solution is produced, or the maximum number of allowed attempts (input by the user) has been reached.
3. return S, *ObjectiveFunctionValue*(S).

Improvement agent I2:

1. Randomly select a solution, say S^1, from Memory M1
2. Generate a 2-neighbor of S^1, say S. If S is better than S^1, then replace S^1 by S and stop. If S is not better than S^1, then repeat the process until either a better solution is produced, or the maximum number of allowed attempts (input by the user) has been reached.
3. return S, *ObjectiveFunctionValue*(S).

Destructor agent D1:

This agent deletes the worst solution in the Memory M1.

Once the initial set of feasible solutions is created by Agents C1 or C3 and saved to Memory M1, the remaining agents work to improve the set of solutions. If an agent generates a candidate solution, the objective function value of the solution is calculated and used to determine whether the solution will be added to M1. Suppose that the objective function value of the new candidate solution is T and the best objective function value of the solutions in M1 is Y, then only if $|Y - T|/Y \leq$ `error tolerance` will the new solution be added to M1. If the solution is added, then Agent D1 is called to delete the worst solution from M1, thus the number of solutions in M1 remains constant.

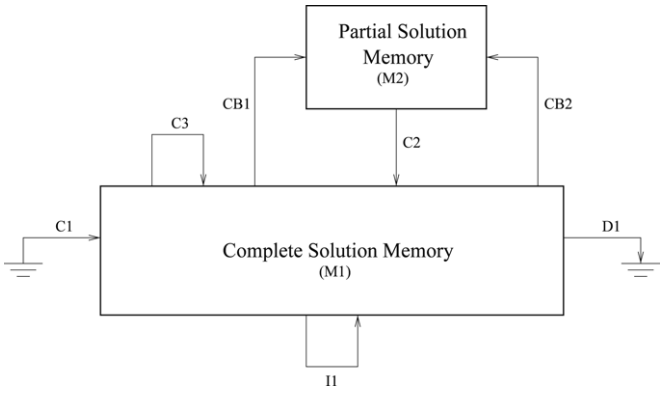

Fig. 3. A-Team for the MCPS: Agents and shared memories.

5. A-Team for the MCPS

In this section we describe an A-Team for the MCPS. As noticed in [1], this problem is a special case of the well-known *Set Covering Problem*, where the universe to be covered is $\mathcal{C}^2$ and the covering sets are the various Δ_p, where Δ_p is defined as follows for a given probe p (see Fig. 3).

Using the notation introduced in Section 2 for Δ_S, we let Δ_{p_j} be the set of pairs of clones in $\mathcal{C}^2$ that are distinguished by the single probe $p_j \in \mathcal{P}$. Algorithm 7 describes in detail how to compute Δ_{p_j}.

To make things clear we define formally the Set Covering Problem.

Set Covering Problem

Instance: A ground set $E = \{e_1, \ldots, e_m\}$, subsets $S_1, \ldots, S_n \subseteq E$, and cost w_j for each subset S_j.

Objective: Find a set $I \subseteq \{1, \ldots, n\}$ such that $\cup_{j \in I} S_j = E$ and $\sum_{j \in I} w_j$ is minimum.

Casting the MCPS as the Set Covering Problem we need to set $E = \mathcal{C}^2$, $S_j = \Delta_{p_j}$, where p_j is a probe in $\mathcal{P}$, and $w_{p_j} = 1$ for each probe $p_j \in \mathcal{P}$.

Construction agent C1:

The idea of this agent is to go through the sets Δ_{p_j} and pick the one that will distinguish the largest number of pairs of clones yet to be distinguished. The algorithm details are given in Algorithm 6. Algorithm 7 describes how Δ_p is found for a given individual probe.

Input: $\mathcal{C}^2$, $\Delta_{p_1}, \ldots, \Delta_{p_n}$, and set of probes S
Output: Probe set S
for $j = 1$ **to** n **do**
  $\tilde{\Delta}_{p_j} \leftarrow \Delta_{p_j}$;
end
while $\Delta_S \neq \mathcal{C}^2$ **do**
  $L \leftarrow Argmax_{p_j : \tilde{\Delta}_{p_j} \neq \emptyset} |\tilde{\Delta}_{p_j}|$;
  $p_i \leftarrow random(L)$;
  $S \leftarrow S \cup \{p_i\}$;
  for $j = 1$ **to** n **do**
    $\tilde{\Delta}_{p_j} \leftarrow \tilde{\Delta}_{p_j} \setminus \Delta_{p_i}$;
  end
end

Algorithm 6: Agent C1 (greedy algorithm). Function *Argmax* returns a set of probes p_j such that $|\tilde{\Delta}_{p_j}|$ is maximized. Function *random* randomly selects a probe from the set L.

Input: $\mathcal{C} = \{c_1, \ldots, c_m\}$, probe p
Output: Δ_p (i.e., set of pairs of clones in $\mathcal{C}^2$ that are distinguished by probe p)
Compute p-fingerprint of each clone in $\mathcal{C}$;
$\Delta_p \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** $m-1$ **do**
  for $j \leftarrow i+1$ **to** m **do**
    if $fingerprint_p(c_i) \neq fingerprint_p(c_j)$ **then**
      $\Delta_p \leftarrow \Delta_p \cup \{(c_i, c_j)\}$;
    end
  end
end
return Δ_p;

Algorithm 7: Compute Δ_p for a given probe $p \in \mathcal{P}$.

Consensus-based agent CB1:
This agent works together with agent C2.

1. Randomly select r solutions, say S^i for $i = 1, \ldots, r$, from Memory M1
2. $S' \leftarrow \cap_{i=1}^{r} S^i$
3. return S'.

Consensus-based agent CB2:
This agent works together with agent C2.

1. Randomly select two solutions, say S^1 and S^2, from Memory M1
2. If $|S^1| > |S^2|$, then $S' \leftarrow S^1 \setminus S^2$; else $S' \leftarrow S^2 \setminus S^1$
3. return S'.

Construction agent C2:
This agent works together with agents CB1 and CB2.

1. Select a solution, say S^1, from Memory M2
2. Call agent C1 with the input parameter $S = S^1$.

Construction agent C3:
This agent is a variant of agent C3 given in the previous section.

1. Randomly select a solution, say S^1, from Memory M1
2. $S' \leftarrow \emptyset$
3. For each $p \in S^1$, compute the fraction of pairs from $\{fingerprint_{\{p\}}(c) : c \in \mathcal{C}\}$ that are distinct. If this fraction is greater than the parameter *ratio*, then insert probe p into S'
4. Call agent C1 with the input parameter $S = S'$.

Improvement agent I1:
Given a feasible solution S for the MCPS, define a 2, 1-exchange as follows: for all pairs of probes in S, if possible, exchange two probes with one probe not in S.

1. Randomly select a solution, say S^1, from Memory M1
2. Apply the 2,1-exchange in S^1 and generate the feasible solution S.
3. return S, $ObjectiveFunctionValue(S)$.

Destructor agent D1:
This agent deletes the worst solution in the Memory M1.

6. Implementation issues

In this section we discuss specific issues pertaining to the implementation of the A-Teams for the MDPS and MCPS. Since the input files often contain a large number of probes and clones, we devoted a considerable amount of attention to the speed and efficiency of the implementation. Three algorithms were of particular concern as potential bottlenecks in the process. The algorithms are, the calculation of $|\Delta_S|$, the Agent C3 in the MDPS A-Team, and the Agent C1 in the MCPS A-Team.

In [1] a fast approach to compute $|\Delta_S|$ was proposed. It is as follows: for each fingerprint f compute the number γ_f of clones with fingerprint f. It can be shown that $|\Delta_S| = \frac{1}{2}\sum_f \gamma_f(m - \gamma_f)$. To find γ_f, sort the clones using $\mathit{fingerprint}_S(c)$ as the key. Then, check consecutive clones in order to get the values γ_f. Since each component of $\mathit{fingerprint}_S(c)$ is an integer less than or equal to R, it is possible to compute $|\Delta_S|$ in $O(mk)$ time by using the radix sort algorithm.

In our implementation of $|\Delta_S|$ we use the function qsort that implements the quicksort algorithm. This function is appropriate to sort large arrays. The computational experiments showed the drastic difference between using the naive implementation of $|\Delta_S|$ and the one explained above.

We found that when large datafiles containing several thousand clones and probes were processed, the Agent C3 in the MDPS A-Team used a significant amount of CPU time. The bottleneck in Agent C3 is in the loop that calculates the number of clone pairs that are distinguished by each probe in a given solution. The loop runs approximately $|C|^2 * |S'|$ iterations per call, where $|S'|$ is equal to the number of probes in the solution.

To speed up the process, we developed a multithreaded preprocessing program in C++ to calculate the fingerprints and number of clone pairs distinguished by each probe. A thread is a path of execution within an instance of a program. Generally a program will have one thread (the primary thread) which terminates when the program terminates. More threads can be created to run multiple paths of execution in parallel. The preprocessing program uses ten threads to perform the calculations for any given pair of clone/probe datafiles with a selected R-value of 1 or 4. Employing multiple threads in the preprocessing program significantly reduced the CPU time required to process files. The output of the program is a file listing the probes and the number of clone pairs distinguished by each probe. The use of preprocessed files significantly increased the speed of Agent C3 by eliminating the need to run through the bottleneck loop.

The Agent C1 in the MCPS A-Team is responsible for the generation of initial solutions in Memory M1. To generate a new solution, Agent C1 chooses the probes that distinguish the greatest number of pairs of clones. As it builds a solution, each time a new probe is added it determines the set of pairs of clones not yet distinguished by the solution. It then chooses the probe that distinguishes the greatest number of pairs of clones in that set. The process continues until the desired number of pairs of clones is distinguished.

The Agent C1 in the MCPS A-Team combines all of the issues that were a concern in the first two cases discussed. The value of $|\Delta_S|$ must be recalculated every time a probe is added to determine if the solution is acceptable, and the number of pairs of clones distinguished by the probes must be repeatedly calculated while the set of clones considered constantly changes as pairs that are distinguished are eliminated. It was necessary to take several measures to speed up Agent C1. The more efficient implementation of the calculation of $|\Delta_S|$ was used to find $|\Delta_S|$. The preprocessed probe files were sorted using the qsort algorithm according to the number of pairs of clones they distinguish. The sorting of the probes eliminated the need for Agent C1 to search for the initial probe. To avoid recalculating the number of pairs of clones distinguished for the entire set of probes (which could easily number more than 4000), only subsets of the probes were worked with at one time. The subset was selected from the top of the sorted list of probes to ensure that the best candidates were included. The size of the subset, which is generally between twenty and fifty, is input by the user. These improvements all worked together to allow Agent C1 to run very efficiently. A simple implementation of the agent would have been infeasible to use.

A dialog-based framework was used to allow for rapid development of the program. Classes from the Microsoft Foundation Class (MFC) library [2] were utilized for storage of probes, clones and solutions. Individual probes were stored as CStrings. CStringArrays were used to store input clones, probes and probe solutions, and a CObArray was used for Memory M1. A random-number generator was used to randomly select the agent to be called at each iteration.

In Section 7.1, we discuss the fact that none of the probe files we had were capable of distinguishing every pair of clones in a clone file. Since it was not possible to distinguish 100% of the pairs of clones, we added a parameter for the minimum percentage of pairs to be distinguished. The user enters this value in the MCPS configuration dialog.

Table 1
Data from [1]

Probe file (no.)	Clone file (no.)	Max possible $\lvert\Delta_S\rvert$
candprobes.a.6 (3872)	dataset1.clones (1158)	669,309
candprobes.a.7 (6241)	dataset1.clones (1158)	669,309
candprobes.a.8 (4209)	dataset1.clones (1158)	669,309
candprobes.a.9 (4581)	dataset1.clones (1158)	669,307
candprobes.a.10 (4209)	dataset1.clones (1158)	669,305
eubacteria5k.a.6 (4064)	eubacteria2k.clones (2000)	1,997,759
eubacteria5k.a.6 (4064)	eubacteria5k.clones (5000)	12,494,429

Table 2
Semi-random data

Probe length (no.)	Clone length (no.)	Max possible $\lvert\Delta_S\rvert$
5 (1009)	1200 (1200)	718,801
6 (4000)	1500 (1200)	718,801
7 (4000)	1500 (1200)	718,801
8 (4000)	2000 (1200)	718,801
9 (4000)	2000 (1200)	718,801
10 (4000)	2000 (1200)	718,801
5,6,7 (4200)	1500 (1200)	718,801
6,7,8,9 (4400)	2000 (2000)	1,998,001
5,6,7,8,9,10 (4200)	2000 (1500)	1,123,501

7. Computational experiments

In this section we present the computational results carried out with the A-Teams described in Sections 4 and 5. In the next subsection we describe the instances used in the tests. In Sections 7.2 and 7.3 we present the results.

7.1. Instances and test environment

The algorithm used for random-number generation is an implementation of the multiplicative linear congruential generator [6], with parameters 16,807 (multiplier) and $2^{31} - 1$ (prime number).

All tests were run on a Pentium 4 CPU with speed of 3.0 GHz and 1 GB of RAM under MS Windows XP. All algorithms were implemented in MS Visual C++ 6.0. CPU times were computed using the function `clock`.

We tested the A-Teams approach on two groups of data. One group of datafiles is detailed in Table 1 and was obtained from the authors of [1]. It consists of three different clone files. The first (dataset1), contains 1158 small-subunit ribosomal genes from GenBank (NCBI). The nucleotide sequence of each gene in the file was edited so that it contains the sequence between two highly conserved primers, but not the primer sequences themselves [1]. Conserved sequences are very similar or identical sequences that appear within various species of an organism or within the molecules of a given organism. The other two clone files (eubacteria2k and eubacteria5k) contain 2000 and 5000 eubacteria samples respectively. In Table 1, the last extension of the probe filename indicates the length of the probes contained in the file.

The second group is semi-random data that we generated using a C++ program we developed. The program randomly generates probes of any length input by the user. It then randomly generates clones with probes from the probe file embedded as substrings. The semi-random datasets are listed in Table 2.

None of the probe sets were able to distinguish every pair of clones in the clone files. To verify this, we calculated $\lvert\Delta_S\rvert$ for each clone/probe file pair with the entire probe set as the solution. The values for each dataset are listed in the last column of Tables 1 and 2.

To date, the only published paper specifically discussing the MDPS and MCPS problems that we are aware of is Borneman et al. [1]. While we did not acquire the results for all of the datasets and tests discussed in their paper, we did have access to a limited number of the results for the MDPS problem for dataset1 and the eubacteria5k dataset.

Table 3
MDPS results (binary distinguishability) on data from [1] after twenty test runs

File	Max $\|\Delta_S\|$	%	Min $\|\Delta_S\|$	Mean	Std Dev	Avg cpu (s)
candprobes.a.6	669,023	99.96	668,882	668,949.42	41.76	130.98
candprobes.a.7	668,777	99.92	668,311	668,507.20	111.24	132.15
candprobes.a.8	668,503	99.88	667,964	668,215.50	153.42	133.69
candprobes.a.9	668,395	99.86	667,458	667,984.00	240.59	132.87
candprobes.a.10	668,903	99.94	667,135	667,948.60	548.6	133.29
Eubacteria (2k)	1,995,476	99.89	1,994,889	1,995,070.60	268.74	232.10
Eubacteria (5k)	12,486,425	99.94	12,484,723	12,484,740.65	938.84	582.46

The % column gives the percent of the maximum $|\Delta_S|$ to the best possible $|\Delta_S|$ that could be obtained for the given dataset.

We compare our results for MDPS in Section 7.2. We did not have results for comparison for MCPS. In Section 7.3, we compare our results for the MCPS problem with those given in Table 1 of [1].

To further measure the performance of the A-Teams approach for the MDPS problem, we compared the best $|\Delta_S|$ obtained from twenty test runs with solutions containing exactly twenty probes to the value of $|\Delta_S|$ obtained when all candidate probes were included in the solution. That is, we compared the results with twenty probes to the best possible result that could be obtained for each given dataset. To evaluate our approach for the MCPS problem, we had the program determine the minimum solution set size that could achieve a minimum cover of 95%. We used the same approach for both the data from [1] and the semi-random data.

In [7], Rash and Gusfield comment that a weakness of the work in [1] is that they only consider probes of a fixed length. Our approach is not limited to a fixed probe length. To address this question, we created three semi-random datasets with probes of varying lengths. The datasets are referenced in Tables 2, 4 and 6 as 5,6,7 and 6,7,8,9 and 5,6,7,8,9,10, indicating the varying lengths of the probes. Each dataset contains an equal number of probes of each length.

Each probe file was run through the Preprocessing program discussed in Section 6 with the associated clone file one time before being used in the A-Team algorithms. The Preprocessing program calculates the number of pairs of clones distinguished by each probe and writes the value to the probe file. The cpu times for preprocessing ranged from 52 s to 380 s.

7.2. Experimental analysis for the MDPS

All of the datasets were tested under the same conditions with the same input parameters for the MDPS problem. The input parameters were selected by performing preliminary tests to determine those that would produce the best overall results.

The input parameters are defined as follows. The number of solutions in M1 determines the constant number of solutions maintained in memory M1. The number of probes per solution is the number of probes each solution in M1 must contain. The error tolerance is defined at the end of Section 4. An iteration refers to a call to a single agent. At each iteration, an agent is randomly selected and called once. The ratio for agent C3 is the minimum fraction of pairs that must be distinguished by any probe included in the solution. The distinguishability criteria is defined at the beginning of Section 2. The CB1 parameter refers to the number of solutions randomly selected in agent CB1. Finally, the number of improvements allowed for agents I1 and I2 defines the number of times the algorithms are allowed to attempt improvement of solutions during a single iteration.

The input parameters were: eight solutions in M1; twenty probes per solution; 5% error tolerance; 300 iterations; ratio for agent C3 equal to 0.80; distinguishability criterion (R) equal to one (binary distinguishability); CB1 parameter equal to two; agents I1 and I2 allowed ten improvements each; and all agents used.

Twenty tests were run for each dataset and the results are reported in Tables 3 and 4. The Max $|\Delta_S|$ and Min $|\Delta_S|$ columns give the maximum and minimum $|\Delta_S|$ value obtained over the twenty test runs, these correspond to the best and worst solutions respectively. The % column gives the percent of the maximum $|\Delta_S|$ to the best possible $|\Delta_S|$ that could be obtained for the given dataset. The Mean, Std Dev and Avg cpu columns give the mean and standard deviation of $|\Delta_S|$ and the average cpu time for all twenty test runs.

The results for the MDPS problem performed on the data from Borneman et al. [1] are given in Table 3. The results are very good. In the worst case, after only 300 iterations and little cpu time, solutions containing twenty probes were

Table 4
MDPS results (binary distinguishability) on semi-random data after twenty test runs

File	Max $\|\Delta_S\|$	%	Min $\|\Delta_S\|$	Mean	Std Dev	Avg cpu (s)
5	718,801	100.00	718,799	718,800.50	0.60	109.86
6	718,801	100.00	718,797	718,799.50	1.00	129.73
7	718,752	99.99	718,438	718,639.20	89.26	136.74
8	718,477	99.95	714,888	717,330.00	973.76	168.20
9	718,168	99.91	712,699	715,685.60	1303.74	171.29
10	717,719	99.85	707,790	714,095.00	2825.12	171.22
5,6,7	718,795	99.999	718,776	718,789.75	4.60	136.11
6,7,8,9	1,997,997	99.999	1,997,914	1,997,980.00	18.81	285.18
5,6,7,8,9,10	1,123,494	99.999	1,123,438	1,123,479.00	15.04	210.81

The % column gives the percent of the maximum $|\Delta_S|$ to the best possible $|\Delta_S|$ that could be obtained for the given dataset.

Table 5
MCPS results (binary distinguishability) on data from [1] after twenty test runs

File	Min $\|S\|$	Max $\|S\|$	Mean	Std Dev	Avg cpu (s)
candprobes.a.6	5	7	6	0.32	43.19
candprobes.a.7	5	6	5.95	0.22	50.27
candprobes.a.8	6	7	6.65	0.49	51.73
candprobes.a.9	6	12	8.75	2.00	21.83
candprobes.a.10	6	8	7.00	0.33	56.89
Eubacteria (2k)	6	7	6.60	0.50	146.27
Eubacteria (5k)	6	6	6.00	0.00	987.50

Table 6
MCPS results (binary distinguishability) on semi-random data after twenty test runs

File	Min $\|S\|$	Max $\|S\|$	Mean	Std Dev	Avg cpu (s)
5	7	7	7.00	0.00	37.35
6	5	6	5.95	0.22	104.22
7	6	6	6.00	0.00	71.39
8	5	5	5.00	0.00	91.18
9	5	5	5.00	0.00	86.94
10	5	5	5.00	0.00	97.88
5,6,7	6	6	6.00	0.00	102.02
6,7,8,9	5	5	5.00	0.00	250.27
5,6,7,8,9,10	6	6	6.00	0.00	95.15

able to distinguish at least 99.86% of the clone pairs distinguished when using all of the candidate probes. It is clear from the standard deviation that the results were consistent over the twenty test runs. The cpu times are most affected by the number of clones in the dataset. The times are consistent among datasets with the same number of clones.

In [1], the authors note that they had significantly better results with non-binary distinguishability. We found the same to be true with the A-Teams for both the MDPS and MCPS problems for all but probes of length 10. As noted, the results shown in Tables 3–6 are all for binary distinguishability.

As discussed in Section 6, we used the fast implementation to calculate $|\Delta_S|$. To ensure that we were comparing results accurately, we ran solutions from [1] through our code to calculate $|\Delta_S|$. We then compared our results using those values. For solutions containing twenty probes of length 6, we obtained better results for non-binary distinguishability in just 350 iterations and 152.6 s of cpu time. Our binary results were within 0.04% of theirs after 500 iterations and 218.9 s. We also compared solutions of twenty probes of lengths 8 and 10 with both binary and non-binary distinguishability. We found that after 350–500 iterations with cpu times of 174–215 s, we obtained results with $|\Delta_S|$ within 0.05% of their $|\Delta_S|$ values, with the exception of the non-binary case for probes of length 10 which was within 0.12%.

The results were excellent for the semi-random data. Solution sets of twenty probes were able to distinguish as many pairs of clones as the entire candidate probe set for probes of length 5 and 6. The probes of length 7 and combined lengths distinguished at least 99.99% of the number of pairs that would be distinguished if the entire candidate set of probes were used. Note that the standard deviation for the probes of length 7 through 10 steadily increased. We observed that the algorithm was still improving results at the 300th iteration in these cases, that is, the improvement had not peaked and leveled off. This resulted in an increase in variability.

We noted that for candidate probe sets containing mixed length probes, the A-Teams for MCPS tended to select more probes of length 6 in each case. In fact, only probes of length 6 were selected for solutions for the length 5 through 10 dataset. The A-Teams for MDPS however selected probes of multiple lengths, and in the length 5 through 10 dataset it found solutions that contained at least one probe of each length.

7.3. Experimental analysis for the MCPS

All of the datasets were tested under the same conditions with the same input parameters for the MCPS problem. The input parameters were selected by performing preliminary tests to determine those that would produce the best overall results.

The input parameters, with the exception of the minimum cover, are defined in the first paragraph of Section 7.2. The minimum cover refers to the minimum percent of clone pairs to be distinguished by each solution.

The input parameters were: eight solutions in M1; 95% minimum cover; 10 iterations; ratio for agent C3 equal to 0.45; distinguishability criterion (R) equal to one (binary distinguishability); CB1 parameter equal to two and all agents used. Twenty tests were run for each dataset and the results are reported in Tables 5 and 6.

In [1], a table is given listing results for optimal and near-optimal probe sets for two of the datasets, dataset 1 and dataset 2. An optimal solution was found with probes of length 5. The authors did not specify what they meant by "near-optimal", so here we report solution sizes ($|S|$) for sets that distinguish at least 99.9% of clone pairs that can be distinguished using the entire set. We had access to probes of length 6 and 8 for dataset 1, but we did not have the probes of length 5. For probes of length 6, we found solutions of size 35 for binary and 22 for non-binary distinguishability; for probes of length 8 we found solutions of size 48 and 20 for binary and non-binary respectively. These are smaller than the values reported in Table 1 of [1], however we must stress that we do not know the criteria used to determine if a solution was near-optimal in the other paper.

The A-Teams for MCPS performed very well on the semi-random data. It was able to distinguish at least 95% of all clone pairs with solutions containing only five to six probes for probes of length 6 and up. Seven probes were required for probes of length 5. Note that the results were very consistent. The only dataset with a standard deviation greater than zero was that with probes of length 6, and the standard deviation was only 0.22. The run times in all cases were short, ranging from 37 to 250 s.

8. Closing remarks

In this paper we have proposed Asynchronous Teams to find near-optimal solutions to probe selection problems. The A-Teams were implemented and tested on both real and semi-random data with probes of both fixed and varying lengths. The computational experiments showed that our approach is extremely effective and can find results comparable to those in [1] in very little computational time. Our approach has some specific advantages over [1]. One advantage is that the A-Teams is able to find near-optimal probe sets containing probes of both fixed and varying lengths. Another is that the A-Teams approach is dynamic in the sense that new heuristic agents can be added at any time without requiring any significant changes to the existing code. This allows for continued improvement of the method with the seamless incorporation of new heuristic algorithms.

Acknowledgments

We gratefully acknowledge the comments provided by the referees which resulted in significant improvements to the paper. We would like to thank Gianluca Della Vedova for providing us with data and results.

First author was supported in part by the Brazilian Federal Agency for Higher Education (CAPES) — Grant No. 1797-99-9.

References

[1] J. Borneman, M. Chroback, G.D. Vedova, A. Figueroa, T. Jiang, Probe selection algorithms with applications in the analysis of microbial communities, Bioinformatics 17 (2001) S39–S48.
[2] Microsoft Corporation, Visual C++ libraries reference, in: Microsoft Developer Network (MSDN), Microsoft Corporation, 2005.
[3] P.S. de Souza, S.N. Talukdar, Asynchronous organizations for multialgorithm problems, in: ACM Symposium on Applied Computing, Indianapolis, 1993.
[4] F.C. Gomes, C.N. Meneses, A.G. Lima, C.A.S. Oliveira, Asynchronous organizations for solving the point-to-point connection problem, in: Proceedings of International Conference on Multi-Agent Systems, ICMAS-98, IEEE Computer Society, 1998.
[5] US Dept. of Energy Office of Science. Genomics GTL, Technical Report, Web page: http://doegenomestolife.org/program/goal3.shtml, 2003.
[6] S. Park, K. Miller, Random number generators: Good ones are hard to find, Communications of the ACM 31 (1988) 1192–1201.
[7] S. Rash, D. Gusfield, String barcoding: Uncovering optimal virus signatures, in: G. Meyers, S. Istrail, S. Hannenballi, P. Perzner, M. Waterman (Eds.), Proceedings of the Sixth Annual International Conference on Computational Biology, 2002, pp. 254–261.
[8] L. Valinsky, G.D. Vedova, A.J. Scupham, S. Alvey, A. Figueroa, B. Yin, R.J. Hartin, M. Chrobak, D.E. Crowley, T. Jiang, J. Borneman, Analysis of bacterial community composition by oligonucleotide fingerprinting of rRNA genes, Applied and Environmental Microbiology (2002) 3243–3250.