Multi-Agent, Multi-Case-Based Reasoning

Susan L. Epstein[1,2], Xi Yun[1], and Lei Xie[1,2]

[1] Department of Computer Science,
The Graduate Center of The City University of New York, New York, NY 10016, USA
[2] Department of Computer Science,
Hunter College of The City University of New York, New York, NY 10065, USA
{susan.epstein,lei.xie}@hunter.cuny.edu, xyun@gc.cuny.edu

Abstract. A new paradigm for case-based reasoning described here assembles a set of cases similar to a new case, solicits the opinions of multiple agents on them, and then combines their output to predict for a new case. We describe the general approach, along with lessons learned and issues identified. One application of the paradigm schedules constraint satisfaction solvers for parallel processing, based on their previous performance in competition, and produces schedules with performance close to that of an oracle. A second application predicts protein-ligand binding, based on an extensive chemical knowledge base and three sophisticated predictors. Despite noisy, biased biological data, the paradigm outperforms its constituent agents on benchmark protein-ligand data, and thereby promises faster, less costly drug discovery.

Keywords: multiple cases, multiple agents, confidence-based reasoning.

1 Introduction

As the problems presented to computers become increasingly difficult, the techniques researchers develop to address them become increasingly sophisticated and complex. Although these programs may perform unevenly, ensembles of them often smooth performance [1]. At the same time, data pertinent to difficult problems has burgeoned, even though it is often noisy and incomplete. Rather than trust the evidence of a single data point, it may be more informative, to consider several. The case-based reasoning paradigm described here, *MAMC* (Multi-Agent Multi-Case-based reasoning), takes both routes: it consults multiple agents and it uses multiple cases. We report here on two MAMC applications: construction of a parallel schedule for constraint satisfaction search, and prediction about the binding energy between two proteins, a key to rational drug design. Although many such *agents* (here, solvers or predictors) exist, none consistently outperforms all the others on a large, diverse set of benchmark examples. The thesis of this paper is that the effectiveness of a set of agents on a set of similar cases supports reasoning about the agents' performance on a new case. Given a new case, MAMC selects from its knowledge base the cases most similar to it, and examines the accuracy of a set of agents on those cases. The principal result reported here is that MAMC improves predictive accuracy in both applications.

S.J. Delany and S. Ontañón (Eds.): ICCBR 2013, LNAI 7969, pp. 74–88, 2013.

Input: new case e, case base C, agents A, pairwise similarity metric s, number of reference cases q
Output: prediction or recommendation for e

Select a subset L of q cases in C most similar to e as measured by s
Predict or recommend on each case in L with A_j for all $A_j \in A$
Combine output from all $A_j \in A$ by their performance on L as output for e

Fig. 1. High-level pseudocode for MAMC

MAMC is outlined in Figure 1. For a particular domain, MAMC's pre-existing agents A are assumed to be the result of extensive research and development, and generally regarded by experts as among the best. The corresponding case base C, shared by all the agents, consists of published results for those agents on their common task. Finally, the features on which similarity is gauged to select the reference cases L are assumed available from other experts' work in the domain of interest, but deliberately differ from those of any individual agent for their prediction.

Unlike earlier work with multiple cases, which drew from case bases for related tasks [2] or focused on distributed resources across multiple machines [3], MAMC reasons over multiple cases resident on the same computer and for the same task. Rather than use portions of multiple cases or multiple agents to produce a solution, as in [4], MAMC uses multiple cases to select one agent most appropriate for a new case. MAMC can also estimate the reliability of its output, an essential but rarely available property in bioinformatics. MAMC's confidence is based not on the quality of its cases, as in [5], but on the degree of similarity it detects between the new case e and the reference cases L, along with the performance of the selected agent on those cases.

The applications described here face similar challenges: development of an extensive case base with an incisive feature-based index, a pairwise similarity metric for cases, and a way to combine the agents' output to make a decision. Each of the next two sections details an application, with relevant background; related work; the origin of C, A, and s; and empirical design and results. The discussion mines this experience to establish commonalities, issues, and promise for future MAMC applications.

2 Parallel Portfolio Construction for Constraint Satisfaction

Constraint satisfaction is a powerful representation for many real-world problems, but search for a solution to a constraint satisfaction problem (*CSP*) is in general NP-hard. Many solvers succeed on quite difficult problems, but unevenly and unpredictably. Our goal is to schedule solvers on multiple processors to solve one problem.

Here, a CSP $<X, D, R>$ is a set X of variables, a set D of finite domains associated with those variables, and a set R of constraints that restrict how values from their respective domains may be assigned to variables simultaneously. A *solution* assigns a value to each variable and satisfies all of R. A *solvable* CSP has at least one solution. The solvers used here assign a value to one variable at a time, and temporarily remove inconsistent values from the domains of as-yet-unassigned variables. If a domain becomes empty, the solver backtracks to the most recent alternative and chooses a new value. Search returns the first solution found, or halts when it shows the problem

is *unsolvable* (i.e., the domain of the variable at the top of the search tree becomes empty). A *variable-ordering* heuristic determines the order in which the solver addresses variables, and a *restart policy* begins search on the problem again, probably with a different root variable. This remainder of this section summarizes work that appeared in the optimization literature, and detailed its rationale and development saga more theoretically [6].

2.1 The Task for MAMC

In this application, the agents A are CSP solvers, C is a set of CSPs, and T is a set of consecutive, unit time intervals. A *simple schedule* σ for a problem on one processor specifies at most one agent to address the problem in each time interval, that is, σ: $T \rightarrow A$. A *schedule* for k processors is a set of k simple schedules, one for each processor. On any one processor, at most T time can be allotted to any solver on any problem. We represent the performance of A on C in a $|C| \times |A|$ *performance matrix* τ, where entry $\tau_{ij} \in \{1,2,\ldots, T\}$ means that the jth solver solves the ith problem in time τ_{ij}; otherwise the problem goes unsolved in time T. The solvers used here are *deterministic,* that is, each τ_{ij} is a fixed, positive integer.

A CSP *portfolio* is a combination of solvers intended to outperform each of its constituents [7-12]. A *solver portfolio* $<A, k, S, T>$ proposes a set S of k simple schedules that deploy agents A on k processors to solve a problem in time T. For $k = 1$, a solver portfolio is *simple;* for $k > 1$, it is *parallel*. If it is deterministic, neither a simple nor a parallel solver portfolio can exceed the performance of an *oracle*, which always selects the single fastest solver. We focus here on *offline* solver portfolio constructors, which observe the performance of A on C and then build a portfolio to optimize performance on new cases [8, 9, 11]. We consider only *switching* schedules, which preserve each solver's intermediate search state when its time elapses, for reuse only by that solver if time is allocated to it on the same processor later.

Given cases C, a new problem e, similarity metric s, and performance matrix τ for solvers A on C, MAMC's task is to find the best parallel schedule σ that uses A to solve e on k processors. The portfolio constructors most relevant here, CPHYDRA [8] and GASS [13], were both intended for a single processor. CPHYDRA is case-based; it selects a small set of problems in C similar to e, and searches all possible schedules, in time $O(2^{|A|})$. It weights problems in C by their Euclidean distance from e, and seeks an optimal schedule, one that maximizes the number of problems solved within T. In contrast, GASS is greedy, and its performance depends on $|C|$. At each step, GASS selects the agent that maximizes the number of problems solved per unit of time, and counts only problems solved for the first time during the current time step. GASS creates schedules in time $O(|C| \cdot |A| \log |C| \cdot \min\{|C|, T \cdot |A|\})$ that are at most four times worse than optimal; any better approximation was shown to be NP-hard [13].

2.2 Cases, Similarity, and Combination

The case base was developed from the 3307 problems in 5 categories at the Third International CSP solver competition (*CPAI'08*), where problems represent a wide

Table 1. CPAI'08 problems by category

Applicable solvers	Category	Competition problems	Experiment problems	Solvable experiment problems
17	GLOBAL	556	493	256
22	k-ARY-INT ($k \geq 2$)	1412	1303	739
23	2-ARY-EXT	635	620	301
24	N-ARY-EXT ($N > 2$)	704	449	156

Table 2. Problems in Table 1 solved by non-parallel solver portfolios in 1800 seconds

Solver	Oracle	GASS	CPHYDRA10	CPHYDRA40
Number solved	2865	2773	2577	2573
% solved	100%	96.79%	89.95%	89.81%

variety of challenges and are intended to be difficult [14]. Cases were represented by the same 36 numeric features (e.g., number of variables, maximum domain size) used by CPHYDRA. To extract feature values, we ran the solver Mistral 1.550 on an 8 GB Mac Pro with a 2.93 GHz Quad-Core Intel Xeon processor. We excluded any problem whose full set of features was not calculated within 1 second, and any problem never solved by any solver at CPAI'08. Table 1 summarizes the remaining 2865 problems.

Agents in A are the 24 solvers in CPAI'08. They include CPHYDRA10 and CPHYDRA40, versions of CPHYDRA that used the same 3 solvers, but with 10 or 40 cases respectively (CPHYDRA won all but one category at CPAI'08.) The CPAI'08 results provide the performance matrix τ. For *neighbor set ratio* r ($0 \leq r \leq 1$), the *neighbor set* L of any case e is the $r \cdot |C|$ problems in C with feature vectors most similar to e.

Portfolio construction experiments performed 10-fold cross-validation. Each iteration partitioned the 2685 into a set of testing problems and a set of training problems (i.e., the case base C). Stratified partitioning maintained the proportions of problems from different categories in each subset. Table 2 reports the performance of an oracle and three non-parallel solver portfolio constructors, given 1800 seconds per problem.

RSR-WG (Retain, Spread and Return with a Weighted Greedy approach) is the MAMC implementation for this task. As in Figure 2, it formulates a parallel schedule for problem e based on τ, A, C, and L, the set of cases most similar to e. RSR-WG tries to build a schedule that solves as many cases as it can from L, under the assumption that the same schedule will then do well on e. RSR-WG tries, heuristically and greedily, to schedule L, and measures the similarity of case $c_i \in L$ to e based on its Euclidean distance $d(c_i, e)$ from e (here, $\varepsilon = 0.001$):

$$s_i = 1 - \frac{(1-\varepsilon)\left[d(c_i, e) - d_{\min}\right]}{d_{\max} - d_{\min}} \tag{1}$$

where $d_{\max} = \max(\{d(c_i, e) \mid c_i \in L\})$ and $d_{\min} = \min(\{d(c_i, e) \mid c_i \in L\})$. Given execution time t for A_j, RSR-WG counts (from performance matrix τ) and weights (with corresponding similarity) how many problems A_j could solve from L in time t:

$$N_j^z(t) = \sum_{x_i \in L} s_i \zeta_{ij}(t) \text{ where } \zeta_{ij}(t) = \begin{cases} 1 & \text{if } \tau_{ij} \le t \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Then, at time t, RSR-WG greedily maximizes (2) per unit of time expended over all solvers and their possible execution duration Δ_z, that is, it calculates:

$$\underset{A_j, \Delta_z}{\operatorname{argmax}} \frac{N_j^z(t+\Delta_z)}{\Delta_z} \quad (3)$$

and removes those now-solved similar problems from L. *Retain* (line 6) places solver A_j on processor π_u if that maximizes equation (2) per unit of expended time and π_u still has time available ($T^u < T$). Among such processors, Retain prefers one that has hosted A_j earlier ($t_{uj} \neq 0$); otherwise it selects one that has thus far been used the least (i.e., has minimum T^u). If a parallel schedule σ solves all of L without making full use of all the processors, *Spread* (line 11) places the solver A_j that solves the most problems in L but does not appear in σ on a processor that was idle throughout σ (if such an A_j exists), breaking ties at random. (The rationale here is that A_j may be generally effective but not outstanding on a particular e.) Finally, if a processor is not fully used in σ (i.e., $T^u < T$), *Return* (line 14) places the first solver it executed there on that processor until the time limit. Obviously, RSR-WG achieves the performance of an oracle when $k = |A|$, but it is also effective when k is relatively small compared to $|A|$.

Input: case base C, solvers A, time limit T, testing problem e, distance function d, similarity function s, neighbor set ratio r, processors $\{\pi_1, \pi_2, \ldots, \pi_k\}$
Output: schedule $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ for a parallel switching portfolio

```
1  Compute distance d(c_i, e) for all c_i in C
2  L ← {100r% of problems in C closest to y}
3  Compute similarity s_i for each c_i in L with equation (1)
4  Initialize time step z ← 1, overall time T^u ← 0 on processor π_u,
   time t_uj ← 0 for A_j on π_u
5  While L ≠ Ø and T^u < T for at least one u
6    Select A_j on π_u with time Δ_z to maximize equation (3)        ** Retain **
7    Remove from L all problems solved by A_j during step z
8    Schedule A_j with execution time Δ_z on π_u
9    Update times: t_uj ← t_uj + Δ_z, T^u ← T^u + Δ_z, and z ← z + 1
10 For each π_u where T^u < T
11   If T^u = 0                                                     ** Spread **
12   then assign A_j to π_u for T, where A_j solves the most problems in L and A_j ∉ σ
13        update times: t_uj ← T, T^u ← T, and z ← z + 1
14   else π_u executes the first solver placed on π_u until T       ** Return **
15        update times: t_uj ← t_uj + (T – T^u), T^u ← T, and z ← z + 1
16 Return σ
```

Fig. 2. High-level pseudocode for RSR-WG

2.3 Experimental Design and Results

The parallel constructors tested here are RSR-WG, PGASS (a naïve parallel version of GASS with uniform weights $s_i = 1$), and PCPHYDRA (a naïve parallel version of CPHYDRA that randomly partitions L into k subsets and then uses CPHYDRA on each subset to construct a schedule for each processor). PCPHYDRA selects $|L| = 10k$ neighbors, randomly distributes them to k processors, and executes a complete search for the optimal schedule on each processor. If it does not produce an optimal schedule within 180 seconds, it takes the best schedule it has found so far. For RSR-WG, we simulated all 24 solvers from the original competition [15].

All portfolio experiments ran on a Dell PowerEdge 1850 cluster with 696 Intel 2.80 GHz Woodcrest processors. We gauged performance as in recent competitions, on the number of problems solved with a fixed, per-problem time limit, with ties broken on average solution time across solved problems [15, 16]. Time for RSR-WG included both portfolio construction (i.e., scheduling) and search, but time for PGASS and PCPHYDRA excluded portfolio construction, which gave them a slight advantage.

As Table 3 shows, for $k > 1$, RSR-WG consistently solved more problems than PGASS or PCPHYDRA. For RSR-WG only, we also tested $k = 16$ processors, which produced near-oracle performance. Although 2 of the 10 runs for $k = 16$ were perfect, this becomes very nearly a race among solvers that did well on the cases in L. The near-optimal performance of $k = 8$, or even $k = 4$, along with the fact that only RSR-WG was charged for scheduling time, is more noteworthy.

Figure 3 compares an oracle's runtime to that of RSR-WG with $r = 0.005$. Each circle represents one of the 2865 problems. Those at the far right are problems unsolved by RSR-WG in 1800 seconds; those on the diagonal were solved by RSR-WG as quickly as an oracle. Clearly, more processors solved more problems (from 2769 to 2859 in this particular run) and solved more problems as quickly as an oracle.

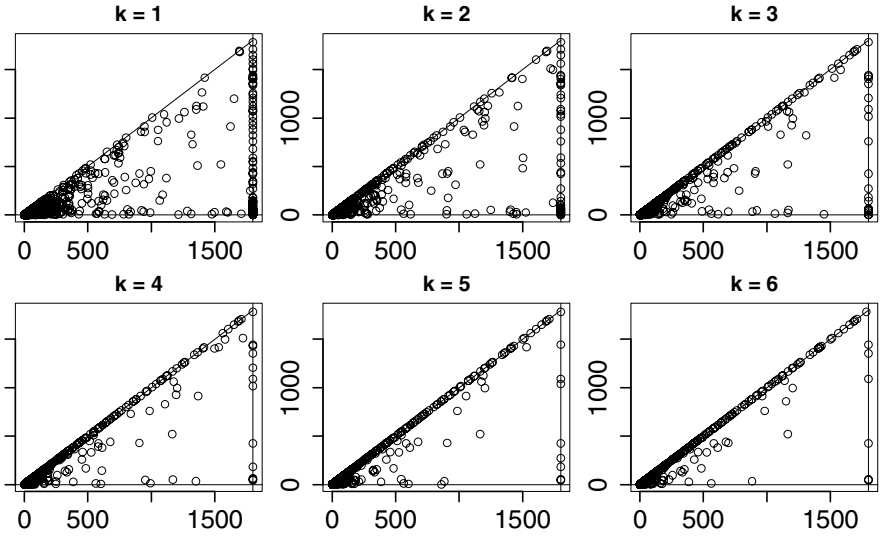

Fig. 3. (Ideal) oracle runtime (*y-axis*) compared to RSR-WG time (*x-axis*) on k processors with neighbor set ratio $r = 0.005$. Each circle is a result on one of the 2865 problems.

Table 3. Mean performance of 3 constructors on 2865 problems over 10 runs, with the (significantly) best value for k processors in boldface. * denotes RSR-WG outperformed PGASS; † denotes RSR-WG outperformed PCPHYDRA ($p < 0.005$). Neighbor set ratio was $r = 0.005$.

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$
PCPHYDRA	2779	2807	2817	2827	2830	2831	2834	2834
PGASS	2771	2801	2808	2810	2817	2821	2823	2825
RSR-WG	2773	**2826***†	**2841***†	**2850***†	**2855***†	**2857***†	**2858***†	**2859***†

MAMC's computational cost for parallel schedule construction is worthwhile, compared to that of other schedulers. Recall that the time allotted to each problem is 1800 seconds. RSR-WG constructs its schedules quickly. For example, it averages less than 15 seconds ($\sigma \approx 6$) with $r = 0.16$ and 8 processors, and is faster for smaller r and k. Recall, however that PCPHYDRA sometimes fails to compute its (optimal) schedule within 180 seconds. Indeed, given its $O(2^{|A|})$ complexity, it produced no schedule at all on 4.81% of the cases when $k = 1$, and 14.39% of the cases when $k = 8$. Because it learns on all the cases, however, GASS was even slower; its single entry in Table 2 required more than 5 days to compute.

3 Protein-Ligand Docking

The central topic of rational drug design is protein-ligand interaction, where a small molecule (a *ligand*) binds to a specific position (e.g., an open cavity) in a protein [17]. Protein-ligand docking (*PLD*) evaluates the ligand's orientations and conformations (three-dimensional coordinates) when bound to a receptor. PLD seeks ligands with the strongest (i.e., minimal) total binding energy in a protein-ligand complex, but most PLD software predicts binding energy poorly. Thus, for reliability, conventional PLD meta-predictors use *consensus scoring*, which averages scores or takes a majority prediction from several predictors [18-20]. Consensus scoring ignores similarities between examples, as well as domain-specific and example-specific data about its individual predictors. Thus it is inaccurate when most of its component predictors are.

In a single docking run, virtual high-throughput screening predicts which of thousands of compounds should be tested in the laboratory [21-23]. Recent approaches tried chaining [24] or bootstrapping with an ensemble based on a single function [25]. The work reported here, however, is the first to combine different PLD predictors based on case similarity plus information from and about individual predictors.

3.1 The Task for MAMC

Here the agents A are three pre-existing PLD scoring functions: eHiTS[1], AutoDock Vina[2], and AutoDock[3]. Although all rely on some form of machine learning, each has

[1] http://www.simbiosys.ca/ehits/ehits_overview.html
[2] http://vina.scripps.edu/
[3] http://autodock.scripps.edu/

its own conformational sampling, scoring, and feature-based representation. They often perform dramatically differently on the same data, with no consistent winner.

A case is the binding energy measured in the laboratory between a given *receptor* (a target in a protein) and a chemical compound. Each compound is a potential ligand, represented by a feature vector that reports chemical properties (e.g., whether it is a hydrogen-bond donor, or whether its topological distance between two atoms lies in some range). These are different features from those used by the agents; the agents consider three-dimensional chemical conformation, while the cases describe physiochemical and topological properties derived from two-dimensional chemical structure. In a case base C, all cases address the same receptor. To describe a case, values for its standard chemical *footprint* of 1024 boolean features were calculated offline with programs such as openbabel[4]. Given a case base C, a new chemical e, chemical-similarity metric s, and performance matrix τ for the agents A on C, MAMC's task is to predict the binding energy between C's receptor and e.

3.2 Cases, Similarity, and Combination

We tested MAMC on datasets from *DUD* (Directory of Useful Decoys), a set of benchmarks for virtual screening [26]. A *decoy* is a molecule similar to its ligand in its physical properties but dissimilar in its topology. DUD has multiple ligands for each receptor, and 36 decoys for each of its ligands. We considered two receptors from DUD: `gpb` and `pdg`. All three agents perform relatively poorly on them. Moreover, eHiTS, is the worst of the three on `gpb` but the best on `pdg`. Together these receptors challenge MAMC to choose the most accurate predictor for each chemical.

The similarity metric s on example e and case $c_i \in C$ is defined by the *Tanimoto coefficient*, the ratio of the number of features present in their intersection to the number of features present in their union, where $N(c)$ is the number of 1's in c:

$$s(e, c_i) = N(e \,\&\, c_i) \,/\, N(e \mid c_i) \tag{4}$$

Each predictor A_j was asked to calculate a score for each ligand and decoy in the dataset. We eliminated the very few chemicals that did not receive three scores; this left 1901 chemicals for `gpb`, and a separate set of 5760 for `pdg`. The agents' incomparable scales, however, required a simple but robust *rank-regression scoring* mechanism to map raw scores uniformly to a normalized rank score that reflects only the preference of an agent for one case over another. For each agent $A_j \in A$, MAMC sorts the raw scores from A_j for all $c_i \in C$ in ascending order, replaces each score with its rank, and normalizes the ranks in [0,1]. The normalized rank, denoted by $p(c_i, A_j)$, predicts the score of A_j on c_i. Higher-ranked cases thereby receive lower scores, in line with the premise that lower binding energy is better.

We again represent the performance of A on C in a $|C| \times |A|$ performance matrix τ. To evaluate the performance $\tau(e, A_j)$ of $A_j \in A$ on e, we use the set of cases L similar to e, but weight more heavily those more similar to it:

[4] `http://openbabel.org/wiki/Main_Page`

$$\tau\left(e,A_j\right)=\sum_{c_i\in L} s(e,c_i)\tau_{ij} \tag{5}$$

Then, to predict on e with all of A, we take the agent A_j with the highest $\tau(e, A_j)$ and combine its predicted scores on L, again weighting more similar cases more heavily:

$$p\left(e\right)=\sum_{c_i\in L} s(e,c_i)p(c_i,A_j) \tag{6}$$

Intuitively, a scoring function that accurately distinguishes ligand set G from decoy set Y (where $Y \cup G = C$) should predict lower scores for ligands and higher scores for decoys. In other words, agent A_j is more accurate on ligand g only if its prediction for g is generally lower than its predictions for Y, and it is more accurate on decoy y only if its prediction for y is generally higher than its predictions for G. The *performance score* of agent A_j on case c is thus

$$\tau\left(c,A_j\right)=\begin{cases}\dfrac{\left|\left\{y\in Y\middle| p\left(y,A_j\right)>p\left(c,A_j\right)\right\}\right|}{\left|\{y\in Y\}\right|} & \text{if } c\in G\\[2ex] \dfrac{\left|\left\{g\in G\middle| p\left(g,A_j\right)<p\left(c,A_j\right)\right\}\right|}{\left|\{g\in G\}\right|} & \text{if } c\in Y\end{cases} \tag{7}$$

Scores in equation (7) lie in [0,1], where a higher value indicates better performance.

3.3 Experimental Design and Results

Each of these experiments predicts the binding energy of a chemical e to a receptor. We examine the accuracy of five predictors: the three individual agents (eHiTS, AutoDock Vina, AutoDock) and two meta-predictors: MAMC and *RankSum,* a typical bioinformatics consensus-scoring meta-predictor. To predict the score on example *e,* RankSum adds the rank-regression scores from the three predictors, where a lower sum is better. In advance, for MAMC, we computed the similarities between all ${}_nC_2$ pairs of chemicals *(*about 1.8 million for `gpb` and 16.6 million for `pdg`) with equation (4), and recorded the five chemicals most similar to each chemical, along with their similarity scores. Experiments ran on an 8 GB Mac Pro with a 2.93 GHz Quad-Core Intel Xeon processor, and analysis used the R package ROCR.

First, we evaluated the three individual predictors with leave-one-out validation: in turn, each of the n chemicals for a receptor served as the testing example *e,* while the other n-1 served as C. MAMC extracted the $|L|$ cases in C most similar to e, and then used equation (5) to gauge the accuracy of each individual predictor across all the cases in L. MAMC then chose the individual predictor with the best predictive accuracy on L and reported as a score the rank-regression score on e from that best individual predictor as in (6).

We compare predictors' performance by their hit ratio across C. ROC (Receiver Operating Characteristic) curves illustrate the tradeoff between true positive and false positive rates, an important factor in the decision to test a likely ligand in the laboratory. (Classification accuracy alone would be less helpful, because the prevalence of so

many decoys heavily biases the data sets. Simple prediction of every chemical as a decoy would be highly accurate but target no chemicals for investigation as likely ligands.) A predictor p on any $c \in C$ produces true positives $C_1 = \{g \in G \mid p(g) \leq p(c)\}$ and false positives $C_2 = \{y \in Y \mid p(y) \leq p(c)\}$. Thus the true positive rate for c is $|C_1|/|G|$ and the false positive rate is $|C_2|/|Y|$.

We report first on $|L| = 1$, using the single case c_i most similar to e. (For $|L| > 1$, see the next section.) In this case, MAMC need only reference τ_{ij} for each $A_j \in A$. The ROC curves in Figure 4 compare the performance of all five predictors on receptors `gpb` and `pdg` for $|L| = 1$, based on the predictors' scores and DUD's class labels.

MAMC clearly outperforms the other predictors on both `gpb` and `pdg`. In particular, MAMC outperforms the best individual predictor eHiTS on `pdg`, even though the majority of its individual predictors perform poorly. In contrast, the performance of the consensus scorer RankSum on `pdg` was considerably worse than MAMC; it requires accurate rankings from most of its constituent predictors for satisfactory performance, rankings the individual predictors could not provide.

4 Discussion

We remind the reader that each of the applications described here was developed in part because carefully honed individual agents produced after many millions of hours of development had proved unsatisfactory. Not only is there no reliable way to predict the difficulty of a CSP, but also the solvers' performances vary from one problem to the next. A similar situation exists with predictors for PLD binding energy: their performance varies unpredictably. Both are hard problems on which MAMC has made some progress. Some choices, however, require further examination.

MAMC assumes that an agent's accuracy on similar cases will also be similar, but the number of those cases (i.e., the size of L) is an important decision. For portfolio construction, Table 3 reports on $r = 0.005$ which, given 10-fold cross validation, selects $|L| = 13$ cases from among the 2578 eligible ones. This enabled RSR-WG to outperform its competitors for $k > 1$ ($p < 0.005$). Table 4 explores values of r that enlarge L to as many as 412 cases. The data there suggest that, while the smallest r is reliable, occasionally a larger neighbor set pays off, particularly for the (non-parallel) $k = 1$.

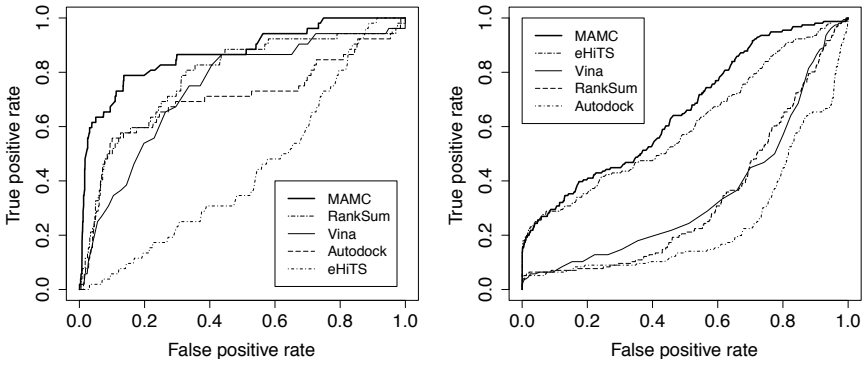

Fig. 4. ROC curves for PLD predictors on receptors `gpb` (left) and `pdg` (right)

Table 4. Mean and standard deviation for the number of problems solved by RSR-WG out of 2865 over 10 runs with k processors. Best value for k processors is in boldface, p <0.005.

	Neighbor set ratio r											
k	0.005		0.01		0.02		0.04		0.08		0.16	
1	2773	3.65	2779	3.20	2786	2.30	**2789**	3.17	2788	3.09	**2789**	2.51
2	**2826**	3.51	2821	2.49	2823	3.16	2816	2.97	2810	2.99	2809	2.87
3	**2841**	2.12	2836	1.93	2839	2.56	2832	2.07	2827	2.27	2819	2.07
4	**2850**	2.15	2847	1.57	2847	2.63	2843	2.06	2838	2.22	2832	2.50
5	**2855**	1.37	2851	2.35	2852	0.88	2850	1.78	2845	2.72	2843	3.26
6	**2857**	0.95	2855	1.07	2856	1.26	2853	1.64	2851	1.03	2850	1.07
7	**2858**	0.79	**2858**	0.57	2857	0.82	2855	1.83	2854	2.35	2854	1.14
8	2859	1.18	**2860**	1.34	2858	1.06	2858	1.18	2856	0.74	2855	1.43
16	**2864**	0.42	**2864**	0.00	**2864**	0.00	2863	0.00	2861	0.42	2861	0.47

Next we consider the impact of larger $|L|$ on PLD prediction. Again, each of the three scoring functions predicts for e, and then MAMC evaluates its performance on L with equation (5). MAMC then combines the predicted scores from all three predictors with equation (6), which allows it to consider the overall weighted performance of each predictor on a set of similar cases, and then takes the weighted prediction of the agent with the best overall performance on those similar cases. Although Figure 5 shows a clear performance improvement for $|L| = 2$ on both receptors, the improvement of $|L| = 3$ over $|L| = 2$ is only marginal, despite the fact that under leave-one-out validation, $|C| = 1900$ for `gpb` and 5759 for `pdg`.

The nature of the data, we believe, accounts for the difference in the appropriate choice for $|L|$. The PLD data is inherently noisy and incomplete; the cases in C are only those that have been tested by a laboratory and made publicly available. The dismal performance of a case, for example, may have dissuaded further testing of similar ones. For such a case there would be very few similar cases, so a larger L would provide little benefit. In contrast, competition CSPs are typically submitted by researchers who expect their own solvers to have an advantage on those problems. A *class* of CSPs consists of problems that may vary somewhat in their size or anticipated difficulty but have some structural or modeling commonality. Each CPAI'08 problem class typically had dozens of problems, so that, even under 10-fold cross-validation, MAMC is likely to find more than a few similar cases. Thus neighbor set size should be dependent on how likely MAMC is to find truly similar cases.

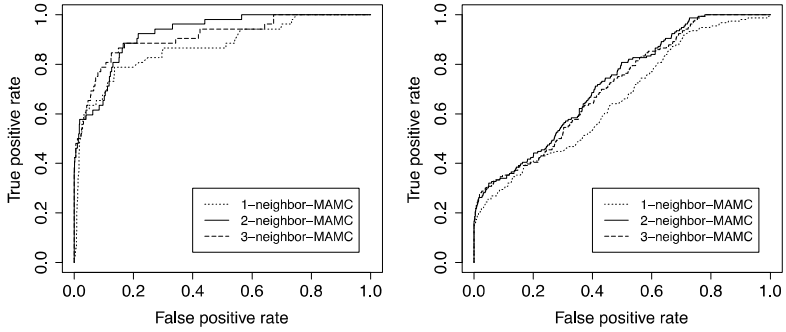

Fig. 5. ROC curves for MAMC with different $|L|$. on `gpb` (left) and `pdg` (right)

Given a fixed size for L, MAMC's confidence about its results is still likely to vary from one case to the next. For example, as noted above, a particular case may have an L whose members are only slightly similar to it. Moreover, individual agents may perform poorly on some members of L. In both situations, MAMC should be less confident about its prediction on the original case. Intuitively, if MAMC could categorize individual cases by confidence level, it might improve its performance on the cases where its confidence level is high.

Our confidence analysis considers three kinds of predictions, demonstrated here on protein-ligand docking where confidence before real-world laboratory testing is particularly important. Two cases c_i and c_j are said to be *similar* if and only if $s(c_i,c_j) > t_1$ (here, 0.8), and *dissimilar* otherwise. A *reliable* predictor is one whose performance, as calculated by equation (7), is greater than t_2 (here, 0.9); otherwise it is *unreliable*. Together t_1 and t_2 define three categories of agent A_j's ability to decide on example e. A prediction has *high confidence* if e's closest neighbor c is similar to e and A_j is reliable on c. A prediction has *low confidence* if c is dissimilar to e and A_j is unreliable on c. In all other situations, a prediction has *normal confidence.*

Figure 6 isolates the performance of MAMC at these three confidence levels for `gpb` and `pdg`. For `gpb`, 31.77%, 49.50%, and 18.73% of the chemicals had high, normal, and low confidence, respectively. For `pdg`, these percentages were 19.77%, 60.63%, and 19.60%. As expected, MAMC performed far better on the high-confidence chemicals for both receptors than it did on the full set. The benefit introduced by the confidence-based classification for `pdg` is particularly promising: although most candidate scoring functions had unreliable performance, confidence-based MAMC achieved almost perfect prediction on the high-confidence chemicals.

Might s alone have accurately predicted whether a chemical was a ligand? To investigate, we ranked by similarity all pairs of cases that included at least one ligand: each pair is either a *match* (two ligands) or a *non-match* (a ligand and a decoy). Ideally, match pairs should have higher similarity scores than non-match pairs. In Figure 7, the ROC curve for each receptor is based only on s and whether or not a pair was a match. Although chemical similarity alone clearly distinguishes ligands from decoys in the DUD benchmark data set, it provides fewer likely ligands than MAMC, whose predictive performance is considerably better, especially when its confidence is high.

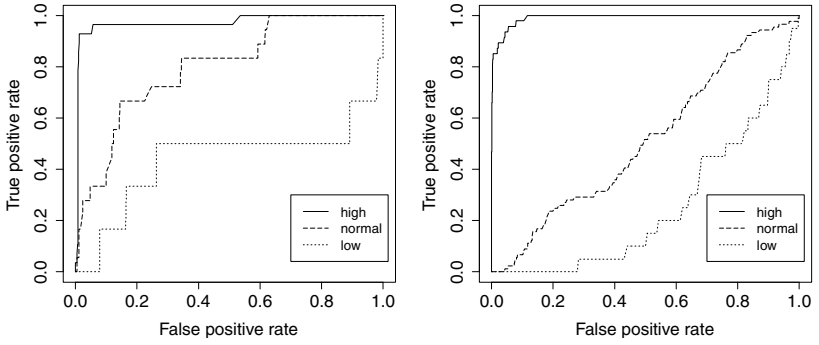

Fig. 6. ROC curves for confidence analysis on `gpb` (left) and `pdg` (right)

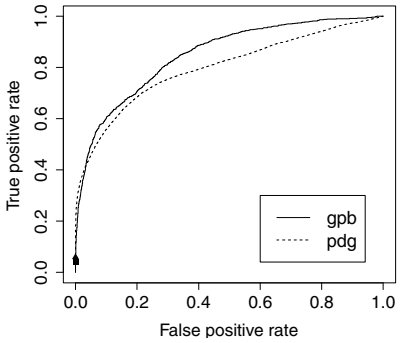

Fig. 7. ROC curves for `gpb` and `pdg` based on computed similarity and match/non-match labels of chemical pairs. The marks at lower left correspond to the predictions based only on the minimum chemical similarity score 0.8.

Diversity is essential in MAMC. For parallel scheduling we used 24 agents, 36 features, and 2578 cases per run under 10-fold cross validation. For PLD, we used 3 agents, 1024 features, and 1710 or 5183 cases per run. We believe that MAMC succeeds in part because it can draw upon such diversity. For PLD, there may be only three agents, but they are quite diverse both in their knowledge base and in their approach: AutoDock and AutoDock Vina use a genetic algorithm to search the ligand conformational space, while eHiTS uses systematic search in the conformational space of ligands. They variously consider force fields, energy terms (e.g., Van der Waals, electrostatic), empirical binding affinities, and knowledge-based scores trained from known protein-ligand complexes. Moreover, the different features in the PLD case representation provided an entirely different perspective on the cases. Although the construction of each case base required machine-months of computation, we view the increased availability of data not as a burden but as an opportunity.

Many sophisticated agents now offer the ability to set parameters. Future work will investigate the use of copies of one agent with different parameter settings or an element of randomization, as well as information flow algorithms to improve the case-similarity metric s. Moreover, although here τ was known in advance, it could be computed during MAMC's execution, after L has been chosen.

5 Conclusion

MAMC's reliance on multiple, well-respected agents draws strength from each of them, and its use of multiple, weighted, similar cases provides greater resiliency when its agents err. MAMC integrates similarity-based reference-case selection with performance-based predictor selection in a single framework. In addition, MAMC can report its confidence in its prediction, and achieves greater accuracy on confident cases. In practice, this will allow real-world laboratory experiments to focus on MAMC's high-confidence predictions, which promise a high success rate.

MAMC's portfolios of deterministic constraint solvers outperform those from naïve parallel versions of popular portfolio constructors. With only a few additional processors MAMC's schedules are competitive with an oracle solver on one processor. MAMC's improved predictions on compound virtual screening for protein-ligand docking suggests that PLD could support real drug-discovery. Moreover, with careful formulation of *C, s,* and *A*, MAMC should readily apply to other challenging bioinformatics and chemo-informatics tasks, including prediction of two- and three-dimensional protein structures, protein-protein interaction, protein-nucleotide interaction, disease-causing mutation, and the functional roles of non-coding DNA. Highly-confident predictions from MAMC there should be worthy of particular attention.

Acknowledgements. This work was supported in part by the National Science Foundation under grants IIS-1242451, IIS-0811437, CNS-0958379 and CNS-0855217, and the City University of New York High Performance Computing Center. Weiwei Han prepared the cases for PLD.

References

1. Dietterich, T.G.: Ensemble methods in machine learning. In: Kittler, J., Roli, F. (eds.) MCS 2000. LNCS, vol. 1857, pp. 1–15. Springer, Heidelberg (2000)
2. Leake, D.B., Sooriamurthi, R.: Automatically Selecting Strategies for Multi-Case-Base Reasoning. In: Craw, S., Preece, A.D. (eds.) ECCBR 2002. LNCS (LNAI), vol. 2416, pp. 204–233. Springer, Heidelberg (2002)
3. Plaza, E., McGinty, L.: Distributed Case-Based Reasoning. The Knowledge Engineering Review 20(3), 261–265 (2005)
4. Redmond, M.: Distributed Cases for Case-Based Reasoning: Facilitating Use of Multiple Cases. In: Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI 1990), pp. 304–309 (1990)
5. Kar, D., Chakraborti, S., Ravindran, B.: Feature Weighting and Confidence Based Prediction for Case Based Reasoning Systems. In: Agudo, B.D., Watson, I. (eds.) ICCBR 2012. LNCS, vol. 7466, pp. 211–225. Springer, Heidelberg (2012)
6. Yun, X., Epstein, S.L.: Learning Algorithm Portfolios for Parallel Execution. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, vol. 7219, pp. 323–338. Springer, Heidelberg (2012)
7. Guerri, A., Milano, M.: Learning Techniques for Automatic Algorithm Portfolio Selection. In: Proceedings of the Sixteenth European Conference on Artificial Intelligence, pp. 475–479 (2004)
8. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using Case-Based Reasoning in an Algorithm Portfolio for Constraint Solving. In: Proceedings of the Nineteenth Irish Conference on Artificial Intelligence and Cognitive Science (2008)
9. Silverthorn, B., Miikkulainen, R.: Latent Class Models for Algorithm Portfolio Methods. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, pp. 167–172 (2010)
10. Xu, L., Hoos, H.H., Leyton-Brown, K.: Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, pp. 210–216 (2010)

11. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-Based Algorithm Selection for SAT. Journal of Artificial Intelligence Research 32, 565–606 (2008)
12. Horvitz, E., Ruan, Y., Gomes, C.P., Kautz, H.A., Selman, B., Chickering, D.M.: A Bayesian Approach to Tackling Hard Computational Problems. In: Proceedings of the Seventeenth Conference in Uncertainty in Artificial Intelligence, pp. 235–244. Morgan Kaufmann Publishers Inc. (2001)
13. Streeter, M., Golovin, D., Smith, S.F.: Combing Multiple Heuristics Online. In: Proceedings of the Twenty-Second National Conference on Artificial Intelligence, pp. 1197–1203 (2007)
14. Mistral, http://4c.ucc.ie/~ehebrard/Software.html
15. Third International CSP Solver Competition (CPAI 2008), http://www.cril.univ-artois.fr/CPAI08/
16. Fourth International CSP Solver Competition (CSC 2009), http://www.cril.univ-artois.fr/CSC09/
17. Huang, S.-Y., Zou, X.: Advances and Challenges in Protein-Ligand Docking. International Journal of Molecular Science 11, 3016–3034 (2010)
18. Charifson, P.S., Corkery, J.J., Murcko, M.A., Walters, W.P.: Consensus Scoring: A Method for Obtaining Improved Hit Rates from Docking Databases of Three-Dimensional Structures into Proteins. Journal of Medicinal Chemistry 42, 5100–5109 (1999)
19. Clark, R.D., Strizhev, A., Leonard, J.M., Blake, J.F., Matthew, J.B.: Consensus Scoring for Ligand/Protein Interactions. Journal of Molecular Graphics Modelling 20, 281–295 (2002)
20. Wang, R., Wang, S.: How Does Consensus Scoring Work for Virtual Library Screening? An Idealized Computer Experiment. Journal of Chemical Information and Computer Sciences 41, 1422–1426 (2001)
21. Zsoldos, Z., Reid, D., Simon, A., Sadjad, B.S., Johnson, P.A.: Ehits: An Innovative Approach to the Docking and Scoring Function Problems. Current Protein and Peptide Science 7, 421–435 (2006)
22. Trott, O., Olson, A.J.: Autodock Vina: Improving the Speed and Accuracy of Docking with a New Scoring Function, Efficient Optimization and Multithreading. Journal of Computational Chemistry 31, 455–461 (2010)
23. Morris, G.M., Goodsell, D.S., Halliday, R.S., Huey, R., Hart, W.E., Belew, R.K., Olson, A.J.: Automated Docking Using a Lamarckian Genetic Algorithm and Empirical Binding Free Energy Function. Journal of Computational Chemistry 19, 1639–1662 (1998)
24. Miteva, M.A., Lee, W.H., Montes, M.O., Villoutreix, B.O.: Fast Structure-Based Virtual Ligand Screening Combining Fred, Dock, and Surflex. Journal of Medicinal Chemistry 48, 6012–6022 (2005)
25. Fukunishi, H., Teramoto, R., Takada, T., Shimada, J.: Bootstrap-Based Consensus Scoring Method for Protein-Ligand Docking. Journal of Chemical Information and Modeling 48, 988–996 (2008)
26. Huang, N., Shoichet, B.K., Irwin, J.J.: Benchmarking Sets for Molecular Docking. Journal of Medicinal Chemistry 49, 6789–6801 (2006)