

Algorithm Selection as a Collaborative Filtering Problem

Mustafa Misir, Michèle Sebag

▶ **To cite this version:**

Mustafa Misir, Michèle Sebag. Algorithm Selection as a Collaborative Filtering Problem. [Research Report] 2013, pp.43. hal-00922840

HAL Id: hal-00922840
https://inria.hal.science/hal-00922840

Submitted on 31 Dec 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Algorithm Selection as a Collaborative Filtering Problem

Mustafa Mısır , Michele Sebag

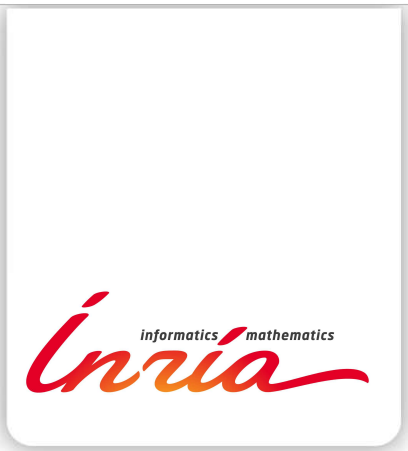
RESEARCH REPORT

N° XX

December 2013

Project-Teams TAO

ISSN 0249-6399 ISRN INRIA/RR--XX--FR+ENG



Algorithm Selection as a Collaborative Filtering Problem

Mustafa Mısır *, Michele Sebag †

Project-Teams TAO

Research Report n° XX — December 2013 — 40 pages

Abstract: Focusing on portfolio algorithm selection, this paper presents a hybrid machine learning approach, combining collaborative filtering and surrogate latent factor modeling.
Collaborative filtering, popularized by the Netflix challenge, aims at selecting the items that a user will most probably like, based on the previous movies she liked, and the movies that have been liked by other users. As first noted by Stern et al (2010), algorithm selection can be formalized as a collaborative filtering problem, by considering that a problem instance "prefers" the algorithms with better performance on this particular instance.
A main difference between collaborative filtering approaches and mainstream algorithm selection is to extract latent features to describe problem instances and algorithms, whereas algorithm selection most often relies on the initial descriptive features.
A main contribution of the present paper concerns the so-called cold-start issue, when facing a brand new instance. In contrast with Stern et al. (2010), ARS learns a non-linear mapping from the initial features onto the latent features, thereby supporting the recommendation of a good algorithm for the new problem instance with constant computational cost.
The experimental validation of ARS considers the domain of constraint programming (2008 CSP and 2011 SAT competition benchmarks) and gradient-free continuous optimization (black-box optimization benchmarks), demonstrating the merits and the genericity of the method.

Key-words: Algorithm selection; hyper-parameter tuning; meta-learning; portfolio approaches

* INRIA Saclay–Île de France, TAO - LRI-Université Paris-Sud, Bât. 660, 91405 Orsay Cedex, France, mustafamisir@gmail.com
† CNRS, LRI-Université Paris-Sud, TAO - Bât. 660, 91405 Orsay Cedex, France, sebag@lri.fr

RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

Sélection d'algorithme: Un problème de filtrage collaboratif

Résumé : Cet article s'intéresse à la sélection de l'algorithme le plus adapté à l'instance de problème considérée, parmi les algorithmes d'une plate-forme donnée.

L'approche proposée s'inspire du filtrage collaboratif. Popularisé par le challenge Netflix, le filtrage collaboratif recommande les produits qu'un utilisateur peut apprécier, en se fondant sur l'historique des produits appréciés par cet utilisateur et par la communauté des utilisateurs.

Comme noté par Stern et al. (2010), la sélection d'algorithmes peut être vue comme un problème d'apprentissage collaboratif, où une instance de problème est vue comme un utilisateur qui "préfère" les algorithmes dont la performance sur cette instance est la meilleure. Une différence essentielle de l'approche par CF par rapport l'état de l'art en slection d'algorithme est de caractriser les instances de problmes et les algorithmes en terme de facteurs latents au lieu de se limiter aux attributs initiaux (comme e.g. CPHydra ou SATzilla).

S'inspirant de Stern et al. (2010), cet article présente l'algorithme ARS (*Algorithm Recommender System*), exploitant les données des performances des algorithmes sur les instances disponibles pour faire de la sélection d'algorithme. Une contribution essentielle concerne le problème dit du démarrage à froid, où on ne dispose d'aucune information sur une nouvelle instance de problème. L'approche proposée s'appuie sur l'apprentissage d'un modèle non-linaire, estimant les facteurs latents à partir des descripteurs initiaux. ARS réalise ainsi la sélection d'algorithmes pour les instances nouvelles à coût de calcul constant.

La validation expérimentale d'ARS considère les domaines de la programmation par contraintes (2011 SAT Competition problems et CSP 2008 Competition benchmarks) et de l'optimisation stochastique continue (BBOB 2012 noiseless datasets), démontrant la généricité de l'approche proposée.

Mots-clés : Sélection d'algorithmes; optimisation d'hyper-paramètres; approche portfolio; meta-apprentissage.

1 Introduction

In many fields, such as constraint programming, machine learning, or gradient-free optimization, a variety of algorithms and heuristics has been developed to address the specifics of problem instances. How to choose the algorithm or the heuristic and how to tune the hyper-parameter values in order to get peak performances on a particular problem instance, has been identified a key issue since the late 70s [Ric76]. Known under various names (meta-learning [BS00], hyper-parameter tuning [Hoo12] or portfolio algorithm selection [XHHLB12a]), this issue commands the efficient deployment of algorithmic platforms outside research labs.

In the domain of constraint programming for instance (more in Section 2), extensive algorithm portfolios have been proposed [GS01] and significant advances regarding algorithm selection have been made in the last decade [XHHLB08, OHH+08, SM07, EFW+06, HHLBS09a, XHHLB12a]. These approaches involve two main ingredients. Firstly, a comprehensive set of features (see e.g. [XHHLB12b]) has been manually designed to describe a problem instance (static features) and the state of the search during the resolution process (dynamic features). Secondly, the performance of the portfolio algorithms on extensive suites of problem instances has been recorded. These ingredients together define a supervised machine learning (ML) problem: learning to predict the performance of a given algorithm based on the feature value vector describing a problem instance. The solution of this ML problem supports the selection of the expected best algorithm for any further problem instance.

In this paper, another machine learning approach is leveraged to handle algorithm selection: collaborative filtering [SK09]. Collaborative filtering (more in Section 3) pertains to the field of recommender systems: exploiting the user data (the items she liked/disliked in the past) and the community data (recording which user liked which items), a recommender system proposes new items that the user is most likely to like.

Taking inspiration from [SSH+10], Algorithm selection is viewed as a collaborative filtering problem (Section 4) where each problem instance (respectively each algorithm) is viewed as a user (resp. an item) and a problem instance is said to "prefer" the algorithm iff the algorithm yields better performance on the problem instance, comparatively to the other algorithms which have been launched on this problem instance. Following Cofirank [WKLS07] and Matchbox [SHG09, SSH+10], the proposed *Algorithm Recommender System* extracts latent features to characterize problem instances on the one hand, and algorithms on the other hand. A main difference compared to the state of the art in CP algorithm selection [OHH+08, XHHLB12a] or Meta-learning [BS00, PBGC00, Kal02, BGCSV09] is to rely on latent features as opposed to original features (see also [SM10]). The use of latent features has two main consequences. On the one hand, it enables to cope with incomplete data (when a small fraction of the algorithms has been run on every problem instance), significantly increasing the usability of the approach and decreasing its computational cost. On the other hand, it provides for free a description of the problem instances and of the algorithms, supporting new facilities, such as estimating the "novelty" of a problem instance, and evaluating the diversity of a benchmark set.

The main limitation of the collaborative filtering approach is the so-called *cold start* issue, when considering a brand new problem instance[1]. In such cases, collaborative filtering usually falls back on recommending the default "best hit" option. When some

[1] Another cold-start problem is faced when a new algorithm is added to the algorithm portfolio. This problem is outside the scope of the present paper.

initial descriptive features are available, taking inspiration from [Hoo12, BBKS13], another contribution of the paper is to define a surrogate model of the latent features. This surrogate model, estimating the latent features from the initial features, is used to select algorithms best suited to this brand new problem instance, thus tackling the cold-start issue.

The merits of the approach are experimentally investigated, considering algorithm selection in several application domains: constraint satisfaction, constraint programming and black-box optimization. The genericity of the approach is studied w.r.t. the performance measures relative to the application domains; additionally, the sensitivity of the approach w.r.t. the sparsity of the collaborative filtering data is investigated.

The paper is organized as follows. Section 2 discusses the state of the art in algorithm selection, specifically focusing on the constraint programming domain and on the related topic of hyper-parameter tuning. The basics of collaborative filtering are introduced in Section 3. Section 4 gives an overview of the proposed ARS approach, and details how it faces the cold-start issue. Section 5 presents the goal of experiments and experimental setting used to conduct the validation. Section 6 gives an overview of the experimental validation of the approach on the SAT 2011 Competition benchmark, the CSP 2008 competition instances and the Black-box optimization benchmark (the details are reported in the Appendix). Section 7 concludes the paper with a discussion and some perspectives for further work.

2 State of the art

This section reviews and discusses some related work concerned with portfolio algorithm selection and hyper-parameter tuning, focusing on the domains of constraint programming and machine learning.

2.1 Algorithm Selection

Since [Ric76], algorithm selection was acknowledged to be a core issue for constraint programming. On the one hand, there was no such thing as a solver outperforming all other solvers on all problem instances. On the other hand, the choice of the best algorithm for a given problem instance could neither rely on any theoretical analysis nor on efficient rules of thumb – all the more so as the set of solvers keeps increasing.

CP algorithm selection takes advantage from two facts. Firstly, the CP community is used to rigorously assess algorithms along international challenges, involving large sets of benchmark problem instances. Secondly, a comprehensive set of features with moderate computational cost was proposed to describe problem instances and the state of the search at any point in time, respectively referred to as static and dynamic features [XHHLB12b, KMS+11]. Examples of static features for a constraint satisfaction problem are the constraint density and tightness; examples of dynamic features are the number of unit propagation or the variance of the number of unsatisfied clauses in local optima.

It thus came naturally to define a supervised machine learning problem, where each benchmark problem instance is represented as a d-dimensional feature vector $\mathbf{x}$ ($\mathbf{x} \in \mathbb{R}^d$), labeled with the actual performance $y_A(\mathbf{x}) \in \mathbb{R}$ of any given algorithm A on this problem instance, say the runtime to solution for satisfiable instances. From the training set

$$\mathcal{E}_A = \{(\mathbf{x}_i, y_A(\mathbf{x}_i)), i = 1 \dots k\}$$

supervised machine learning algorithms, specifically regression algorithms[2], derive an estimate of the computational cost of A on any problem instance described by its feature vector $\mathbf{x}$,

$$\hat{y}_A : \mathbf{x} \in \mathbb{R}^d \mapsto \hat{y}_A(\mathbf{x}) \in \mathbb{R}$$

Considering all estimates $\hat{y}_A$, where A ranges over the algorithm portfolio, algorithm selection can proceed by selecting the algorithm with optimal estimated performance on the current problem instance $\mathbf{x}$ (arg $\max_A \hat{y}_A(\mathbf{x})$).

Among the many algorithm selection approaches designed for CP [XHHLB08, OHH+08, SM07, EFW+06, HHLBS09a, Kot12], CPHydra [OHH+08] and SATzilla [XHHLB08, XHHLB12a] are the most representative.

In CPHydra [OHH+08], a case-based approach was used, akin a nearest neighbor approach. Real-valued features indeed define a (semi)-distance on the set of problem instances; given the current problem instance $\mathbf{x}$, its nearest neighbor $\mathbf{x}_{nn}$ among the benchmark problem instances is determined and eventually CPHydra determines a schedule of solvers, expectedly most appropriate to the current problem instance $\mathbf{x}$.

In SATzilla [XHHLB08], ridge regression was initially used to estimate the runtime $\hat{y}_A$, modelling the empirical hardness of the problem instances for any particular algorithm A. A 2-step process is used to compute the description $\mathbf{x}$ of the current problem instance. Besides the static features, probing features are computed by running a few solvers on the problem instance. In later versions of SATzilla [XHHLB12a], a random forest algorithm is used for each pair of solvers (A,B) to determine whether solver A outperforms B on a given problem instance.

Independently, algorithm selection has also been intensively investigated in the field of supervised machine learning (ML) under the name of meta-learning (see e.g. [KH03, Kal02, BGCSV09, SVKB13]). The ML community is also used to rigorously assess algorithms using benchmark problem instances, and descriptive features have also been proposed to characterize ML problem instances, including the so-called landmarks computed by running fast ML algorithms on (subsets of) the problem instance.

Still, algorithm selection in ML is lagging behind algorithm selection in CP, in terms of empirical efficiency. This fact is blamed on the difficulty of engineering features accurately describing an ML problem instance with limited computational cost. A tentative interpretation for this drawback goes as follows. A constraint satisfaction problem instance is expressed as a set of constraints, thus with some generality (in *intension*). Quite the contrary, a supervised machine learning problem instance is expressed as a set of examples (in *extension*), thus with a low level of abstraction. While (an estimate of) the distribution of these examples could accurately describe the dataset, estimating the data distribution is no less difficult and computationally expensive as supervised learning itself.

Other approaches [SHG09, SSH+10, SM10], based on the extraction of latent features, will be discussed in section 3.

[2] An alternative is to formulate algorithm selection as a classification problem, where each problem instance is labeled with the best algorithm for this instance. The classification formulation is however more brittle (as several algorithms can have similar performances on a problem instance), and less scalable w.r.t. the number of algorithms in the portfolio. Hybrid approaches have been developed with promising results [Kot12], using stacking ML; specifically, the estimated runtimes for each portfolio algorithm are used as features for the classification problem.

2.2 Hyper-parameter tuning

Likewise, it is commonly acknowledged that in many cases the algorithm performance critically depends on the setting of its hyper-parameters, where the algorithm performance is domain-specific, e.g. time-to-solution in constraint programming or test error in machine learning. Tuning the hyper-parameter setting, a.k.a. algorithm configuration, thus is a key issue *on par* algorithm selection to yield peak performance on a particular problem instance.

The difference between algorithm configuration and algorithm selection is twofold. On the one hand, the configuration space is continuous or mixed (as parameters usually are real-valued and discrete), whereas the algorithm selection problem aims at selecting one among the portfolio algorithms. On the other hand, the goal often is to select the best hyper-parameter setting w.r.t. a set of problem instances (as opposed to, for a particular problem instance).

Some representative and effective algorithm configuration algorithms are REVAC [NE07], ParamILS [HHLBS09b] and SMAC [HHLB11], achieving an iterated local search in the parameter configuration space in order to yield optimal performance (e.g. find the solution with optimal quality with a given computational cost, or delivering a solution with prescribed quality with minimal computational cost). They proceed by maintaining a set of candidate configurations. A naive evaluation of a configuration would launch the algorithm with this configuration on all problem instances, thus entailing huge computational costs. The point thus is to discard as soon as possible unpromising configurations, through maintaining a confidence interval on their performances and pruning the dominated ones. In ParamILS for instance, a bound multiplier bm is used to cap the computational cost of the current configuration relatively to the cost of the best configuration so far. Another issue is to generate new configurations, using random univariate moves combined with random restart in ParamILS, and Estimation of Distribution Algorithms in REVAC. In [HHHLB06], the surrogate model predicting the algorithm runtime based on the descriptive features of the problem instance and the hyper-parameter setting, is likewise exploited to achieve the sequential optimization of the hyper-parameter setting.

In machine learning, algorithm configuration most often relies on the systematic exploration of the configuration search space along a grid search. This approach faces scalability issues as the number of configurations exponentially increases with the number of hyper-parameters; furthermore, the assessment of a single configuration is computationally expensive when considering large-sized datasets (the big data issue). A way of decreasing the number of configurations to be considered, known as surrogate optimization, is to iteratively estimate the configuration performance on the current ML problem instance from the configurations already assessed, and to use the performance estimate to select promising configurations. Surrogate optimization has been used for algorithm configuration in the context of deep belief networks [BB12] or on the top of the Weka platform [THHLB12]. Another approach, also based on surrogate optimization is proposed by [BBKS13], with the difference that the performance estimate uses the assessment of the algorithm configuration on other, similar, ML problem instances (datasets); the exploration of the configuration space is based on the expected improvement criterion [VVSW09].

2.3 Discussion

In all generality, both algorithm selection and algorithm configuration can be viewed as a hybrid estimation/optimization problem. Let Ω and X respectively denote the search space (the algorithm portfolio and/or the configuration space), and the instance space (usually $\mathbb{R}^d$ where d is the number of descriptive features on the problem instances). The estimation problem consists of building a surrogate model:

$$y : \Omega \times X \mapsto \mathbb{R}$$

where $y(\omega, \mathbf{x})$ estimates the performance of configuration ω on the problem instance $\mathbf{x}$.

In algorithm selection, the Ω space consists of $\{1, \ldots K\}$ if K denotes the number of algorithms in the portfolio. Algorithm selection thus proceeds by defining K surrogate models independently, from the performance of all K portfolio algorithms on all problem instances in the considered repository. Surrogate models do not share any information, and the exploration of the search space is exhaustive (i.e. all portfolio algorithms are potentially considered).

In algorithm configuration, the configuration space usually involves continuous parameters, preventing the exhaustive exploration of the search space. The surrogate model is used to guide the exploration along a sequential model-based optimization scheme. In some cases the surrogate model captures the *average* configuration behavior w.r.t. the set of problem instances, yielding a noisy optimization problem,and the key issue is to discard dominated configurations without running them on all problem instances (using e.g. Bernstein races as in [HMI09]).
In other cases, the surrogate model is meant to estimate the configuration performance on a specific problem instance $\mathbf{x}$. The issue is to leverage all available information about other problem instances, without compromising the accuracy w.r.t. $\mathbf{x}$.

Several authors [WKLS07, SHG09, BBKS13] suggest that it makes sense to consider rank-based surrogate models, that is, models defined up to a monotonous transformation. More formally, the rank-based surrogate model $y(\omega, \mathbf{x})$ is such that $y(\omega, \mathbf{x}) < y(\omega', \mathbf{x})$ iff configuration ω' is better than ω on problem instance $\mathbf{x}$, though $y(\omega, \mathbf{x})$ does not estimate the actual performance of configuration ω on problem instance $\mathbf{x}$. The rationale for this is that the actual performance might depend in an arbitrarily non-smooth way on $\mathbf{x}$, thereby hindering the surrogate learning phase.

This paper will focus on the algorithm selection issue, leaving the algorithm configuration optimization for further research (section 7).

3 Collaborative Filtering and Algorithm Selection

This section presents collaborative filtering (CF), referring the interested reader to [SK09] for a comprehensive survey. Some algorithm selection approaches based on CF are thereafter introduced and discussed.

3.1 Collaborative Filtering

Over the last years, CF has become increasingly popular to support recommender systems, to deal with the *long tail* phenomenon at the core of the e-commerce. Indeed the unprecedented diversity of the items available online (e.g. items for Amazon, or movies for Netflix) requires recommender systems to enable every user to find the items best matching their tastes. Formally, CF estimates whether an individual user

will like/dislike a particular item, based on the CF matrix $\mathcal{M}$ which stores the purchase and feedback history of the whole community of users: what they bought and what they liked/disliked [MS10]. Let n_r (respectively n_c) denote the number of users (resp. items). The $n_r \times n_c$ matrix $\mathcal{M}$ is high-dimensional; typically the Netflix challenge involves around 480,000 users and 18,000 movies [BL07]. Furthermore, $\mathcal{M}$ is only partially known: circa 1% or less of the pairs (user, item) is filled (e.g. a user actually sees very few movies on average).

Memory-based approaches proceed by defining a metric or a similarity on the user and item space. This metric supports the recommendation of items most similar to those items the user liked in the past, or the recommendation of items that users similar to the target user, liked in the past. Note that in the former case, the recommender system is biased toward *exploitation* as it does not propose new types of items to the user; it might thus fail to discover the true user's tastes. In the latter case, the *exploration* is enforced as the user crowd is assumed to have explored the whole set of items.

Various similarity functions ranging from cosine similarity and Pearson correlation to more adhoc measures (e.g. mixing *proximity, impact* and *popularity* [Ahn08]) have been considered to capture the relationships between users and items.

Model-based approaches proceed by extracting a general model from the CF data, aimed at the latent factors explaining the user behaviors. Such models reportedly provide better performance albeit at a higher computational cost. The most popular model-based approach for collaborative filtering is based on singular-value decomposition (SVD) [BP98], at the core of latent semantic analysis [DDF+90]. Assuming that matrix $\mathcal{M}$ is fully known, its singular value decomposition reads:

$$\mathcal{M} = U\Sigma V^t$$

where U is a square $n_r \times n_r$ matrix relating the users and the (left) singular eigenvectors of $\mathcal{M}$, V is a $n_c \times n_c$ square matrix relating the items and the (right) singular eigenvectors and Σ is a $n_r \times n_c$ matrix with non-null (and positive) coefficients $\lambda_1, \ldots \lambda_K$ on the principal diagonal only; by convention and with no loss of generality, singular eigenvalues λ_i are ordered by decreasing value ($\lambda_i \geq \lambda_{i+1}$). For statistical and computational reasons, a low-rank approximation of $\mathcal{M}$ is usually sought, by cancelling out all eigenvalues in Σ except the top-k ones. It reads:

$$\mathcal{M} \approx U\Sigma' V^t = U_k V_k^t$$

where Σ' differs from Σ as it sets all eigenvalues to 0 but the top-k ones. Denoting D the $k \times k$ diagonal matrix $diag(\sqrt{\lambda_1}, \ldots \sqrt{\lambda_k})$, U_k is the $n_r \times k$ matrix given as the product of U and $\left[\frac{D}{0}\right]$, and V_k is the $n_c \times k$ matrix given as the product of $[D|0]$ and V. For the sake of notational simplicity, U_k and V_k will be denoted U and V when no confusion is to be feared.

In the case of collaborative filtering, when $\mathcal{M}$ is sparsely filled, one can directly search for the U and V such that their product $U_k V_k^t$ approximates $\mathcal{M}$ for all (i, j) such that $\mathcal{M}_{i,j}$ is known.

$$\text{Find } U_k, V_k = argmin_{U,V} \left\{ \sum_{i,j \ s.t. \ \mathcal{M}_{i,j} \text{ is known}} \left(\sum_{\ell=1}^{k} U_{i,\ell} V_{j,\ell} - \mathcal{M}_{i,j} \right)^2 \right\}$$

Matrices U and V provide an interpretation of the CF matrix in terms of latent factors. For instance in the Netflix challenge, U models every user as a linear combination of

k archetypal users (e.g. romance, comedy or gore amateurs); likewise V models every movie as a linear combination of k archetypal movies. Along this line, U and V are used to estimate every missing value $\mathcal{M}_{i,j}$ in the CF matrix:

$$\widehat{\mathcal{M}_{i,j}} = \sum_{\ell=1}^{k} U_{i,\ell} V_{j,\ell}$$

The above is not operational when considering a new user or a new item for which no like/dislike is available in the CF matrix. Referred to as cold-start problem, this issue will be further discussed in section 4.3.

Several variants of matrix decomposition and several optimization criteria have been explored in the CF literature. Let us particularly mention the Cofirank approach [WKLS07], which exploits the ranks of the performances in the $\mathcal{M}$ matrix, as opposed to the precise performance values. The rationale for it goes as follows: what matters is the fact that a given user prefers movie1 to movie2, rather than the precise marks given to movie1 and movie2.

This intuition is captured by redesigning the optimization criterion above, and minimizing the fraction of triplets (i, j, k) such that the i-th user prefers the j-th item to the k-th item ($\mathcal{M}_{i,j} > \mathcal{M}_{i,k}$), while their estimated ranks are such that $\widehat{\mathcal{M}_{i,j}} < \widehat{\mathcal{M}_{i,k}}$. Formally, CofiRank is based on the Maximum Margin Matrix Factorization (MMMF) [SRJ05], alternating the optimization of U and V; bundle methods [TSVL07a] using lower-bounds on the optimization criteria based on Taylor approximations are used, where the minimization objective is set to Normalized Discounted Cumulative Gain (NDCG) (section 4.2), Ordinal Regression or Root-mean squared loss.

Note that collaborative filtering can accommodate user and item descriptive features, by extending the $\mathcal{M}$ matrix. Denoting f_r (respectively f_c) the number of user (resp. item) descriptive features, matrix $\mathcal{M}$ becomes an $(n_r + f_r) \times (n_c + f_c)$ matrix, where the i-th row concatenates the judgment of the i-th user on the movies she saw, and her own descriptive feature values, and the j-the column symmetrically concatenates the judgment of all users on the j-th movie, and the feature values associated to the j-th movie.

3.2 The Matchbox approach

A particular collaborative filtering is that of Matchbox [SHG09]. Likewise, the goal is to find some representation U and V, respectively mapping the users and the items on a k-dimensional space, in such a way that the appreciation by the i-th user of the j-th product can be estimated from $\langle U_{i,\cdot}, V_{j,\cdot} \rangle$, where $U_{i,\cdot}$ (respectively $V_{j,\cdot}$) denotes the i-th row of U (respectively the j-th row of V).

The main specificity of Matchbox is to achieve incremental learning, through an approximate message passing algorithm. The motivation behind this algorithm is twofold: on the one hand, it is meant to accommodate the preference drift of the users, whose tastes generally evolve along time; on the other hand, the algorithm can be used in fast mode with a single pass through the data. Matchbox uses a Bayesian approach, gradually updating mappings U_k and V_k and the associated biases, using Gaussian priors on the models and modelling the interpretation of the data (e.g. nominal ratings or clicks) by means of an additional decoding mechanism, the posterior distribution of which is also estimated. Some care must be exercized to enforce the decrease of the variance, using the minimization of the KL divergence in lieu of Expectation Propagation.

Matchbox has been approach approach applied to the portfolio algorithm selection in [SSH+10], based on the remark that *users are analogous to tasks and items are analogous to experts*. In the algorithm selection problem, tasks and experts respectively correspond to problem instances, and algorithms. A motivating application concerns the so-called scheduling strategy [SS08], where the point is to define a sequence of (algorithm, time-out) to be run successively on a given problem instance[3] [EFW+06]. A three-step process is devised to reach this goal. A distribution probability on the runtime of an algorithm for a given problem instance is set, conditionally to the latent rank of the algorithm. The mapping from the latent performance to the latent rank is learned via a cumulative threshold model, specific to the algorithm. Thirdly, the rank model is learned from the available data, recording the performances of all solvers on all training instances.

3.3 Discussion

Let us discuss the relevance of formulating algorithm selection (AS) as a collaborative filtering (CF) problem.

In formal terms, algorithm selection truly is a collaborative filtering problem: given some matrix $\mathcal{M}$ recording the preferences of the rows (instances or users) w.r.t. the columns (algorithms or items), the goal is to complete the matrix and find the algorithm best suited to a given problem instance.

In an applicative perspective however, the two problems differ by the nature, amount and cost of the available data. By the "big data" standards, algorithm selection is a small or medium-size problem, involving usually less than a few thousand problems and a few hundred algorithms, thus smaller by two orders of magnitude than the Netflix challenge. It is fair to say that, as soon as algorithm portfolios will involve a performance recording archive, the number of documented problems will increase; it is however unlikely that the number of algorithms will outpass the number of problems[4]. Collaborative filtering thus is a data-extensive problem, facing a data deluge with no possibility of conducting complementary experiments. Quite the contrary, algorithm selection could be viewed as a data-intensive problem, where complementary data acquisition is possible if expensive (e.g. one could use the computer idle cycles to conduct exploratory experiments [AHS10]).

Another difference regards the cold-start issue: while this issue is relevant to collaborative filtering, it is critical in algorithm selection.

A third difference regards the descriptive features of the users/problems. Poor descriptive features if any are available in collaborative filtering (e.g. age or gender). Sophisticated features have been (manually) proposed to describe CP problem instances, the relevance of which is at least implicity confirmed by the success of algorithm selection in the CP domain. Overall, it seems more feasible to compute descriptive features in algorithm selection than in collaborative filtering.

4 The Algorithm Recommender System

This section gives an overview of the *Algorithm Recommender System* (ARS). After discussing the position of the problem, the collaborative filtering aspects are tackled

[3] Note that finding the best sequence of actions to apply in order to solve a given problem instance should rather be formulated in terms of reinforcement learning [SB98, Sze10]. We shall return to this point later on.

[4] This does not hold for the hyperparameter tuning problem; we shall return to this in section 7.

Table 1: An example CF matrix $\mathcal{M}$ reporting the ranks of he some solvers among 7-solver portfolio and 5 problem instances

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
i_1	2	-	3	4	-	-	1
i_2	-	4	3	5	1	2	-
i_3	1	2	-	-	3	4	-
i_4	2	5	1	3	-	-	4
i_5	-	-	2	3	-	4	1

and the section last focuses on the cold start issue.

4.1 Position of the problem

Following [SSH+10], ARS tackles algorithm selection as a collaborative filtering problem. It takes in input a matrix $\mathcal{M}$, with $\mathcal{M}_{i,j}$ reporting the performance of the j-th algorithm on the i-th problem if available.

A main difference compared to the state of the art in algorithm selection (and even in the AS application of Matchbox reported in [SSH+10]) is that ARS does not assume the matrix $\mathcal{M}$ to be fully informed. The fraction of available data ranges in $10\%, 95\%$ in the experimental validation.

The performance depends on the application domain: it typically reflects the time-to-solution in CP and the test accuracy in ML. As argued by [WKLS07] however, it makes sense to consider that only the performance order matters: the actual performance f could be replaced by any $g\,o\,f$, with g a monotonous function on $\mathbb{R}$. As noted by [HRM+11, AAHO11], the principle of invariance under monotonous transformations of the objective function is at the core of robust and general optimization algorithms. Along this line, CF matrix $\mathcal{M}$ is only required to preserve the performance order: $\mathcal{M}_{ij} < \mathcal{M}_{ik}$ if algorithm k outperforms algorithm j on the i-th problem. In practice, $\mathcal{M}_{ij}$ can be thought of as the rank of the j-th algorithm on the i-th problem as in [SHG09, SSH+10], with the difference that i) the number of distinct ranks does not need to be limited and fixed in advance (Table 1); ii) the learning criterion aims at preserving the performance order (as opposed to the ranks; nominal regression).

4.2 Algorithm recommendation for known problem instances

The first ARS functionality consists of recommending new algorithms for problem instances on which some algorithms have already been launched. This functionality is achieved by estimating the missing $\mathcal{M}_{ij}$ values in $\mathcal{M}$. State of the art memory- and model-based options are considered, depending on how incomplete matrix $\mathcal{M}$ is.

The model-based option borrows CofiRank algorithm [WKLS07]. Formally, denoting k the number of latent factors, the goal is to map each problem instance (respectively algorithm) onto $\mathbb{R}^k$, in such a way that $\mathcal{M}_{i,j}$ is approximated by the scalar product of the k-dimensional vector $U_{i,\cdot}$ and $V_{j,\cdot}$:

$$\mathcal{M} \approx U'V$$

Matrices U and V are optimized using the Normalized Discounted Cumulative gain (NDCG) as learning criterion [JK00]. Let π_i denote the performance order related to

the i-th problem instance, with $\pi_i(j)$ being the j-th best algorithm after $\langle U_{i,.}, V_{j,.} \rangle$, the NDCG criterion is defined as:

$$DCG = \sum_i \sum_{\ell=1}^{K} \frac{2^{\pi_i(\ell)}-1}{log(\ell+2)}$$

$$NDCG() = \sum_i \sum_{\ell=1}^{K} \frac{2^{\pi_i(\ell)}-1}{log(\ell+2)} / \sum_i \sum_{\ell=1}^{K} \frac{2^{\ell}-1}{log(\ell+2)}$$

As NDCG is not a convex criterion, a linear upper-bound thereof is used and its optimization is tackled by alternate minimization [TSVL07b]. In the following, the i-th row $U_{i,.}$ of matrix U is referred to as *latent description* of the i-th problem instance.

The memory-based approach is a nearest neighbor-based regression, where the similarity of two problem instances i and j is the cosine of the i-th and j-th rows in $\mathcal{M}$. Denoting n_r the number of columns in $\mathcal{M}$,

$$sim(i,j) = \frac{\sum_{\ell=1}^{n_r} \mathcal{M}_{i,\ell}.\mathcal{M}_{j,\ell}}{\sqrt{\sum_{\ell=1}^{n_r} \mathcal{M}_{i,\ell}^2}.\sqrt{\sum_{\ell=1}^{n_r} \mathcal{M}_{j,\ell}^2}}$$

where by convention $\mathcal{M}_{i,j}$ is set to 0 if missing.

Finally the performance of the j-th algorithm on the i-th problem instance is defined up to a monotonous transformation as follows [RIS+94]:

$$\widehat{\mathcal{M}_{i,j}} = \overline{\mathcal{M}}_i + \frac{\sum_\ell sim(j,\ell) \left(\mathcal{M}_{i,\ell} - \overline{\mathcal{M}}_\ell\right)}{\sum_j sim(i,j)} \quad (1)$$

where $\overline{\mathcal{M}}_j$ is the average of $\mathcal{M}_{j,\ell}$ over all ℓ such that the ℓ-th algorithm was launched on the j-th problem instance.

4.3 The cold-start issue

The simplest way of tackling the cold-start issue is to first launch a few algorithms on the current problem instance, to augment matrix $\mathcal{M}$ with an extra-row and fill this extra-row with the ranks of these algorithms on the current problem instance, and to complete the extended $\mathcal{M}$ matrix as detailed in the above section.

This strategy suffers from two computational limitations. On the one hand, it requires to first launch sufficiently many algorithms on the current problem instance; on the other hand, it requires to redecompose matrix $\mathcal{M}$ from scratch for each new problem instance.

In the meanwhile, it is enough to estimate the latent factors $U_{\mathbf{x}}$ that would be associated to a new problem instance, to estimate the performance ranks on $\mathbf{x}$ of all algorithms in the portfolio. Since by construction

$$\mathcal{M}_{i,j} \approx \langle U_{i,.}, V_{j,.} \rangle$$

the performance of the j-th algorithm on the new problem instance could be estimated, up to a monotonous transformation, by $\langle U_{\mathbf{x}}, V_{j,.} \rangle$.

It will be assumed in the rest of the paper that a d-dimensional initial description of the problem instances is available, using e.g. sophisticated manually designed features as in CP [XHHLB12b], or more elementary features as in black-box optimization [AHS12].

The strategy proposed in ARS to handle the cold-start problem consists of learning the k-dimensional latent description $U_{\mathbf{x}}$ as a function of the initial description $\mathbf{x}$, where k is the dimension parameter of the CF decomposition, section 3:

$$\Phi : \mathbf{x} \in \mathbb{R}^d \mapsto U_{\mathbf{x}} \in \mathbb{R}^k$$

The learning of Φ is enabled as both the initial description $\mathbf{x}_i$ and the latent description $U_{i,\cdot}$ are available for all training examples. Considering the multi-dimensional regression dataset defined as $\mathcal{E} = \{(\mathbf{x}_i, U_{i,\cdot}), \mathbf{x}_i \in \mathbb{R}^d, U_{i,\cdot} \in \mathbb{R}^k\}$, Φ is learned using multi-dimensional regression.

Algorithm 1 Latent Estimate Cold Start (LECS)

input a sparse CF matrix $\mathcal{M} \in \mathbb{R}^{n_r \times n_c}$, ; number k of latent features; initial description $X \in \mathbb{R}^{n_r \times d}$ of all problem instances (centered with standard deviation 1).

1: Achieve k-rank decomposition of $\mathcal{M} = UV^t$
2: Solve the k-dimensional regression problem $\Phi : \mathbb{R}^d \mapsto \mathbb{R}^k$ from $\mathcal{E} = \{(X_{i,\cdot}, U_{i,\cdot}), i = 1 \ldots n_r\}$.
3: Let $\mathbf{x}$ be the initial description of a new problem instance $\mathbf{x} \in \mathbb{R}^d$)
4: Estimate its latent representation $U_{\mathbf{x}} = \Phi(\mathbf{x})$
5: Estimate the performance of the j-th algorithm on problem instance $\mathbf{x}$ as $\langle \Phi(\mathbf{x}), V_{j,\cdot} \rangle$
6: Return $\text{argmax}_j \langle \Phi(\mathbf{x}), V_{j,\cdot} \rangle$

This approach, referred to as Latent Estimate Cold Start (LECS), is investigated along two modes. The first mode, referred to as SVM-LECS, uses a support vector regression algorithm to tackle k independent regression problems. The second mode, referred to as NN-LECS, learns a neural net with d input and k output neurons.

This approach differs from [SSH+10], where a probabilistic estimate of the latent description is built as a linear combination of the initial description. Given a manually defined utility function, the cold-start problem is handled by returning the algorithm with best expected utility on the new problem instance. The main difference lies in how the latent features are estimated from the initial description. A linear estimate is considered in Matchbox, with the possibility of using a utility function to further calibrate the cold-start decision. A plain regression approach is used in ARS, based on two claims. Firstly, it might be inappropriate to constrain the CF matrix decomposition using the initial description, the relevance of which is unknown in many domains. The necessary regularization on the decomposition can be enforced through limiting the number of latent factors. Secondly, estimating latent factors as linear combinations of the initial features might lead to underfitting, all the more so as the number of training instances and the number of initial features enable to consider more complex regression models.

5 Experiment goals and setting

This section describes the methodology used for the experimental validation of ARS. For reproducibility, ARS as an executable jar and the benchmarks generated for each testbed are publicly available[5].

[5] http://allserv.kahosl.be/~mustafa.misir/ARS.html.

5.1 Goal of experiments

Two measures of performance are considered. The first one concerns the identification of the best algorithm for a problem instance when some algorithms have been launched on this problem instance, referred to as collaborative filtering (CF) performance; the sensitivity of the CF performance w.r.t. the fraction of algorithms launched on the problem instance, or incompleteness, is studied.

The CF performance is measured as follows. For each domain of application (CF matrix $\mathcal{M}$), an incompleteness level p ranging from 10 to 90% is considered. For each value of p, 10 datasets are generated by randomly removing $p\%$ of $\mathcal{M}$, conditionally to the fact that at least one entry remains for each problem instance and for each solver[6].

The second one concerns the identification of the best algorithm for a brand new problem instance, referred to as cold start (CS) performance. The CS performance is measured as follows. For each $\mathcal{M}$, a 10fold cross-validation is achieved; iteratively, 90% of the problem instances are used to train ARS and the surrogate latent model (section 4.3), and this model is used to compute the ranks of the solvers on the remaining 10% of the problem instances, a.k.a. test instances. Like for the CF performance, an incompleteness level p ranging from 10 to 90% is considered. For each value of p and each fold of the 10fold cross validation, 10 training sets are generated by randomly removing $p\%$ of the training set, conditionally to the fact that at least one entry remains for each problem instance and for each solver.

This paper last examines whether some insights into the structure of the problem instance space are provided by the latent description of the problem instances. Formally, the initial descriptive features and the latent features revealed by the model-based strategy are visually inspected using multi-dimensional scaling (MDS) [BG05]: given the $m \times m$ $\mathcal{D}$ matrix yielding the Euclidean distance among the m problem instances, computed from the initial (respectively, latent) features, the m problem instances are mapped on $\mathbb{R}^2$ in such a way that the matrix distance of the projected instances matches $\mathcal{D}$ to the best possible extent.

5.2 Application domains

Three domains of applications are considered.

5.2.1 Constraint satisfaction

The problem instances from the SAT 2011 competition[7], listed in Table 2 involve three categories of SAT problems, named APPLICATION (APP), CRAFTED (CRF) and RANDOM (RND). The data associated to phase 1 reports the performance of all solvers on all problems, i.e., the number of instances solved within a CPU budget of 1,200 seconds. The data associated to phase 2 reports the performance of the best solvers after phase 1, with timeout 5,000 seconds.

5.2.2 Constraint programming

The CSP 2008 competition benchmarks (Table 3) involve 4 categories of datasets: GLOBAL, k-ARY-INT, 2-ARY-EXT and N-ARY-EXT. Among them, k-ARY-INT is a

[6] After the removal, the ranks in $\mathcal{M}$ are updated for consistency (e.g. row $2, -, -, -, 5$ would be converted in $1, -, -, -, 2$). While this conversion is useless for the rank- model-based collaborative filtering algorithms, it matters for the memory-based algorithms).

[7] http://www.satcompetition.org/2011/

Table 2: SAT test datasets from SAT 2011 competition. The number of descriptive instance features is 54

	Dataset	# Solvers	# Instances	# Solved Instances	Best Single Solver
Phase 1	APP	67	300	228 (76%)	(174) Glucose 2.0 [AS09]
	CRF	52	300	199 (66%)	(138) ppfolio-seq [Rou11]
	RND	42	600	462 (77%)	(399) 3S [KMS+11]
Phase 2	APP	26	300	253 (84%)	(215) Glucose 2.0
	CRF	24	300	229 (76%)	(163) 3S
	RND	14	600	492 (82%)	(408) 3S

combination of 2-ARY-INT and N-ARY-INT datasets from the competition. The performance of the CSP solvers is, again, the number of instances solved within a CPU budget of 1800 seconds per instance.

Table 3: CSP 2008 competition datasets (from [YE12]). The number of descriptive instance features is 36

Dataset	# Solvers	# Instances	# Solved Instances	Best Single Solver
GLOBAL	17	548	493 (90%)	(416) Sugar-v1.13+picosat [TTB08]
k-ARY-INT ($k \geq 2$)	22	1411	1302 (92%)	(1165) cpHydra-k_40 [OHH+08]
2-ARY-EXT	23	633	620 (98%)	(572) cpHydra-k_10
N-ARY-EXT ($N > 2$)	24	546	449 (82%)	(431) cpHydra-k_40

5.2.3 Black-box Continuous Optimization

The problem instances from the black-box optimization benchmark [AHS12] [8], listed in Table 4, involve 24 well-known mathematical optimization objectives, each one being tackled in dimension 2, 3, 5, 10, 20, and 40. For each objective, each dimension and each algorithm, 15 independent runs are launched.

BBOB-Avg associates to each (objective, dimension, algorithm) the average performance (rank of the algorithm) out of the 15 runs. BBOB-Full considers each run as an independent problem instance[9].

5.3 Experimental setting

Two collaborative filtering options, respectively model-based and memory-based, are considered in ARS. The model-based option relies on CofiRank with the root-mean

[8] http://coco.gforge.inria.fr/doku.php?id=bbob-2012

[9] But instances with same objective, dimension, algorithm are either all training instances, or test instances.

Table 4: BBOB datasets. The number of descriptive instance features is 10 and 9 respectively

Dataset	# Solvers	# Instances	Best Single Solver
BBOB-Full	25	2160	(Rank: 8.11) DE-AUTO
BBOB-Avg	25	144	(Rank: 9.07) NIPOPaCMA

squared loss function, with rank k ranging in $\{1, \ldots, 10\}$. After a few preliminary experiments, the regularization parameter of CofiRank is set to 70 for SAT, 10 for BBOB and 200 for CSP.

The memory-based CF approach implements a nearest neighbor algorithm, with the number of nearest neighbors set to the overall number of solvers, legended as FullNN-Inst.

The surrogate latent model involved in the cold-start performance is learned using either a radius-based kernel SVM-regression, or a neural net (respectively legended with SVM or Neural prefix). After a few preliminary experiments, the parameters for the SVM-regression are $C = 1$ and $\gamma = 0.2$; the parameters for the neural net are set to 2 hidden layers, with d input neurons and d neurons on the hidden layers; the number of iterations of the back-propagation is set to 1000.

The ARS performances are assessed along a single-prediction and a top-3 prediction modes. The single prediction mode records the performance of the top recommended algorithm; the top-3 prediction mode likewise records the performance of the best algorithm among the top-3 recommended by ARS. The latter mode is legended with a 3P prefix.

All performance measures are assessed comparatively to the following baselines:

- The oracle (best solver for each individual problem instance);
- The single best algorithm (best algorithm in each problem category), and the top-3 best algorithms;
- A uniformly selected algorithm (legend Rand) and 3-uniformly selected algorithms (subject to include at least one algorithm solving the problem instance).

6 Experimental validation

The experimental results of ARS on the SAT 2011, CSP 2008 and BBOB 2012 application domains are organized along three issues: matrix completion, cold start, and visual inspection of the latent factors. The detailed results related to each application domain will be found in appendix.

6.1 Matrix completion

6.1.1 SAT 2011 domain

For the SAT 2011 domain, the memory-based FullNN-Inst strategy[10] outperforms the single-best baseline when the incompleteness ranges from 10% to 90% on SAT-Phase1 (Fig. 1.left), and from 10% to 80% on SAT-Phase2 (Fig. 2.left). Same trends are observed for the 3-parallel option (Figs. 1 and 2, right).

6.1.2 CSP 2008 domain

For the CSP 2008 domain, the memory-based FullNN-Inst strategy outperforms the single-best baseline when the incompleteness ranges from 10% to 80% on CSP-Global (Fig. 3, top), CSP-2-AryExt (Fig. 3, 3rd row) and from 10% to 70% on CSP-k-AryInt

[10]Detailed results show that for high incompleteness level ($> 80\%$), the model-based CofiRank@3 strategy slightly outperforms the memory-based one (Appendix 1).

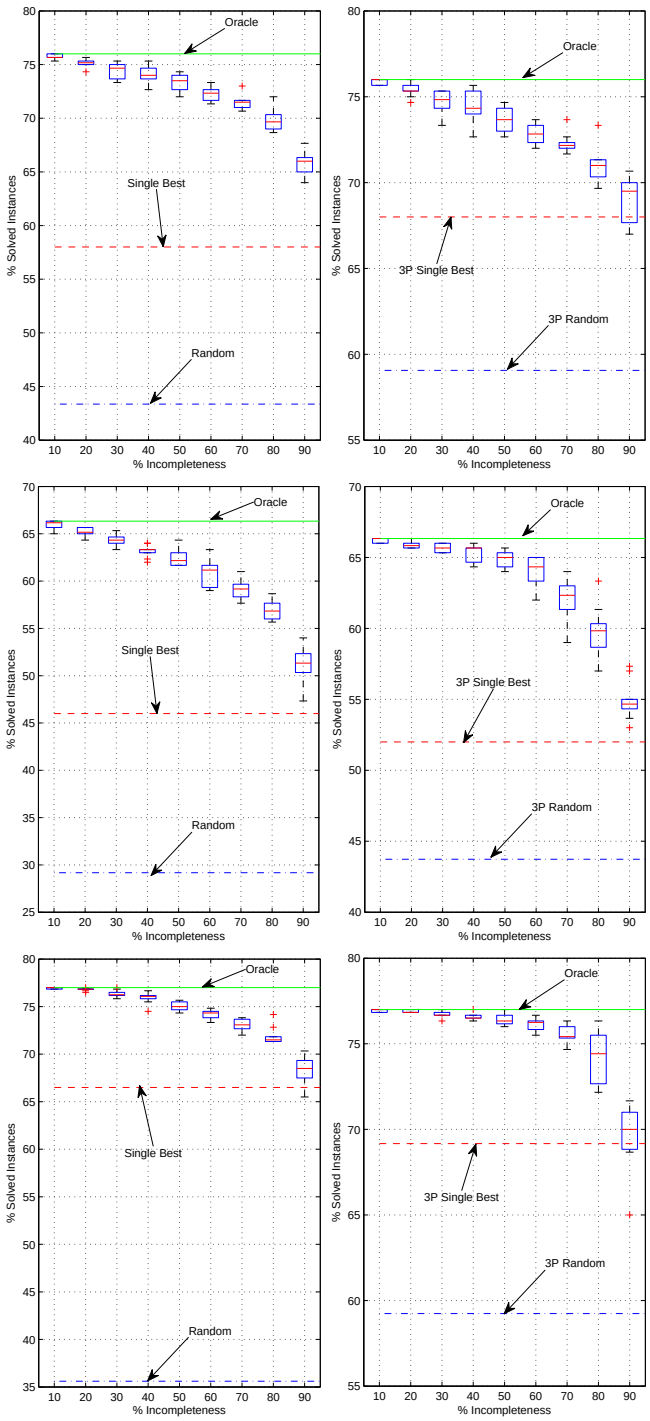


Figure 1: ARS performance on SAT-Phase1: percentage of solved instances versus incompleteness of the CF matrix. Top to bottom row: APP, CRF and RND. Left: single option; Right: 3-parallel option.

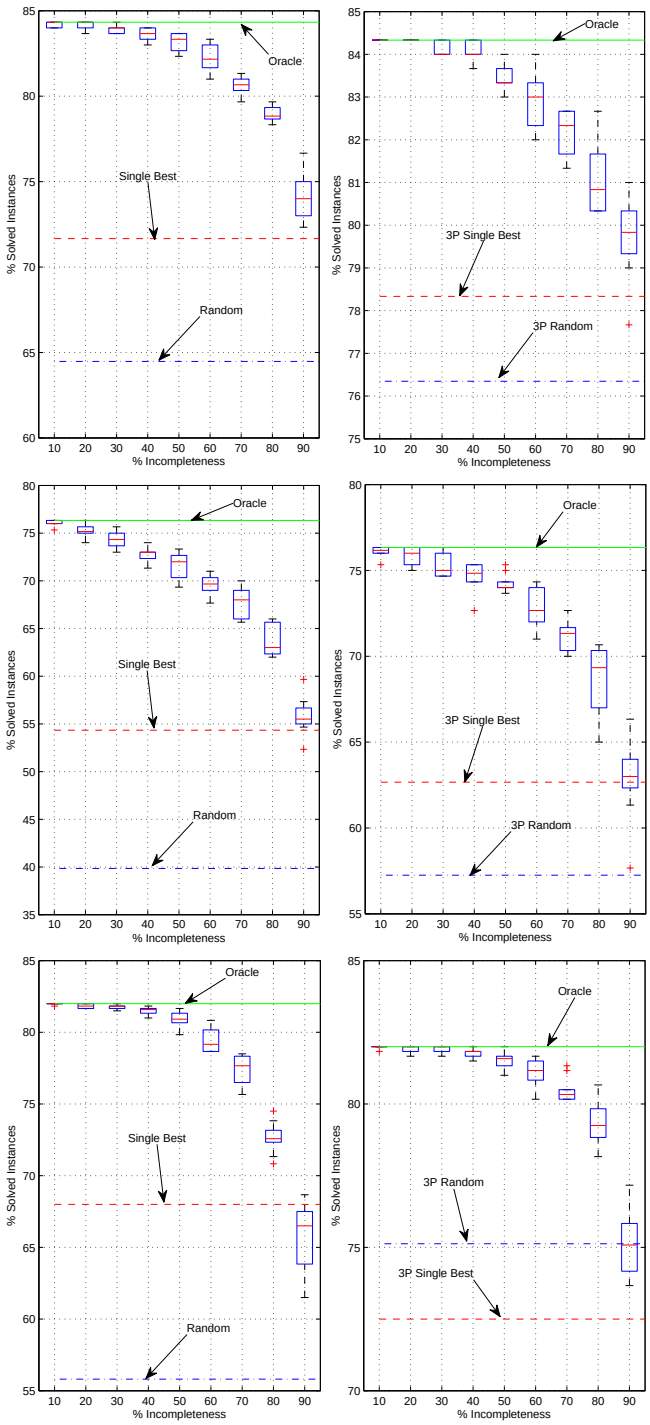


Figure 2: ARS performance on SAT-Phase2: percentage of solved instances versus incompleteness of the CF matrix. Top to bottom row: APP, CRF and RND. Left: single option; Right: 3-parallel option.

Table 5: Average number of solved instances with standard deviation across all the SAT cold-start benchmarks. The statistically significant best results after a Wilcoxon test with a 95% condifence interval, are indicated in bold

	Method	Phase 1			Phase 2		
		APP	CRF	RND	APP	CRF	RND
	Oracle	22.8 ± 2.5	19.9 ± 2.1	46.2 ± 3.7	25.3 ± 2.1	22.9 ± 2.5	49.2 ± 2.4
	Random	13.0 ± 2.2	8.8 ± 1.5	21.4 ± 1.8	19.3 ± 2.9	12.0 ± 1.2	33.5 ± 2.3
	SingleBest	$\mathbf{17.4 \pm 3.0}$	13.8 ± 1.9	39.9 ± 3.2	$\mathbf{21.5 \pm 3.6}$	16.3 ± 2.5	40.8 ± 2.4
	3P-Random	17.7 ± 2.2	13.1 ± 1.6	35.5 ± 2.5	22.9 ± 2.1	17.2 ± 1.4	45.1 ± 2.2
	3P-SingleBest	$\mathbf{20.4 \pm 2.3}$	15.6 ± 2.0	41.5 ± 3.3	23.5 ± 3.0	18.8 ± 2.5	43.5 ± 2.3
SVM	CofiRank	17.2 ± 2.5	$\mathbf{15.1 \pm 3.1}$	43.0 ± 3.9	$\mathbf{21.5 \pm 3.1}$	18.1 ± 3.0	$\mathbf{44.5 \pm 4.2}$
SVM	FullNN-Inst	$\mathbf{17.7 \pm 2.6}$	$\mathbf{15.2 \pm 2.6}$	42.2 ± 4.3	21.3 ± 3.2	$\mathbf{18.5 \pm 2.7}$	44.0 ± 4.4
SVM	3P-CofiRank	19.6 ± 2.4	$\mathbf{17.0 \pm 2.8}$	$\mathbf{45.0 \pm 3.8}$	23.4 ± 2.6	$\mathbf{20.8 \pm 3.4}$	47.1 ± 2.9
SVM	3P-FullNN-Inst	19.6 ± 2.4	$\mathbf{17.0 \pm 2.4}$	$\mathbf{44.9 \pm 3.6}$	$\mathbf{23.5 \pm 2.6}$	$\mathbf{20.8 \pm 2.6}$	46.9 ± 3.2
NNet	CofiRank	16.9 ± 2.9	$\mathbf{14.8 \pm 3.1}$	42.9 ± 3.7	$\mathbf{21.5 \pm 3.1}$	17.8 ± 2.6	$\mathbf{44.8 \pm 4.2}$
NNet	FullNN-Inst	17.2 ± 2.8	14.6 ± 2.8	42.2 ± 3.7	$\mathbf{21.5 \pm 3.3}$	$\mathbf{18.3 \pm 2.6}$	$\mathbf{44.5 \pm 4.4}$
NNet	3P-CofiRank	19.4 ± 2.5	16.6 ± 2.5	44.6 ± 3.6	23.5 ± 2.5	20.5 ± 2.3	$\mathbf{47.5 \pm 2.9}$
NNet	3P-FullNN-Inst	19.3 ± 2.6	$\mathbf{16.8 \pm 2.5}$	44.7 ± 3.5	$\mathbf{23.8 \pm 2.6}$	$\mathbf{20.7 \pm 2.4}$	$\mathbf{47.3 \pm 3.0}$

Fig. 3, 2nd row) and CSP-k-AryExt Fig. 3, bottom row). As in the SAT 2011 domain, the memory-based strategy FullNNInst outperforms the model-based strategy CofiRank[11]. Regarding the 3P option, ARS improves on the 3P-Best baseline when the incompleteness is less or equal than 60% on CSP-Global, 70% on CSP-k-AryExt, 80% on CSP-2-AryExt, and 90% on CSP-k-AryInt.

6.1.3 BBOB 2012 domain

On the BBOB 2012 domain, the ARS performance is measured as the average rank of the recommended algorithm (the smaller the better). As in the SAT 2011 and CSP 2008 domains, the memory-based FullNN-Inst strategy outperforms the other ARS options, and outperforms the single best baseline for incompleteness levels up to 80%.

6.2 Cold start

6.2.1 SAT 2011 domain

Table 5 shows the cold-start performance of ARS, averaged over all incompleteness levels of the training set, compared to the Oracle, Random and Single Best baselines. ARS significantly improves on the single-best baseline for the CRF and RND categories. The SVM- and neural net-based surrogate model of the latent factors yield similar results; the memory-based strategy FullNN slightly outperforms CofiRank on the APP and CRF categories, while CofiRank outperforms FullNN on RND. Similar trends are observed for the 3P option.

6.2.2 CSP 2008 domain

Table 6 shows the cold start performance of ARS averaged over all incompleteness levels of the training set, compared to the baselines. ARS matches the single-best

[11] Although the latter catches up for high incompleteness levels, see Appendix C.

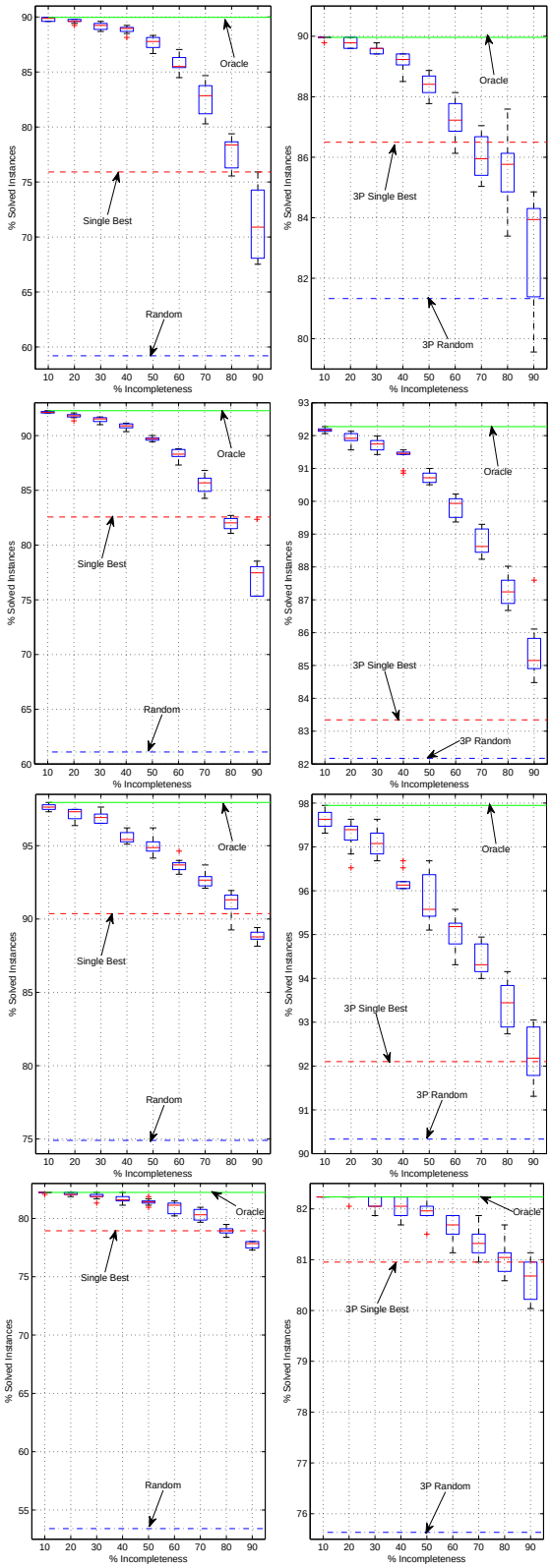


Figure 3: ARS performance on CSP 2008 versus incompleteness of the CF matrix: Percentage of solved instances on CSP-Global (top row), CSP-k-AryInt (second row), CSP-2-AryExt (third row) and CSP-k-AryExt (bottom row) using FullNN-Inst. Left: single option; Right: 3-parallel option.

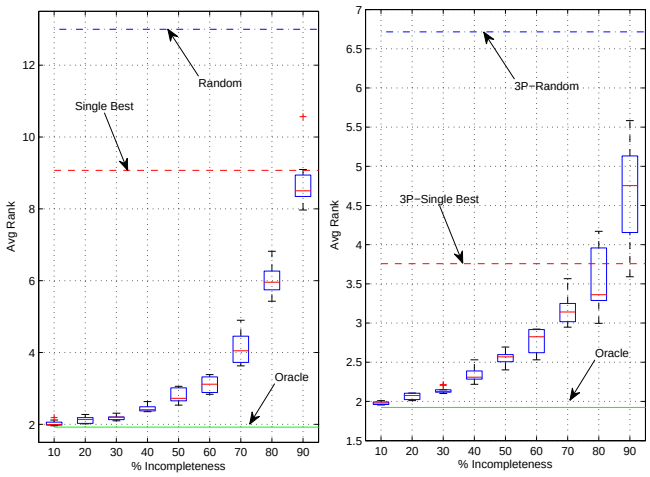


Figure 4: ARS performance on BBOB 2012 versus incompleteness of the CF matrix: Average rank of the recommended algorithm, using FullNN-Inst. Left: single option; Right: 3-parallel option.

baseline on all problem categories. The neural net-based and SVM-based surrogate models yield similar results, with the memory-based strategy FullNN slightly outperforming CofiRank. On the 3P option, ARS significantly outperforms the baseline for CSP-k-AryInt, and matches the baseline for other problem categories.

6.2.3 BBOB 2012 domain

Table 7 shows the cold-start performance of ARS averaged over all incompleteness levels of the training set, compared to the Oracle, Random and Single Best baselines.

In the BBOB 2012-avg setting (where a same problem instance is associated the average result over the 15 independent runs), ARS is slightly behind the single best baseline for the single option; this shortcoming is blamed on the poor initial features used to describe the benchmark problems, which cannot be compensated for by the latent features. Nevertheless, ARS significantly outperforms the baseline for the 3-parallel option.

As for the other domains, the memory-based FullNNInst strategy outperforms the model-based CofiRank strategy. The neural net surrogate model of the latent factors slightly outperforms the SVM one.

6.3 Visual inspection of the latent features

6.3.1 SAT domain

Fig. 5 displays the Phase 2 problem instances, mapped onto $\mathbb{R}^2$ using multi-dimensional scaling [BG05] based on the Euclidean distance of the 54 SATzilla features (left) and the first 10 latent features (right). In the APP and CRD categories (top and middle row), the micro-clusters based on the initial features are scattered, as latent features reveal new differences among the problem instances. In the RND category (bottom row), some micro-clusters are scattered while a bigger cluster is formed, suggesting

Table 6: Average number of solved instances with standard deviation across all the CSP cold-start benchmarks. The statistically significant best results after a Wilcoxon test with a 95% condifence interval, are indicated in bold

	Method	GLOBAL	k-ARY-INT	2-ARY-EXT	N-ARY-EXT
	Oracle	49.3 ± 3.2	130.2 ± 3.2	62.0 ± 1.3	44.9 ± 2.3
	Random	32.4 ± 3.2	86.2 ± 4.9	47.4 ± 3.2	29.2 ± 2.4
	SingleBest	41.6 ± 5.2	$\mathbf{116.5 \pm 5.8}$	$\mathbf{57.2 \pm 2.3}$	43.1 ± 2.8
	3P-Random	44.6 ± 3.2	115.9 ± 4.4	57.2 ± 2.3	41.3 ± 2.0
	3P-SingleBest	$\mathbf{47.4 \pm 3.8}$	117.6 ± 5.6	$\mathbf{58.3 \pm 2.5}$	$\mathbf{44.2 \pm 2.3}$
SVM	CofiRank	39.5 ± 5.1	111.5 ± 7.4	56.2 ± 2.9	42.2 ± 2.0
SVM	FullNN-Inst	43.6 ± 4.8	$\mathbf{115.3 \pm 6.9}$	$\mathbf{57.1 \pm 2.9}$	$\mathbf{43.4 \pm 2.4}$
SVM	3P-CofiRank	44.0 ± 4.0	119.6 ± 5.8	57.4 ± 2.4	43.8 ± 2.2
SVM	3P-FullNN-Inst	$\mathbf{47.0 \pm 3.6}$	$\mathbf{122.2 \pm 5.8}$	$\mathbf{58.3 \pm 2.2}$	$\mathbf{44.2 \pm 2.2}$
NNet	CofiRank	39.4 ± 5.1	110.8 ± 6.9	56.1 ± 2.9	42.1 ± 2.1
NNet	FullNN-Inst	$\mathbf{44.1 \pm 4.4}$	115.0 ± 6.4	$\mathbf{57.4 \pm 2.9}$	$\mathbf{43.4 \pm 2.6}$
NNet	3P-CofiRank	43.9 ± 4.0	119.1 ± 5.7	57.3 ± 2.5	43.8 ± 2.2
NNet	3P-FullNN-Inst	$\mathbf{46.9 \pm 3.5}$	$\mathbf{121.9 \pm 5.4}$	$\mathbf{58.6 \pm 2.3}$	$\mathbf{44.2 \pm 2.2}$

Table 7: Average ranks with standard deviation on the BBOB and BBOB-Avg datasets

	Method	BBOB	BBOB-Avg
	Oracle	2.49 ± 0.10	1.99 ± 0.57
	Random	13.00 ± 0.00	13.00 ± 0.00
	SingleBest	8.11 ± 0.69	$\mathbf{7.32 \pm 1.05}$
	3P-Random	6.79 ± 0.05	6.74 ± 0.26
	3P-SingleBest	3.74 ± 0.05	5.10 ± 0.98
SVM	CofiRank	5.86 ± 1.17	7.73 ± 2.41
SVM	FullNN-Inst	$\mathbf{5.34 \pm 1.06}$	7.98 ± 2.36
SVM	3P-CofiRank	3.48 ± 0.35	4.73 ± 1.59
SVM	3P-FullNN-Inst	$\mathbf{3.34 \pm 0.21}$	4.56 ± 1.46
NNet	CofiRank	6.39 ± 1.01	7.63 ± 2.20
NNet	FullNN-Inst	5.90 ± 0.78	7.70 ± 2.12
NNet	3P-CofiRank	3.62 ± 0.35	4.58 ± 1.35
NNet	3P-FullNN-Inst	3.43 ± 0.20	$\mathbf{4.36 \pm 1.31}$

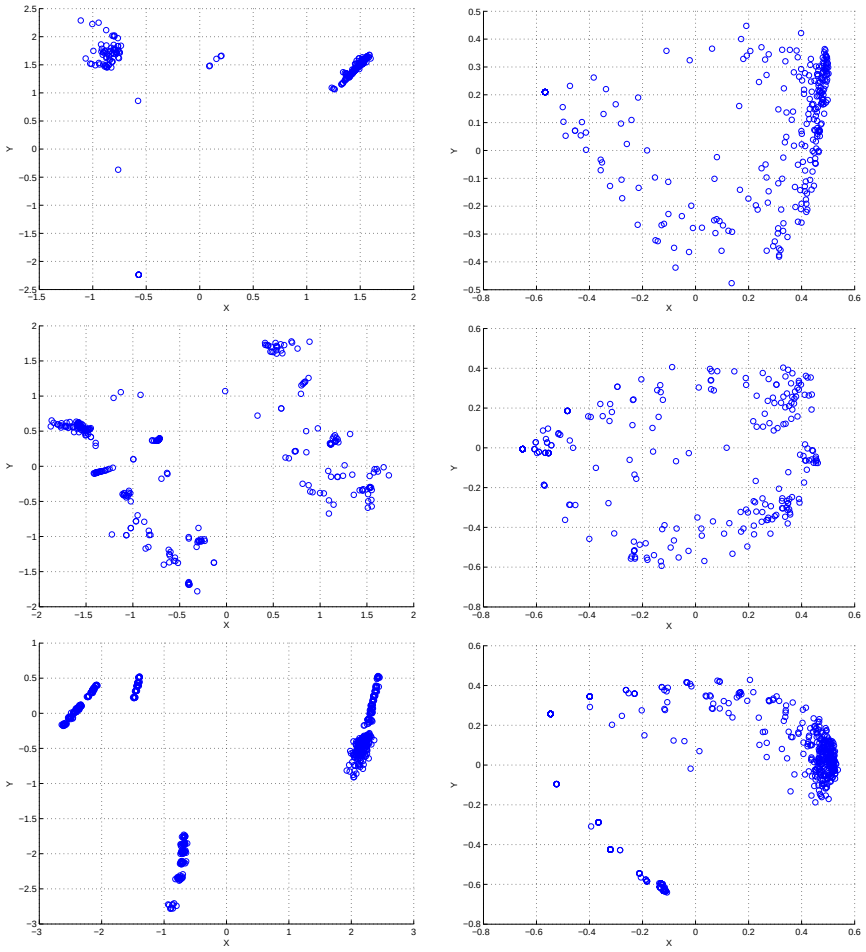


Figure 5: Multi-dimensional scaling of SAT-Phase2, based on the 54 SATzilla features (left) and the first 10 latent features (right): APP problems on the top row, CRF on the middle row and RND on the bottom row.

that some differences based on the initial features are not so relevant in the perspective of the SAT solvers.

6.3.2 CSP 2008 domain

Likewise, Fig. 6 reports the multi-dimensional scaling of the problems in the CSP-Global, CSP-k-AryInt, CSP-2-AryExt and CSP-k-AryExt using the initial 36 SATzilla features (left) and the first 10 latent features (right).

6.3.3 BBOB 2012 domain

Fig. 7 reports the multi-dimensional scaling of the problem instances, using the initial 9 features (left) and the first 10 latent features (right).

The MDS view can be further inspected by grouping together the problem instances based on the same optimization objective (Fig. 7, bottom left) and by grouping the

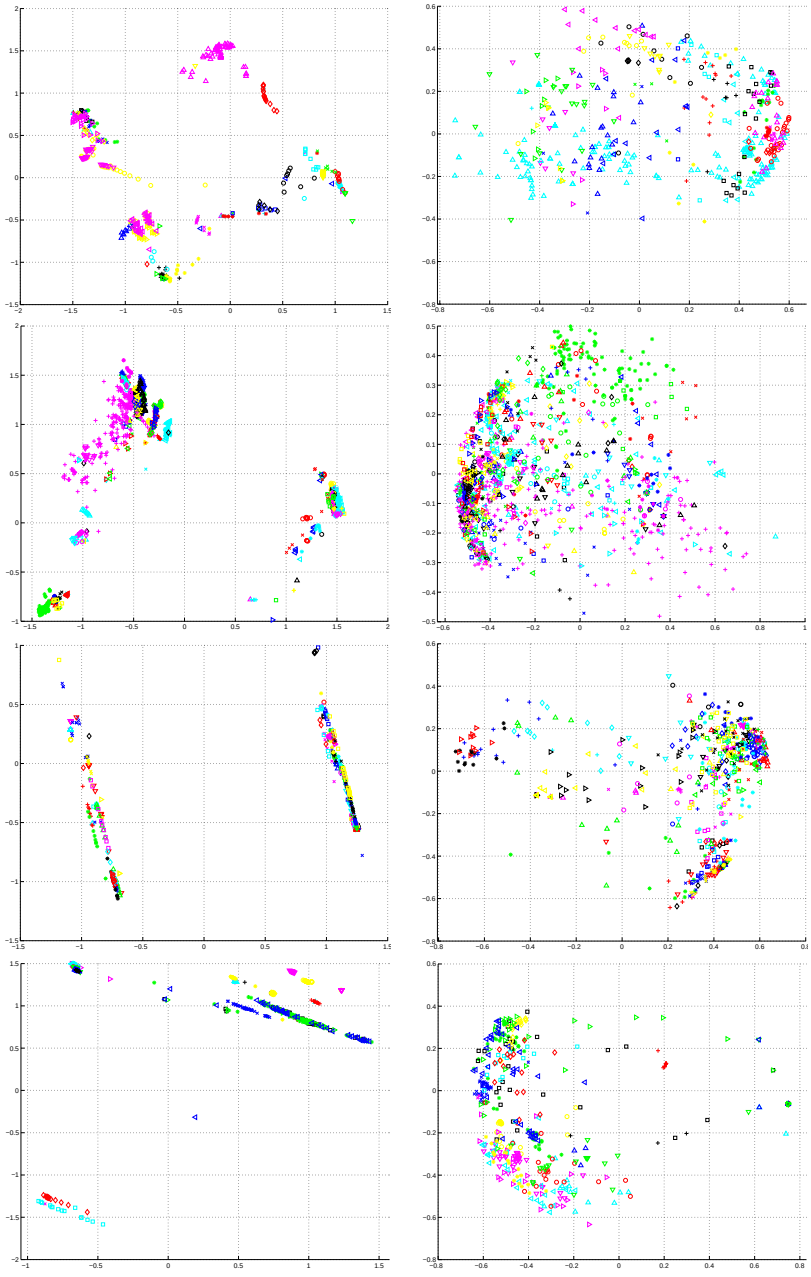


Figure 6: Multi-dimensional scaling of CSP 2008, based on the 36 SATzilla features (left) and the first 10 latent features (right): from CSP-Global (top row) to CSP-k-AryInt, CSP-2-AryExt and CSP-k-AryExt (bottom row).

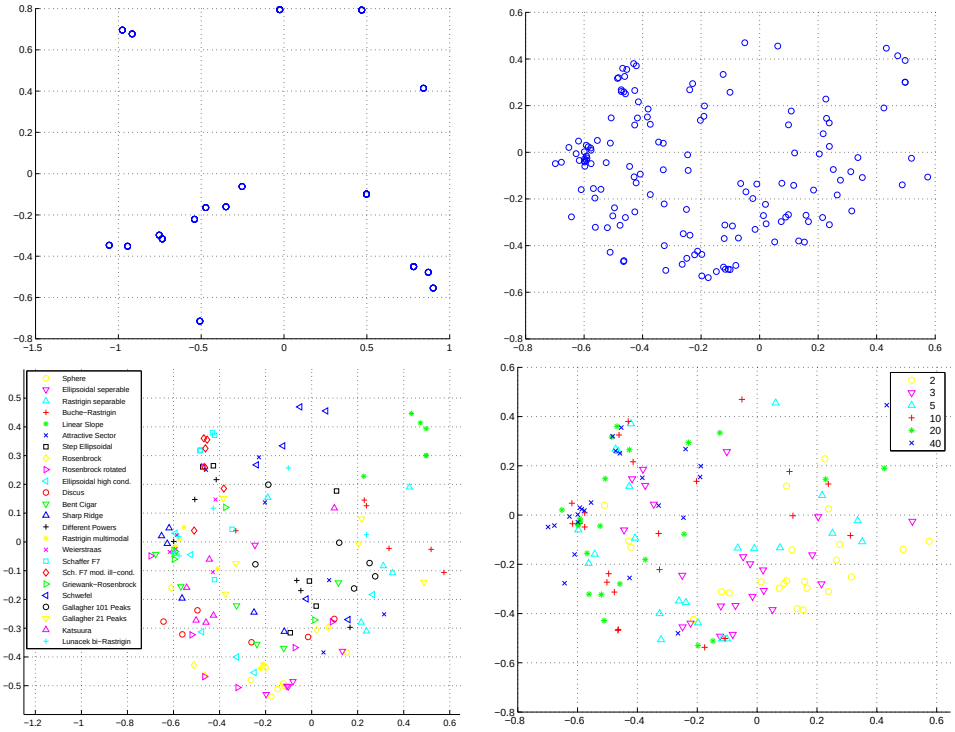


Figure 7: Multi-dimensional scaling of BBOB 2012, based on the 9 initial features (top row, left) and the first 10 latent features (top row, right). The problem instances are grouped by optimization objectives (bottom row, left) and by search space dimension (bottom row, right).

problem instances with same search space dimension (Fig. 7, bottom right), suggesting that the dimension is more relevant than the optimization objective on the considered benchmarks, regarding the algorithm selection issue.

7 Discussion and Perspectives

The experimental validation on three application domains reported in the previous section establishes a successful proof of principle for the merits of the ARS approach regarding the algorithm selection issue.

The main lesson learned from the experiments is twofold. On the one hand, it suggests that collaborative filtering does offer a generic answer for the algorithm selection problem. With respect to the state of the art in the SAT and CSP domains, it shows that satisfactory results can be obtained along the collaborative filtering principles, using a fraction of the training information used in e.g. [XHHLB12a, OHH+08]: ARS improves on the single-best baseline although it uses as little as 10 or 20% of the overall matrix reporting the relative performance of each algorithm in the portfolio on each benchmark problem instance.

The main novelty compared to SAT and CSP algorithm selection, first investigated by [SHG09, SSH+10], is to rely on the latent features to efficiently solve the cold-start problem. The learning of latent features compensates to some extent for the inadequacies of the initial features manually designed to describe the problem instance space; it also opens some perspectives for further research and the assessment of benchmarks (more below). Compared to Matchbox [SHG09, SSH+10], ARS provides a simpler framework to translate the initial algorithm performances onto rank categories: the number of such categories is not limited *a priori* and no prior knowledge is required to adjust the thresholds among the categories. Secondly and most importantly, ARS uses a full-fledged surrogate approach to learn the mapping from the initial onto the latent features, whereas Matchbox is limited to linear mappings. Note that the ARS code and experiment data are made available for the reproducibility of the experiments.

A second lesson regards the difference between the collaborative filtering and the algorithm selection settings. Specifically, in the current state of the art, algorithm selection is (not yet) a big data problem: the number of benchmark problems is a few thousands at best, and the number of algorithms in the portfolio is also limited. This difference might explain the fact that the memory-based strategy outperforms the model-based strategy in the considered ranges of incompleteness: there is not enough information in the data to recover from very high incompleteness levels.

Several research perspectives are opened by the presented approach. An on-going work concerns the use of ARS to handle algorithm selection on top of the Weka [HFH+09] and RapidMiner [HK13, SVKB13] platforms; in particular, the results should give some insights into the quality of the initial descriptive features, pioneered in [Kal02, KH03].

A short-term perspective concerns the ARS extension to the hyper-parameter tuning problem.As already said, the selection of the parameter setting best suited to a given problem instance is tightly related to the algorithm selection issue, with a main difference: the configuration space (the space of hyper-parameter settings) is usually continuous, preventing any exhaustive search. The surrogate model mapping the descriptive features of the problem instance and the hyper-parameter values onto the latent features must thus be extended to support a sequential model-based optimization

scheme, taking inspiration from [Hoo12, BBKS13].

In the mid-term, a second research perspective concerns the exploitation of the latent feature space to provide a rigorous methodology in order to assess the coverage of a domain benchmark and its diversity w.r.t. an algorithm portfolio. Symmetrically, the latent features can also be used to inspect the contribution of a new algorithm and its "niche" w.r.t. the problem instance landscape. It must be emphasized that the latent features do not depend in any way on the initial descriptive features. The accuracy of the surrogate model, mapping the initial onto the latent features, thus provides some insights into the quality and shortcomings of the initial features. Eventually, ill-predicted clusters in the latent space could lead to designing new descriptive features relevant to algorithm selection, thereby contributing to better understanding of the critical specifics of the problem instances.

In the long-term, the presented approach suggests that any algorithmic platform should involve an archive functionality, reporting the performance of some algorithm/configuration/workflows on the problem instances tackled by the user. On the one hand, this archive could support algorithm selection, enabling the lifelong learning of the algorithm platform and allowing the user to reach peak performance on her specific problem distribution. On the other hand, these archives could be shared among different users with no or little breach of confidentiality (a problem instance being associated a mere identifier). Sharing the archive information on a large scale could provide a general picture of the algorithm behavior on the problem space, and contribute to the emergence of a "folksonomy" of the problem instances.

References

[AAHO11] Ludovic Arnold, Anne Auger, Nikolaus Hansen, and Yann Ollivier. Information-geometric optimization algorithms: A unifying picture via invariance principles. *arXiv preprint arXiv:1106.3708*, 2011.

[Ahn08] Hyung Jun Ahn. A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences*, 178(1):37–51, 2008.

[AHS10] Alejandro Arbelaez, Youssef Hamadi, and Michèle Sebag. Continuous search in constraint programming. In *22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010*, pages 53–60. IEEE Computer Society, 2010.

[AHS12] Anne Auger, Nikolaus Hansen, and Marc Schoenauer. Benchmarking of continuous black box optimization algorithms. *Evolutionary Computation*, 20(4):481, 2012.

[AS09] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI'09)*, pages 399–404. Morgan Kaufmann Publishers Inc., 2009.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[BBKS13] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michèle Sebag. Collaborative hyperparameter tuning. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, volume 28 of *JMLR Proceedings*, pages 199–207. JMLR.org, 2013.

[BG05] Ingwer Borg and Patrick J.F. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Verlag, 2005.

[BGCSV09] Pavel Brazdil, Christophe G. Giraud-Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning - Applications to Data Mining*. Cognitive Technologies. Springer, 2009.

[BL07] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of of the 13th International Conference on Knowledge Discovery and Data Mining (KDD'07) Cup and Workshop*, volume 2007, page 35, 2007.

[BP98] D. Billsus and M.J. Pazzani. Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine Learning (ICML'98)*, pages 46–54, 1998.

[BS00] Pavel Brazdil and Carlos Soares. A comparison of ranking methods for classification algorithm selection. In Ramon López de Mántaras and Enric Plaza, editors, *Machine Learning: ECML 2000, 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 63–74. Springer, 2000.

[DDF+90] Scott Deerwester, Susan T. Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[EFW+06] Susan L. Epstein, Eugene C. Freuder, Richard Wallace, Anton Morozov, and Bruce Samuels. The adaptive constraint engine. In Pascal Van Hentenryck, editor, *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP'02)*, volume 2470 of *LNCS*, pages 525–540. Springer, 2006.

[GS01] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artif. Intell.*, 126(1-2):43–62, 2001.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11:10–18, 2009.

[HHHLB06] Frank Hutter, Youssef Hamadi, Holger H. Hoos, and Kevin Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In Frédéric Benhamou, editor, *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006*, volume 4204 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2006.

[HHLB11] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization, LION-5*, volume

6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer, 2011.

[HHLBS09a] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.

[HHLBS09b] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An automatic algorithm configuration framework. *J. Artif. Intell. Res. (JAIR)*, 36:267–306, 2009.

[HK13] Markus Hofmann and Ralf Klinkenberg. *RapidMiner: Data Mining Use Cases and Business Analytics Applications*. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series, CRC Press, 2013.

[HMI09] Verena Heidrich-Meisner and Christian Igel. Hoeffding and bernstein races for selecting policies in evolutionary direct policy search. In Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman, editors, *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009*, volume 382 of *ACM International Conference Proceeding Series*, page 51. ACM, 2009.

[Hoo12] Holger H. Hoos. Programming by optimization. *Commun. ACM*, 55(2):70–80, 2012.

[HRM+11] Nikolaus Hansen, Raymond Ros, Nikolas Mauny, Marc Schoenauer, and Anne Auger. Impacts of invariance in search: When cma-es and pso face ill-conditioned and non-separable problems. *Appl. Soft Comput.*, 11(8):5755–5769, 2011.

[JK00] Kalervo Järvelin and Jaana Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *SIGIR*, pages 41–48, 2000.

[Kal02] Alexandros Kalousis. *Algorithm selection via meta-learning*. PhD thesis, University of Geveve, 2002.

[KH03] Alexandros Kalousis and Melanie Hilario. Representational issues in meta-learning. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003)*, pages 313–320. AAAI Press, 2003.

[KMS+11] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *Principles and Practice of Constraint Programming (CP'11)*, pages 454–469. Springer, 2011.

[Kot12] Lars Kotthoff. Hybrid regression-classification models for algorithm selection. In Luc De Raedt, Christian Bessière, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter J. F. Lucas, editors, *ECAI 2012 - 20th European Conference on Artificial Intelligence*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 480–485. IOS Press, 2012.

[MS10] Prem Melville and Vikas Sindhwani. Recommender systems. *Encyclopedia of Machine Learning*, 1:829–838, 2010.

[NE07] Volker Nannen and A. E. Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 975–980, 2007.

[OHH+08] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. OSullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.

[PBGC00] Bernhard Pfahringer, Hilan Bensusan, and Christophe G. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 743–750. Morgan Kaufmann, 2000.

[Ric76] J.R. Rice. The algorithm selection problem. *Advances in computers*, 15:65–118, 1976.

[RIS+94] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[Rou11] O. Roussel. Description of ppfolio. Solver description `www.cril.univ-artois.fr/~roussel/ppfolio/solver1.pdf`, 2011.

[SB98] R.S. Sutton and A. G. Barto. *Reinforcement learning*. MIT Press, 1998.

[SHG09] David H. Stern, Ralf Herbrich, and Thore Graepel. Matchbox: large scale online bayesian recommendations. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web, WWW 2009*, pages 111–120. ACM, 2009.

[SK09] X. Su and T.M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009:19, 2009.

[SM07] H. Samulowitz and R. Memisevic. Learning to solve QBF. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'07*, volume 22, pages 255–360, 2007.

[SM10] Bryan Silverthorn and Risto Miikkulainen. Latent class models for algorithm portfolio methods. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*, 2010.

[SRJ05] N. Srebro, J.D.M. Rennie, and T. Jaakkola. Maximum-margin matrix factorization. *Advances in neural information processing systems*, 17(5):1329–1336, 2005.

[SS08] M. Streeter and S.F. Smith. New techniques for algorithm portfolio design. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI'08)*, volume 10, page 26, 2008.

[SSH+10] David H. Stern, Horst Samulowitz, Ralf Herbrich, Thore Graepel, Luca Pulina, and Armando Tacchella. Collaborative expert portfolio management. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press, 2010.

[SVKB13] Floarea Serban, Joaquin Vanschoren, Jörg-Uwe Kietz, and Abraham Bernstein. A survey of intelligent assistants for data analysis. *ACM Comput. Surv.*, 45(3):31, 2013.

[Sze10] Csaba Szepesvari. *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010.

[THHLB12] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms. *CoRR*, abs/1208.3719, 2012.

[TSVL07a] C.H. Teo, A. Smola, SVN Vishwanathan, and Q.V. Le. A scalable modular convex solver for regularized risk minimization. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 727–736. ACM, 2007.

[TSVL07b] Choon Hui Teo, Alex J. Smola, S. V. N. Vishwanathan, and Quoc V. Le. A scalable modular convex solver for regularized risk minimization. In Pavel Berkhin, Rich Caruana, and Xindong Wu, editors, *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 727–736. ACM, 2007.

[TTB08] Naoyuki Tamura, Tomoya Tanjo, and Mutsunori Banbara. System description of a sat-based csp solver sugar. In *Proceedings of the 3rd International CSP Solver Competition*, pages 71–75, 2008.

[VVSW09] Julien Villemonteix, Emmanuel Vázquez, Maryan Sidorkiewicz, and Eric Walter. Global optimization of expensive-to-evaluate functions: an empirical comparison of two sampling criteria. *J. Global Optimization*, 43(2-3):373–389, 2009.

[WKLS07] M. Weimer, A. Karatzoglou, Q. Le, and A. Smola. Cofirank-maximum margin matrix factorization for collaborative ranking. In *Proceedings of the 21st Annual Conference on Neural Information Processing Systems (NIPS'07)*, pages 222–230, 2007.

[XHHLB08] L. Xu, F. Hutter, H.H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32(1):565–606, 2008.

[XHHLB12a] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Evaluating component solver contributions to portfolio-based algorithm selectors. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012*, volume 7317 of

Lecture Notes in Computer Science, pages 228–241. Springer Verlag, 2012.

[XHHLB12b] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Features for SAT. University of British Columbia, http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/Report_SAT_features.pdf, 2012.

[YE12] X. Yun and S. L. Epstein. Learning algorithm portfolios for parallel execution. In Y. Hamadi and M. Schoenauer, editors, *Proceedings of the 6th Learning and Intelligent OptimizatioN Conference (LION'12)*, volume 7219 of *LNCS*, pages 323–338. Springer, 2012.

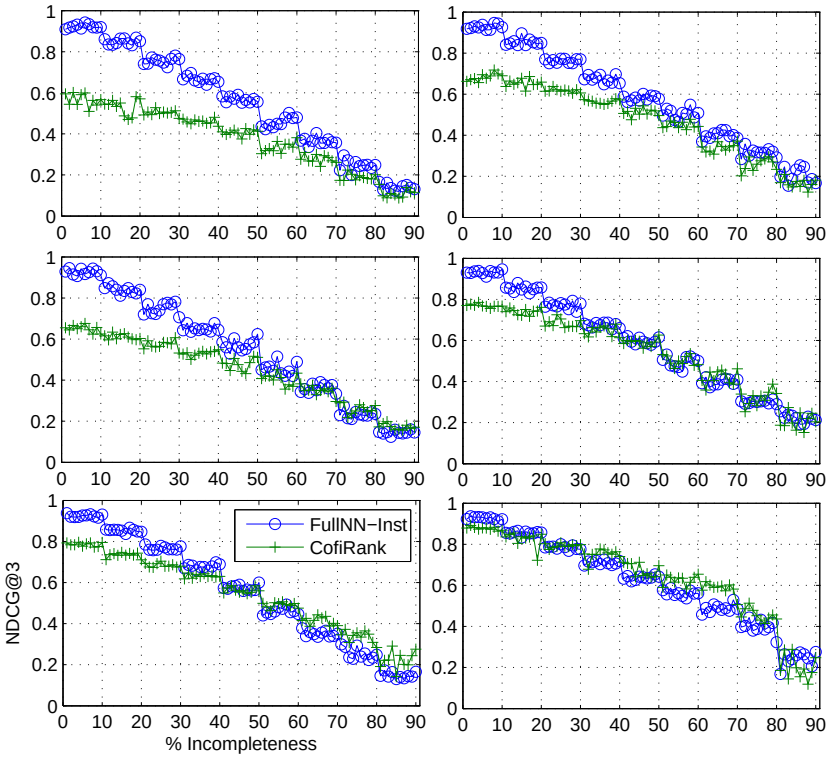


Figure 8: Comparison of memory-based FullNN and strategy-based CofiRank using the NDCG@3 indicator on the SAT 2011 domain. Top to bottom: categories APP, CRF and RND. Left: SAT-Phase1; right: SAT-Phase2.

Appendices

A Detailed results of ARS on SAT 2011

A.1 Matrix completion

As shown in section 6.1 (Figs. 1 and 2), ARS outperforms the single best algorithm baseline. The detailed comparison of the memory-based and model-based approaches, respectively FullNN and CofiRank, is analyzed using the NDCG@3 indicator (Fig. 8). As could have been expected, FullNN outperforms CofiRank for low levels of incompleteness, especially for the APP and CRF categories; CofiRank catches up and outperforms FullNN for high levels of incompleteness.

A.2 Cold start

The ARS performance in terms of percentage of solved instances on the Phase 1 and Phase 2 problems is displayed on respectively Figure 9 and 10, demonstrating that it outperforms the single-best algorithm. For the P2-CRF and P2-RND datasets, a performance drop is observed when the incompleteness level of the training set decreases from 80% to 90%.

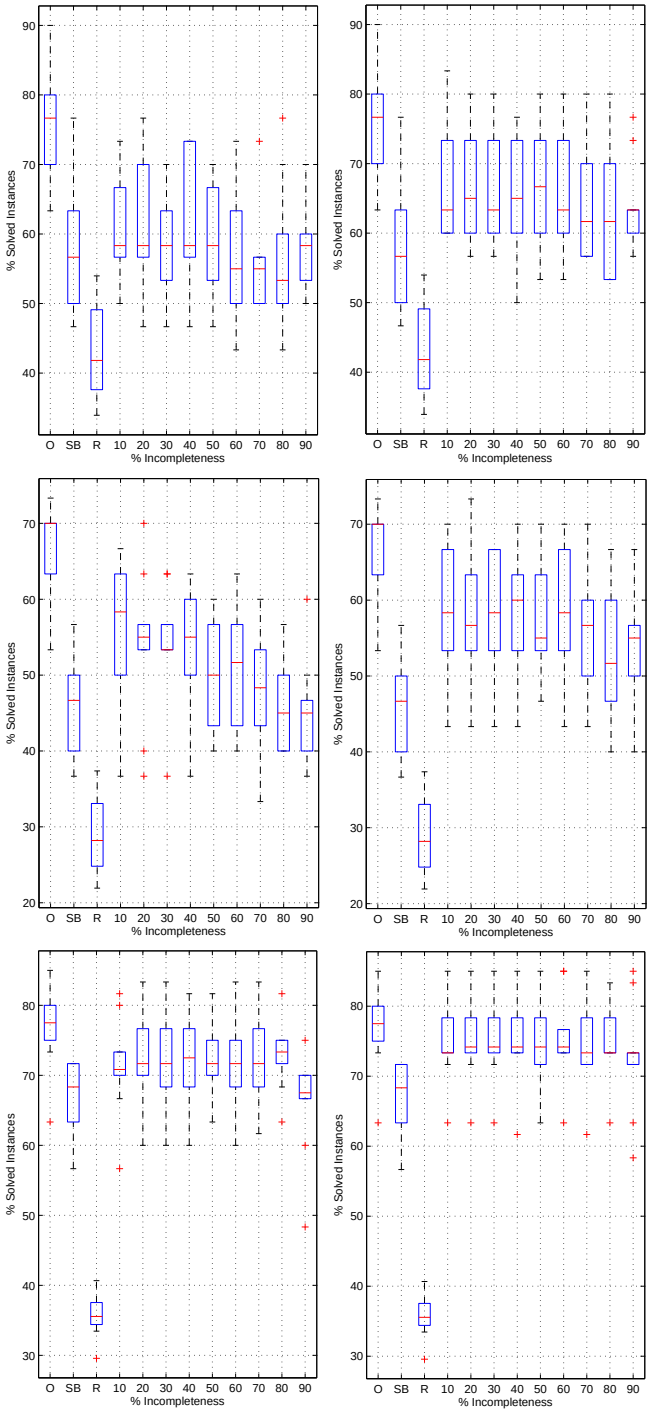


Figure 9: ARS performance on the Phase 1 cold start benchmarks: % solved instances using single PC (left) and using 3 parallel PCs (right) for SAT categories APP (top), CRF (middle) and RND (bottom) vs % incompleteness (sparsity).

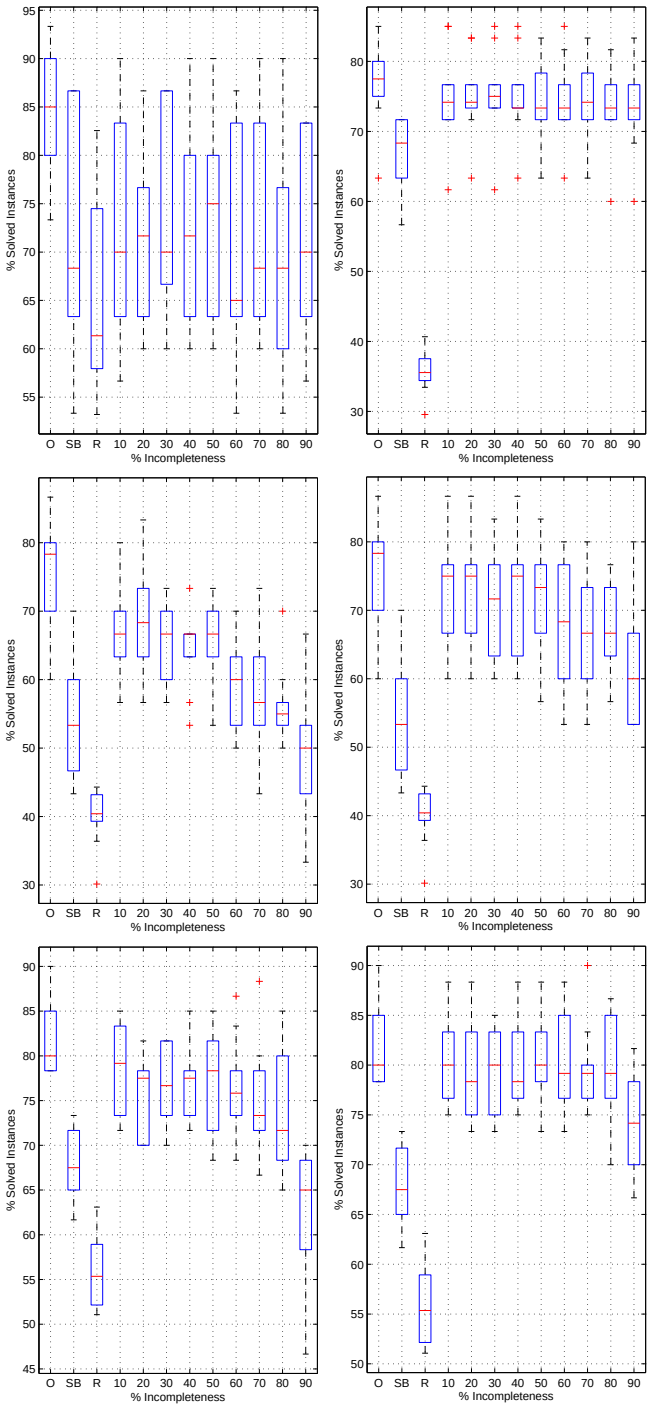


Figure 10: ARS performance on the Phase 2 cold start benchmarks: % solved instances using single PC (left) and using 3 parallel PCs (right) for SAT categories APP (top), CRF (middle) and RND (bottom) vs % incompleteness (sparsity).

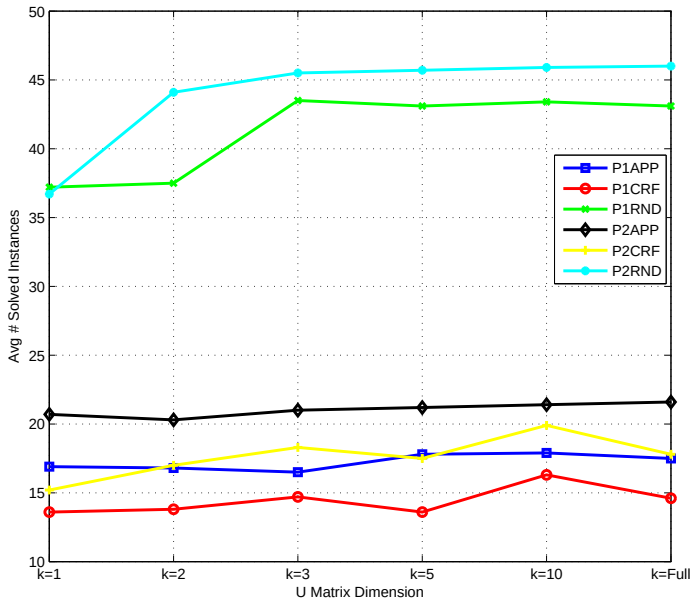


Figure 11: Cold start: Sensitivity of the SVM-FullNN w.r.t. the number k of latent features

A.3 Sensitivity to the number of latent factors

The sensitivity of the cold-start approach to the dimension k of the low-rank approximation is studied in Figure 11, regarding the ARS based on SVM-regression. Interestingly, very good performances are achieved for low k values ($k \leq 5$) and a performance plateau is observed for higher values. For most settings, except P2-CRF and P1-APP, a value of $k = 3$ is enough to almost reach the top performances. This result suggests that the intrinsic dimensionality of the problem instance space is low.

B Detailed results of ARS on BBOB 2012

B.1 Matrix completion

As shown in section 6.1 (Fig. 4), ARS outperforms the single best algorithm baseline for incompleteness level up to 90%. The detailed comparison of the memory-based and model-based approaches, respectively FullNN and CofiRank, is displayed on Fig. 12.

B.2 Cold start

The ARS performance in terms of average rank of the recommended algorithm versus the incompleteness level is displayed on Fig. 13 for the 2160 runs (left) and for the average setting (140 pbs, right).

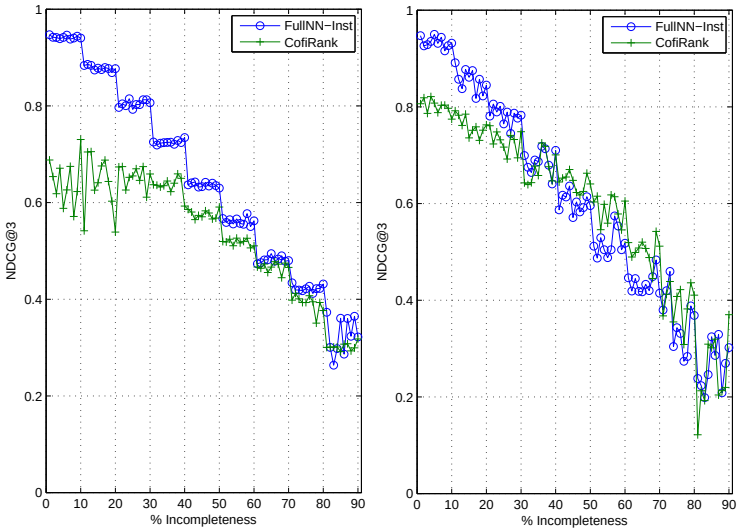


Figure 12: Comparison of memory-based FullNN and strategy-based CofiRank using the NDCG@3 indicator on the BBOB 2012 domain, 2160 instances (left) and 144 (averaged, right).

C Detailed results of ARS on CSP 2008

C.1 Matrix completion

As shown in section 6.1 (Fig. 3), ARS outperforms the single best algorithm baseline on all problem categories when the incompleteness level is sufficiently low (less than 80% on CSP-Global, 70% on CSP-2-AryExt and CSP-k-AryInt, and 90% on CSP-k-AryExt). The detailed comparison of the memory-based and model-based approaches, respectively FullNN and CofiRank, is displayed on Fig. 14. As in the SAT 2011 domain, FullNN outperforms CofiRank for low levels of incompleteness while CofiRank catches up and outperforms FullNN for high levels of incompleteness.

C.2 Cold Start

The ARS performance (NNEt based on the FullNN-Inst) in terms of percentage of solved instances on the CSP 2008 domain is displayed on Figure 15, distinguishing the CSP-Global (top row), the CSP-k-AryInt (2nd row), CSP-2-AryExt (3rd row) and CSP-k-AryExt (bottom row) categories.

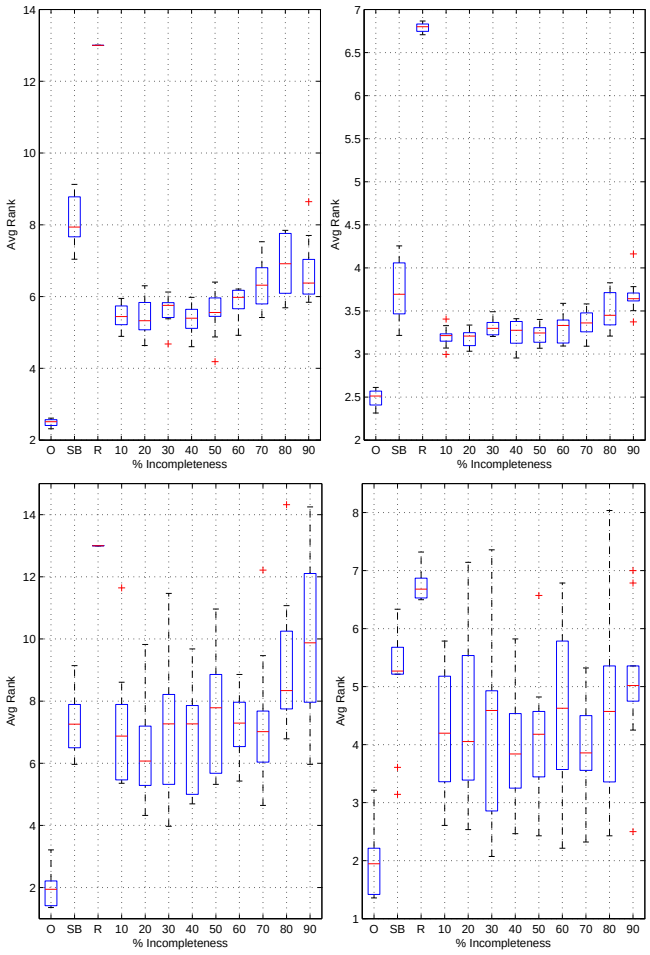


Figure 13: ARS performance on the BBOB 2012 cold start benchmark: average rank of the recommended algorithm for the NNet-CofiRank (left) and 3P-NNet-FullNN-Inst (right) on the 2160 instances (top row) and 144 problems (bottom row).

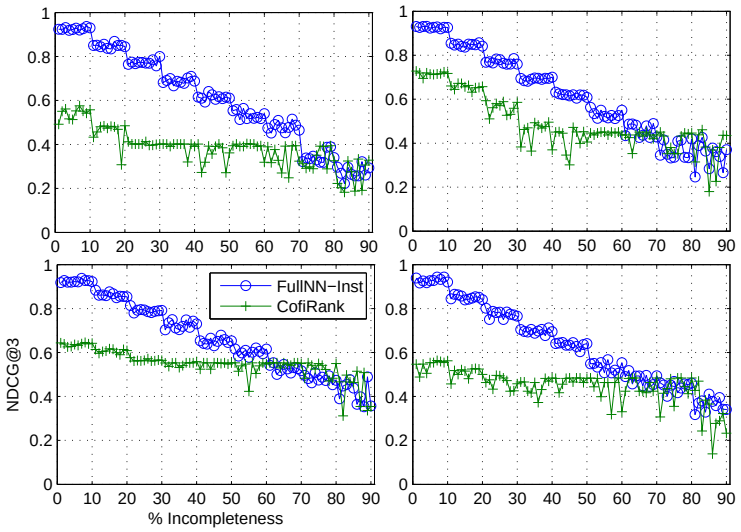


Figure 14: NDCG@3 comparative performance of FullNNInst and CofiRank on the CSP 2008 domain: CSP-Global (top left), CSP-k-AryInt (top right), CSP-2-AryExt (bottom left) and CSP-k-AryExt (bottom right).

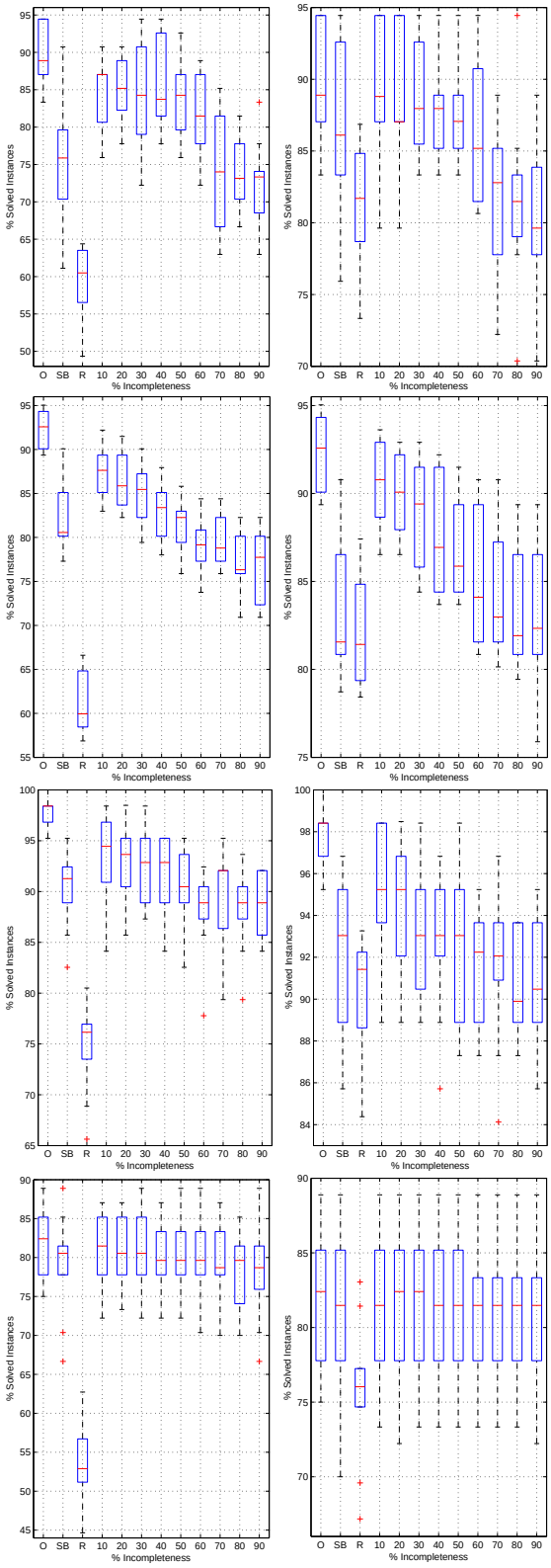
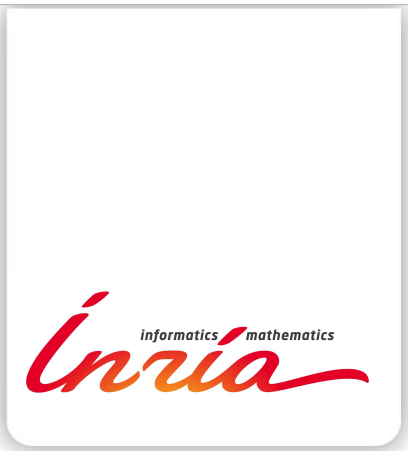


Figure 15: Cold start: Percentage of solved instances comparatively to the baselines, on CSP-Global (top, with NNet-FullNN-Inst); on CSP-k-AryInt (2nd row, using SVM-FullNN-Inst); on CSP-2-AryExt (3rd row, with NNet-FullNN-Inst); on CSP-k-AryExt (bottom row, with SVM-FullNN-Inst).



RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399