Journal of Computer and System Sciences 77 (2011) 122–128

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

# Heuristic algorithms in computational molecular biology

Richard M. Karp [a,b,*]

[a] University of California at Berkeley, United States
[b] International Computer Science Institute, Berkeley, CA, United States

ARTICLE INFO

*Article history:*
Received 11 September 2009
Received in revised form 27 April 2010
Accepted 7 June 2010
Available online 11 June 2010

*Keywords:*
Heuristic algorithm
Implicit hitting set
Multi-genome alignment
Protein interaction

ABSTRACT

In this paper we develop a framework for designing and validating heuristic algorithms for NP-hard problems arising in computational biology and other application areas. We introduce two areas of current research in which we are applying the framework: implicit hitting set problems and analysis of protein–protein interaction networks, with emphasis on a specific problem in each area: multi-genome alignment and colorful connected graph detection.



## 1. Introduction

Combinatorial optimization problems arise frequently in computational molecular biology in connection with such tasks as sequencing genomes and identifying genes and regulatory structures within them, mapping out the regulatory pathways by which proteins combine to perform cellular functions, determining the evolutionary history of organisms, and finding associations between genetic variation and disease. In all these cases, mathematical objective functions and constraints provide an imperfect approximation to the true goal, which is to discover nature's ground truth; for this reason, it is desirable to generate a variety of near-optimal solutions, rather than a single optimal one. Except in rare cases, the problems are NP-hard, and the performance guarantees provided by polynomial-time approximation algorithms are far too pessimistic to be useful. Average-case analysis of algorithms is also of limited use because the spectrum of real-life problem instances is unlikely to be representable by a mathematically tractable probability distribution. Thus it appears necessary to attack these problems using heuristic algorithms. Although we focus here on computational biology, heuristics are also likely to be the method of choice in many other application areas, for reasons similar to those that we have advanced in the case of biology.

In algorithmic research within theoretical computer science, the dominant style is to evaluate an algorithm by its worst-case asymptotic performance over the space of all possible inputs, and to consider all possible algorithms as candidates. By contrast, the successful design of a heuristic algorithm in computational biology often hinges on special properties of the instances that are expected to occur. For example, the success of whole genome shotgun sequencing was based on detailed knowledge of problem data, such as the types of repeated sequences that were expected to occur, the parameters of the technology for generating the input data, and the exclusion of certain hard-to-sequence "heterochromatic" genomic regions. Similarly, boolean satisfiability solvers perform well on many large-scale design automation problems, even though they predictably fail on random instances with a certain critical relationship among the number of variables, the number of clauses, and the number of literals per clause.

Typically, a heuristic algorithm homes in on a particular algorithmic strategy and, within that strategy, strives to tune the algorithm to the expected characteristics of the input data. The algorithmic strategies that interest us most are not

* Address for correspondence: Department of Computer Science, University of California at Berkeley, Berkeley, CA 94720, United States.
*E-mail address:* karp@cs.berkeley.edu.

0022-0000/$ – see front matter 
doi:10.1016/j.jcss.2010.06.009

the standard generic metaheuristics such as greedy methods, divide-and-conquer methods, hill-climbing, local search and genetic algorithms, but rather strategies that are specific to narrower classes of algorithmic problems.

Thus we propose a methodology for heuristic algorithm design with five components: an *algorithmic strategy*, a *training set* of instances, a *verification set* of instances, a systematic *tuning process* and an *evaluation procedure*.

By an algorithmic strategy we mean a high-level algorithmic description, perhaps expressed in pseudocode, which omits certain specific choices, such as the subroutines to be used for specific subtasks and the numerical parameters of the algorithm. The number of concrete algorithms consistent with an algorithmic strategy may be very large or even infinite.

The concept of a training set is borrowed from the field of statistics. We assume that the instances to be solved are drawn from a probability space of instances, and the training set consists of a finite number of instances sampled from that same distribution. We do not require a succinct mathematical description of the probability space; the training set may be derived empirically. For example, when a computational problem is to be solved repeatedly on different biological data sets obtained using the same measurement technology, it may be legitimate to draw the training set from previous samples.

A tuning strategy is a method of searching the space of concrete algorithms consistent with the algorithmic strategy to find the one that performs best on the training set.

Finally, an evaluation method compares the chosen algorithm with its competitors on a verification set of instances drawn from the same distribution as the training set. A second, more expensive, approach to evaluation is to compare the solutions produced by the chosen algorithm against optimal solutions. Because of the typically exponential time to compute an optimal solution, this evaluation may need to be performed on instances scaled down from those in the training set.

In the sections that follow we present two examples of the development of heuristic algorithms. In the first example we derive an algorithmic strategy for a broad class of implicit hitting set problems, and then focus on multi-genome alignment, a specific problem within this class. In the second example, after describing a class of problems involving protein interaction networks, we focus on a particular problem called the colorful connected subgraph problem. Both examples represent ongoing work, and in each case the tuning and evaluation processes are still in progress.

## 2. Implicit hitting set problems

An *independence system* is specified by: a finite *ground set* $U$; for each element $x$ in $U$ a *weight* $w(x)$; and a family of subsets of $U$ called *circuits*, such that no circuit contains another circuit. The *weight* of a subset of $U$ is the sum of the weights of its elements.

A *hitting set* is a set having a nonempty intersection with every circuit. The hitting set problem is to find a hitting set of minimum weight. The hitting set problem is identical to the classic weighted set cover problem, except that the roles of sets and elements have been interchanged. It is NP-hard and cannot be approximated in polynomial time within a factor $(1 - o(1)) \ln n$, where $n$ is the number of circuits, unless NP has quasi-polynomial-time algorithms [5,12]. A simple polynomial-time greedy algorithm achieves the worst-case approximation ratio $\ln n$, but this bound is very pessimistic in practice. In the author's limited experience, the greedy algorithm has always yielded an approximate solution whose weight is within a few percent of optimal.

The hitting set problem can be expressed as a 0–1 integer program, For each element $j \in U$ let $w_j$ be the weight of element $j$ and let $x_j$ be a 0–1 variable indicating whether element $j$ is selected. The integer program is: Minimize $\sum_j w_j x_j$ subject to: for each circuit $S$, $\sum_{j \in S} x_j \geqslant 1$.

The linear programming relaxation, in which $x_j$ is allowed to be a real number in $[0, 1]$, is very efficiently solvable by a variety of methods [16,10].

### 2.1. Implicit hitting set problems

An *implicit hitting set problem* is one in which, for each instance, the set of circuits is not listed explicitly but instead specified implicitly by a *separation oracle*: a polynomial-time algorithm which, given a set $H \subset U$, either certifies that $H$ is a hitting set or returns a circuit that is not hit by $H$.

Each of the following is an implicit hitting set problem:

**Feedback Vertex Set in a Digraph**
Ground set: Set of vertices of digraph $G$.
Circuits: Vertex sets of simple cycles in $G$.

**Feedback Edge Set in a Digraph**
Ground set: Set of edges of digraph $G$.
Circuits: Edge sets of simple cycles in $G$.

**Max Cut**
Ground set: Set of edges of graph $G$.
Circuits: Edge sets of simple odd cycles in $G$.

**k-Matroid Intersection**
Ground set: Common ground set of $k$ matroids.
Circuits: Circuits in the $k$ matroids.

**Maximum Feasible Set of Linear Inequalities [4]**
Ground set: A finite set of linear inequalities.
Circuits: Minimal infeasible subsets of the set of linear inequalities. The existence of a polynomial-time separation oracle follows from the fact that feasibility of a system of linear inequalities can be tested in polynomial time.

**Synchronization in an Acyclic Digraph**
Ground set: A collection $U$ of pairs of vertices drawn from the vertex set of an acyclic digraph G.
Circuit: Minimal collection $C$ of pairs from $U$ with the property that, if each pair in $C$ is contracted to a single vertex, then the resulting digraph contains a cycle.

In one application of this problem, the acyclic digraph represents precedence relations among a set of tasks. Two tasks are *synchronized* if they are executed at the same moment in time. The given pairs of vertices represent pairs of tasks that we wish to synchronize, and the hitting set problem is to delete a minimum-weight set of pairs such that there is a schedule respecting the precedence constraints in which the remaining pairs can be synchronized. An entirely different application, involving synchronization in space rather than time, is discussed in Section 2.3.

### *2.2. A generic algorithm*

We present a generic algorithm for solving instances of the implicit hitting set problem. It assumes a separation oracle, a fast polynomial-time algorithm for the approximate solution of the (explicit) hitting set problem, and an algorithm for the optimal solution of the explicit hitting set problem.

The generic algorithm maintains as dynamic variables $\Gamma$, a set of circuits of the implicit hitting set problem, and $H$, a hitting set for $\Gamma$. The algorithm is as follows:

$\Gamma \leftarrow \phi$
Repeat:
Using the approximation algorithm, construct a hitting set $H$ for $\Gamma$;
Using the separation oracle, attempt to find a circuit that $H$ does not hit;
If a circuit is found
then add that circuit to $\Gamma$
else
$H \leftarrow$ an optimal hitting set for $\Gamma$;
Using the separation oracle, attempt to find a circuit that $H$ does not hit;
if a circuit is found
then add the circuit to $\Gamma$;
else return $H$ and halt

The idea underlying the generic algorithm is to build up a small set of "important" circuits, in the hope that, before too many circuits have been added, the optimal hitting set $H$ for this explicit set of circuits will in fact hit all the circuits of the implicit problem. In this case, $H$ will be optimal for the implicit problem. The approximation algorithm is used as a device for constructing important circuits rapidly, without going to the expense of constructing an optimal hitting set for $\Gamma$. The use of the approximation algorithm is a win/win proposition: it either yields a new circuit or provides an approximate solution to the implicit problem.

Clearly, the effectiveness of the generic algorithm depends on the choice of the separation oracle, the speed of the heuristic algorithm and quality of the approximate solutions it produces, and the speed of the algorithm for computing optimal solutions for the explicit hitting set problems that arise.

In many cases one can design the separation oracle to construct a minimum-cardinality circuit that is not hit by the current set $H$. For example, in the feedback vertex set problem for a digraph $G = (V, E)$, this amounts to finding a shortest cycle in the digraph induced by $V - H$, which is easily done using an all-pairs shortest path algorithm. It is reasonable to expect that a hitting set for the collection of small-cardinality circuits will also be a hitting set for the collection of all circuits. In this case, the generic algorithm will succeed after generating a polynomial-bounded number of circuits. The paper [3] explores this direction.

### *2.3. Multi-genome alignment*

The multi-genome alignment problem is the example that first motivated our study of algorithms for implicit hitting set problems. This problem concerns *anchor pairs*: pairs of substrings from two related genomes that are highly similar, suggesting that both substrings have been derived via mutations from a common ancestral substring. Our goal is to align the genomes against one another so that, to the extent possible, corresponding pairs of anchors are aligned against each other, thus exhibiting the evolutionary relationships among the genomes [7,8].

Formally, we regard each genome as a linearly ordered set of anchors (ignoring the case where two anchors on a genome overlap), and certain anchor pairs are assigned weights indicating the value of synchronizing them (i.e., placing them in the

same column of the alignment). Thus the multi-genome alignment problem is a special case of the previously mentioned problem of Synchronization in an Acyclic Digraph, in which the digraph is a union of disjoint chains, one for each genome.

Using the generic algorithm for the implicit hitting set problem as a framework, Erick Moreno Centeno developed a computer program for the multi-genome alignment problem. The program was presented with 4096 problem instances involving the alignment of up to five worm genomes. It obtained proven optimal solutions to 98 percent of the instances and proven near-optimal solutions to all the remaining instances. The majority of the instances were solved on a laptop within a few seconds each, and the maximum time allowed for an instance was one hour. A paper describing the program and its performance is under preparation [14]. In the course of developing the program we explored a wide range of choices for the major subroutines of the generic algorithm: the separation oracle, the fast approximation algorithm and the exact solution method, and for the conditions under which each of those subroutines is invoked. A major part of our effort was devoted to finding the best combination of choices. Erick Moreno Centeno approached the problem as an optimization problem to be solved by local search. He defined a neighborhood structure over a search space in which each point is a combination of choices for the generator, the approximation algorithm and the exact algorithm. He then used hill-climbing to find a local optimum with respect to this neighborhood structure. On a given instance, the running times for different versions of the algorithm sometimes differed by a factor of one hundred or more. Erick's approach to tuning the algorithm was a model for the evaluation procedure in the present paper's methodology for heuristic algorithm design.

## 3. Analysis of protein–protein interaction networks

A *protein–protein interaction (PPI) network* is an undirected edge-weighted graph in which the vertices are proteins and edges represent physical interactions between two proteins. The evidence for such an interaction may be based on direct physical measurements or by indirect evidence such as similarity of functional annotation or analogy with known interactions between similar proteins. Each edge may carry a weight representing the strength of evidence for the interaction.

PPI networks are available for a handful of species and are continually being refined, but at present the data is noisy. Nevertheless, combinatorial analysis of PPI networks is a promising approach to understanding genetic regulation, since the set of proteins occurring in a molecular machine is expected to induce a connected subgraph rich in edges, and the set of proteins occurring in a signal transduction pathway is expected to include a simple path containing many high-weight edges [6,9,11,15,18–20].

Here is a sample of natural optimization problems related to PPI networks. For simplicity we frame most of these problems in terms of unweighted networks.

*Max clique problem* Find a clique of maximum size. Although this problem is NP-hard and hard to approximate for general graphs, it tends to be easy for PPI networks, which have many vertices of low degree and a small number of hubs of high degree. This is an illustration of the fact that hardness results from complexity theory may not preclude effective solution by heuristics in the case of PPI networks.

*Dense subgraph problem* Among all connected subgraphs with a given number of vertices, find those with a maximum or near-maximum number of edges. Such subgraphs, especially if their vertices have similar functional annotations, provide starting points for more detailed biochemical experimentation to determine if they correspond to molecular machines [21].

*Graph partitioning problem* Find a subgraph with a maximum number of edges, such that no connected component contains more than a given number of vertices. The components of such subgraphs may plausibly represent molecular machines.

*Heavy path problem* Find simple paths of a given length whose total edge weight is maximum or near-maximum. Such paths are plausible candidates for the central spines of signal transduction pathways [17].

*Colorful connected subgraph problem* A PPI network is given together with a set of colors. Associated with each vertex is a (possible empty) set of colors, any one of which may be chosen to label the vertex. The problem is to find a connected subgraph $G = (V, E)$ with a minimum number of vertices, such that there is a an assignment of labels that includes all colors. Here each color represents a type of protein, the colors associated with a vertex indicate the types to which the vertex can belong, and we are seeking a compact subnetwork in which each type of protein is present [1,2].

*Disease signature problem* We are given a PPI network, a set of $n$ *patients* (individuals with the disease under consideration), and, for each patient, a set of proteins that the patient expresses abnormally. For positive integer parameters $t$ and $k$, a *disease signature* is a set $N$ of proteins that is connected in the PPI network, such that, for at least $k$ if the $n$ patients, the number of proteins within $N$ that the patient expresses abnormally is at least $t$. We seek a disease signature of minimum cardinality. Such a signature can be used to screen patients for the disease by observing how many proteins in $N$ they express abnormally. The connectedness requirement ensures that the proteins in $N$ interact cooperatively, making their use for diagnostic purposes more plausible. Mathematically the problem is a set multicover problem with connectedness as an additional constraint [22].

*Causality analysis* We are given a PPI network and a set of ordered pairs of proteins. For each ordered pair $(s, t)$ we have evidence that protein $s$ influences protein $t$ (for example, that a chemical modification in $s$ induces a modification in $t$). The influence may be indirect, through a causal chain of direct influences. To explain the evidence

parsimoniously, we seek to assign directions to a minimum number of the (undirected) edges of the PPI network, indicating putative direct influences, such that, for each given ordered pair $(s, t)$, a directed path from $s$ to $t$ is created [13].

### 3.1. Conservation and functional orthologs

A fundamental tenet of biology is that, once nature develops a mechanism for the survival or propagation of one species, the mechanism will be conserved, with modifications, during the course of evolution. Thus, an important tool for discovering or validating regulatory mechanisms in one species is to find similar mechanisms in related species. Combinatorially, this similarity may be indicated by an isomorphism, or near isomorphism, between subgraphs of the PPI networks of two species, in which the corresponding proteins in the two species are *functional orthologs*; i.e., proteins that are capable of playing similar roles. The evidence for functional orthology can consist of sequence similarity, similarity in structure or domain content, membership in the same protein family, or similarity to a common protein in an ancestral species.

In an ideal case there would exist a bijection between large sets of proteins in species $A$ and species $B$, such that any two corresponding proteins are each other's unique functional orthologs. Given such a bijection $F$, we say that edge $(u, v)$ in species $A$ is *conserved* if edge $(F(u), F(v))$ exists in species $B$.

In practice, some proteins have no functional orthologs, and others have more than one possible functional ortholog. One approach to resolving the ambiguity is to choose unique functional orthologs so as to maximize the number of conserved edges. This leads to the following combinatorial problems.

*Global selection of functional orthologs* Given PPI networks $N$ with vertex set $V$ and $N'$ with vertex set $V'$, and a subset $S$ of $V \times V'$ indicating the possible functional orthologs, select a bijection $F$ that is contained in $S$ and maximizes the number of conserved edges.

*Local selection of functional orthologs* Given $N$. $N'$ and S as above, and a (small) positive integer $t$, find $t$-element sets $A \subseteq N$, and $A' \subseteq N'$ and a bijection $F$ between $A$ and $A'$ that maximizes the number of conserved edges (i.e., the cardinality of the intersection of $S$ with $A \times A'$).

*Query-guided selection of functional orthologs* Given a subgraph $K$ of $N$ representing a molecular complex, find a connected subgraph $K'$ of $N'$ with a minimum number of vertices, such that there is a one-to-one function from the vertex set of $K$ into the vertex set of $K'$, in which each vertex in $K$ is mapped to a potential functional ortholog in $K'$. This is a special case of the colorful connected subgraph problem described above.

For several of these problems we intend to obtain a training set of typical instances, devise an algorithmic strategy and, within the family of concrete heuristic algorithms consistent with this strategy, select one with optimal performance on the training set and evaluate its performance on further instances drawn from the same distribution as the training set.

To illustrate this approach we describe our direction of attack on the colorful connected subgraph problem. References [1,2] motivate the problem and describe our experience with a combination of dynamic programming and integer programming. We are using as a training set a family of query graphs corresponding to known molecular complexes in yeast, where each color indicates the potential orthologs of a protein in the query graph. In this particular case the instances in the training set are easily solvable by visual inspection, given suitable renderings of them by a graph drawing program. Thus the task is to achieve the same performance using a heuristic algorithm.

We have chosen the following framework for our heuristics. Initial testing indicates that, within this framework, we shall find a heuristic that gives optimal solutions to nearly all the instances in the training set.

Let $G = (V, E)$ be the given graph and $C$, the set of colors. For each $v \in V$ let $S(v)$ be the set of colors assignable to $v$. For any set $W \subseteq V$ let $G[W]$ denote the subgraph of $G$ induced by $W$. For any $S \subseteq C$ and $X \subseteq V$, $X$ is called *S-matchable* if there exists a 1–1 function $f$ from $S$ into $X$ such that, for every color $c$ in $S$, $c \in S(f(c))$, and $X$ is *connected* if the induced subgraph $G[X]$ is connected. A connected $C$-matchable set is called *feasible*. Our task is to construct a minimum-cardinality feasible set.

In our algorithmic strategy, each heuristic constructs a sequence of feasible sets $V_1, V_2, \ldots, V_t$ such that $V_1 = V$ and, for $i = 1, 2, \ldots, t-1$, $|V_{i+1}| \leqslant |V_i|$. It is not required that $V_{i+1}$ be a subset of $V_i$; thus a vertex that is deleted in one iteration may be reinserted in a later iteration.

Given $V_i$, an iteration consists of the following steps:

1. Partition the set of colors into two sets, $A$, the infrequent colors, and $B$, the frequent colors.
2. Choose sets $P$ and $Q$ such that $P$ is $A$-matchable, $Q$ is $B$-matchable and $P \cup Q$ is connected.
3. $V_{i+1} \leftarrow P \cup Q$.

To select $P$ let $W = \{v \in V_i \mid S(v) \cap A \neq \phi\}$; Form $G[W]$, the subgraph of $G$ induced by $W$. Delete connected components from $G[W]$ as long as it is possible to do so while maintaining the property of $A$-matchability. Given $P$, choose $Q$ as a minimal $B$-matchable set such that $P \cup Q$ is connected. Optionally, a *pruning step* may be added, in which we start with $G[P \cup Q]$ and repeatedly delete vertices of degree one whose deletion does not destroy feasibility.

```
I N T H I S P A
P E R W E D E V
E L O P A F R A
M E W O R K F O
R D E S I G N I
N G A N D V A L
I D A T I N G H
E U R I S T I C        Frequent letters: I,R,A,N,E
```

**Fig. 1.** The original graph.

```
    T H
    R W
  L O P
M E W O   K F
  D   S   G
  G   N D V A L
  D       G H
  U       I C        Frequent letters: D,G,L,W
```

**Fig. 2.** Subgraph resulting from the first iteration.

```
        P
M E W O   K F
      S   G
      N D V A L
      T       H
  U R I       C
```

**Fig. 3.** The final subgraph.

Thus the main steps within an iteration are to restrict attention to $W$, the set of vertices that can be matched with infrequent colors, delete components of $G[W]$ that are not needed for matching the infrequent colors, minimally reconnect the remaining components while matching the vertices in $B$, and (optionally) delete redundant degree-1 vertices.

Within this framework we consider algorithms for constructing $V_{i+1}$ from $V_i$ by selecting $A, B, P$ and $Q$ and optionally pruning. Here is one example of such an algorithm. Throughout the following process of building up $V_{i+1}$ from $V_i$, whenever a component is to be added to $P$ or a vertex is to be added to $Q$, we break ties in favor of selecting a vertex that does not lie in $V_i$, to avoid repeating the same set $V_i$.

*Selecting infrequent colors* Starting with $A = \phi$ repeatedly augment $A$ by adding a color that increases the number of vertices $v$ such that $S(v) \cap A \neq \phi$ by the smallest amount. Continue as long as this number does not exceed $p|V|$, where $p$ is a parameter of the algorithm. This yields the set $A$, and $B$ is then set to $V - A$.

*Selecting components* We can use a variant of the greedy set cover algorithm. A component of $G[W]$ is *essential* if there is a color that lies in that component and no other component. Initially, let $P$ be the union of the vertex sets of all essential components. Then repeatedly add to $P$ a component which maximizes the ratio of benefit to cost, where the benefit is the increase in the maximum cardinality of a set of colors $S$ such that $P$ is $S$-matchable, and the cost is the increase in the number of vertices in a minimum spanning tree connecting the selected components.

*Adding vertices to create a feasible solution* Starting with $P$ do the following as long as possible: add to $Q$ a vertex which allows an additional color from $B$ to be matched and, among such vertices, minimally increases the number of vertices in a minimum spanning tree of the components of $P \cup Q$. Once $P \cup Q$ is $C$-matchable, add to $Q$ the edges of a minimum spanning tree of the components of $P \cup Q$.

*Pruning* Perform pruning at each iteration.

*Stopping* Stop when $t$ successive iterations fail to decrease the size of $V_i$, where $t$ is a parameter of the algorithm.

In the example of Figs. 1–3, the original graph is an $8 \times 8$ array of cells. Each cell is considered adjacent to its north, south, east and west neighbors. There are 20 colors, represented by Roman letters. Each cell contains a letter indicating its color. A solution is obtained in two iterations.

## Acknowledgments

The ongoing research on the multi-genome alignment is being conducted jointly with Erick Moreno Centeno. Erick's work on tuning algorithms for this problem has greatly influenced my approach to the design of heuristics. The work on PPI networks, and specifically the colorful graph problem, is a continuation of joint work with Falk Hueffner, Sharon Bruckner, Ron Shamir and Roded Sharan reported in [1,2].

## References

[1] S. Bruckner, F. Hueffner, R.M. Karp, R. Shamir, R. Sharan, Topology-free querying of protein interaction networks, in: RECOMB2009, 2009, pp. 74–89.

[2] S. Bruckner, F. Hueffner, R.M. Karp, R. Shamir, R. Sharan, Torque: Topology-free querying of protein interaction networks, Nucl. Acids Res. 37 (2009) 106–108.
[3] K. Chandrasekharan, R.M. Karp, E. Moreno-Centeno, S. Vempala, Algorithms for implicit hitting set problems, manuscript, 2010.
[4] J.W. Chinneck, Fast heuristics for the maximum feasible subsystem problem, INFORMS J. Comput. 13 (3) (2001) 210–223.
[5] U. Feige, A threshold of $\ln n$ for approximating set cover, JACM 45 (4) (1998) 643–652.
[6] J. Flannick, A. Novak, B.S. Srinivasan, H.H. McAdams, S. Batzoglou, Graemlin: General and robust alignment of multiple large interaction networks, Genome Res. 16 (2006) 1169–1181.
[7] J.D. Kececioglu, The maximum weight trace problem in multiple sequence alignment, in: CPM1993, 1993, pp. 106–119.
[8] J.D. Kececioglu, H.-P. Lenhof, K. Mehlhorn, P. Mutzel, K. Reinert, M. Vingron, A polyhedral approach to sequence alignment problems, Discrete Appl. Math. 104 (1–3) (2000) 143–186.
[9] B.P. Kelley, R. Sharan, R.M. Karp, T. Sittler, D.E. Root, B.R. Stockwell, T. Ideker, Conserved pathways within bacteria and yeast as revealed by global protein network alignment, Proc. Natl. Acad. Sci. USA 100 (20) (2003) 11394–11399.
[10] C. Koufogiannakis, N.E. Young, Beating simplex for fractional packing and covering linear programs, in: FOCS, 2007, pp. 494–504.
[11] M. Koyoturk, Y. Kim, U. Topkara, S. Subramanian, W. Szpankowski, A. Grama, Pairwise alignment of protein interaction networks, J. Comput. Biol. 13 (2) (2006) 182–199.
[12] C. Lund, M. Yannakakis, On the hardness of approximating minimization problems, JACM 41 (5) (1994) 960–981.
[13] M. Medvedoskey, V. Bafna, R. Sharan, U. Zwick, An algorithm for orienting graphs based on cause-effect pairs and its application to orienting protein networks, in: WABI, 2008, pp. 222–232.
[14] E. Moreno Centeno, R.M. Karp, Implicit hitting set problems and multi-genome alignment, manuscript, 2010.
[15] M. Narayanan, R.M. Karp, Comparing protein interaction networks via a graph match-and-split algorithm, J. Comput. Biol. 14 (7) (2007) 892–907.
[16] S. Plotkin, D.B. Shmoys, E. Tardos, Fast approximation algorithms for fractional packing and covering problems, Math. Oper. Res. 20 (1995) 495–504.
[17] J. Scott, T. Ideker, R.M. Karp, R. Sharan, Efficient algorithms for detecting signaling pathways in protein interaction networks, J. Comput. Biol. 13 (2) (2006) 133–144.
[18] R. Sharan, T. Ideker, Modeling cellular machinery through biological network comparison, Nature Biotechnol. 24 (4) (2006) 427–433.
[19] S. Suthram, R. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R. Karp, S. Sharan, T. Ideker, Proc. Natl. Acad. Sci. USA 102 (2005) 1974–1979.
[20] R. Singh, J. Xu, B. Berger, Pairwise global alignment of protein interaction networks by matching neighborhood topology, in: LNBI, vol. 4453, Springer, Heidelberg, 2007, pp. 16–31.
[21] D.A. Spielman, S.-H. Teng, A local clustering algorithm for massive graphs and its application to nearly linear-time graph partitioning, in: CoRRabs, arXiv:0809.3232, 2008.
[22] I. Ulitzky, R.M. Karp, R. Shamir, Detecting disease-specific disregulated pathways via analysis of clinical expression profiles, in: RECOMB2008, 2008, pp. 347–359.