Review of the Algorithm Selection

Rui Rodrigues[1] and José Carlos Monteiro[2]

[1]Praceta da Amizade, torre 308, 8ºB,
2735-387 Mira-Sintra, Portugal
ruirodrigues1971@gmail.com
[2]Instituto Superior Técnico
Av. Rovisco Pais, 1049-001 Lisboa
jcm@inesc-id.pt

Abstract. The algorithm selection problem consists of choosing, from a predefined set, the best algorithm to run in a given instance of a an optimization problem. Typically, the objective is either to reduce the time to obtain a solution and/or to maximize the quality of this solution. The algorithm selection problem is a rich area that uses many ideas from other fields. In this paper, we make a review of the most significant contributions in this area and try to extrapolate future tendencies. We make an attempt at classifying this sample of works, to give a better perspective of the methods proposed so far. With this objective, we try to unify the terms more commonly used and propose a new nomenclature in some areas. We believe that the proposed taxonomy gives a better perspective of the field and that it will help contextualize future research.

Keywords: Algorithm Selection, Machine Learning, Meta-algorithm.

1. Introduction

The algorithm selection problem consists of choosing, from a predefined set, the best algorithm to run on a given instance of an optimization problem [1]. If a methodology were available that could determine beforehand the best optimization algorithm to apply for each problem instance, faster and better solutions in scientific and engineering problems would be obtained.

Many different approaches have been proposed to address this problem. Some techniques try to determine what are the best parameters values of the algorithm using prior knowledge obtained in previous tests. Other techniques make dynamic decisions as to what are the chances to achieve a good solution if continues running, or if it is better to restart the algorithm, or even to chose a new algorithm.

The Meta-algorithm that deals with the algorithm selection problem, tries to encounter the algorithm that best solve a particular instance of the problem. It creates, adapts or chooses the algorithm that can potentially be the best (using the information available) for a particular instance, and using a particular performance indicator.

Typical objectives pursued when analyzing the algorithm selection problem are:

- What are the best parameters values for a class of algorithms to a particular class of problems?
- What is the best algorithm in a poll of previous selected algorithms?

- How much time will be dedicated to a particular algorithm? When is the best moment to restart?
- What are the best features to use?
- Can the features of the search space detected during run time help the search?
- Is it necessary to use rich knowledge to obtain good decisions?
- Changing the focus (codifications of the solution, domain to explore, etc.) of each algorithm can also be seen as a change of the algorithm?

There are two main metrics to evaluate an algorithm in this context. Typically, we can divide in the type of problems that the algorithm tries to resolve. In decision problems (like SAT problems) the focus is in speed, and the algorithm selection problem tries to encounter the fastest algorithm to achieve the solution, or when is the best moment to restart a particular algorithm. In optimization problems, the typical focus is in the quality of the solution. Nevertheless, some algorithm selection problem in optimization problems context also uses some speed information to help in deciding the best algorithm in a particular moment. In the following table shows some of the work done in the field divided in these two types of problems.

Table 1. Type of Applications

Optimization Problems	Decision Problems
[2],[3],[4],[5],[6],[7], [8],[9],[10],[11],[12],[13]	[14],[15],[16],[17],[18], [19],[20],[21],[22],[23],

We can see that both types of applications have a similar number of works, considering that we were not biased to any type of problem in our sample of the work done in the field.
The outline of this paper is as follows. In Section 2, we give a formal definition of the selecting algorithm problem. Next, in Section 3, we present high-level approaches to the problem, from brute force to machine learning approaches. In Section 4, we organize the papers we reviewed and define the taxonomy used. Within Section 5, we classify the sample of field papers selected for this review using the taxonomy proposed, allowing us to organize the papers in some interesting clusters. Finally, in section 6, we discuss future tendencies of this rich field of research.

2. Definition of the algorithm selection problem

We can see the algorithm selection problem as having a pool of algorithms $A=\{a_1,a_2,...,a_n\}$, and an instance problem p to resolve. The selection algorithm, F, has to choose the best algorithm to use [1]. This selection algorithm has to use some features to help the decision. These features can be from small number of simple features like the velocity of improvement to a huge number of complex features – increasing the computation. These features can be static features that do not change

over run-time, or can be dynamic features – force F to decide many times in run-time. Typically the feature more used are the instance problem features, the algorithm features, the search space features and some progress features (like speed of progress)

$$a_i=F(f_1,f_2,\ldots,f_n),\ f_i \text{ some features.} \quad (1)$$

As the computational power as improved – and studies about this problem increased, we can see our problem has a selection algorithm function F that generates group S of algorithms. We are interested in new synergies obtained when we have many algorithms running – portfolio [24] is one example of this strategy.

$$S = \{a_k,a_j,...,a_t\} = F(f_1,f_2,\ldots,f_n). \quad (2)$$

The set S can be a combination of different algorithm that runs in parallel, but also can have the same algorithm repeated in a number of times proportion to the strength of that algorithm. The set S also could represent a sequential order of algorithms. Obtained in offline manner (by previous tests) or in an online manner reacting to the changing features as the present chosen algorithm evolves in the search space.

The function F has to choose S that maximizes some criteria C (running time, final solution quality) based on the actual available features. The perfect function F is difficult to obtain in real problems with interest. We have to use some machine learning strategies (neural networks, ridge regression, Bayesian networks, reinforcement learning, etc.) that use some statistic information or use some user heurist rules to obtain an approximation of the perfect function F.

$$\begin{cases} S = F(.) \\ \arg\max_S C(S) \end{cases} \quad (3)$$

C(S) – The result (time, evaluation of the solution, etc.) of using the set S of algorithms with the available resources and strategies using the criteria C function.
When we share information, the problem becomes more complex, but can bring new synergies that can lead to better solutions (for example using global and local search). In problems that depend heavily in the first seed of the algorithm or in random strategies (example: Genetic Algorithms, Simulated Annealing, etc.) then the problem complexity increases dramatically.

3. General Strategies

Some works have tried to encounter the hardness of an instance problem. It is important to have an idea of the hardness of a problem – how these hardness progresses – to be possible to compare different strategies or to identify where the interesting areas of research are. Many aspects affect the hardness of a particular class of problems, could be de dimension of some features or relation between one or more features. Typically, it is necessary to know the most importance features of a problem. If we find the most important features, we could have fewer features in our selection

algorithm problem – maintaining the more important ones, to reduce the complexity of the problem.

The most naïve effort to know the relation between the hardness of a problem confronted with a set of algorithm is using brute force. We have the class problem that we want to resolve (typically we can parameterize all the instances of the class) and make a number huge of experiences with the algorithms available, changing parameters of the algorithms and the instances of the class of problems. We chose the best algorithm. This option consumes many resources and is difficult to extrapolate what is the best algorithm to new unobserved instances of the problem. Some researchers looking to this result try to create some heuristic rules based in the observed results.

A strategy that has the same problem of difficult to extrapolate to new instance, but is very less expensive with the resources and use statistical techniques to help in finding the best algorithm to a instance problem is the racing technique [5],[7],[13]. These techniques do not use the parameters of the algorithm or the features of the problems instances to decide what the best algorithm is. Just look to the results obtained to discard the bad algorithms with some statistical rules – low knowledge approach. Has a number of advantages because can deal with non-numeric features (for example nominal and order variables) that are very difficult to be used with the most learning algorithms. We can use some machine leaning techniques that can explore the parameter space and use racing techniques in the place of the brute force method to moderate size of candidates to train as in [13].

Another strategy is using some process that can learn how evolves the hardness with the algorithms and extrapolate what is the best algorithm for a particular instance/algorithm that was not used in the training phase. In this strategy we can use machine learning [25],[26],[27],[28],[29] to try to catch this relation between instances of a problem and the algorithms. Some of the concrete learning strategies used in the selection algorithm problem are in the Table 2.

The Table 2 show only part of the work done, other works exist; we try to show a good sample of the work in this particular field. It is obvious that heuristic approach is less use in opposing with approaches that are more automatic – machine learning.

Table 2. Typical Concrete Learning Strategies

Learning	References
Heuristic	[19], [22],[8]
Bayesian Networks	[9], [10]
Statistical Learning	[21],[22],[2],[3],[4],[23],[5], [6], [7],[12],[15],[13],[18]
Decision Tree	[16],[17]
Neural Networks	[11]
Dynamic Programming	[4]

4. Field Taxonomy

In this section, we will try to organize the area. We use some suggestions to a common classification used in previous works but sparse in different papers. Some terms that we purposed are new in the field but are important in the opinion of the authors to future search. The main goal is to help the future works in the field to characterize better the research and help the search of related works.

Table 3. Nomenclature to use in the Algorithm Selection Problem

Type	Name	Comments
Knowledge	High-Knowledge	Based on the characteristics of the problem/algorithm, i.e., in their parameters.
	Low-knowledge	Based in indirect information, like speed of the algorithm or evaluation, or in few features[1] of the problem and algorithm
Learning	Offline	The process of learning which of the algorithms to run is made in previous tests.
	Online	The process of learning which of the algorithms to run is made in run-time.
	Life Learning	The process of learning which of the algorithms to run is made using offline and online knowledge.
Reactivity	Reactive	Reacts in run time, can change the algorithm that is running
	No reactive	System decides what algorithm to run and cannot change in run time
Cooperation	Cooperative	Different algorithm that cooperate, and share efforts to obtain the final result
	Competitive	Different algorithm run as isolated algorithms, do not share efforts
Architecture	Parallel	Many algorithms running at the same time
	Sequential	The algorithms are ordered to run in sequential order
	Isolated	Just one algorithm running in each run

The Table 3 resumes our recommend taxonomy to use in the field, which we will detail more in the following sections. This table tries to resume and expand the common taxonomy of previous works.

1.1 Type of Knowledge

We have two different approaches. One approach tries to gather the maximum information and try to predict the best algorithm based in that information – high knowledge approach. The other approach tries to see what are the best algorithms

[1] This is a fuzzy frontier because we can use few parameters with huge complexity (has to be computed) or in contrast many simple parameters that do not need any complex calculations.

based in simple features like the change of evaluation function. The first option has the advantage of have more information the second the advantage to be more simple to used to a more vast range of problems diminishes the need of human expertise or the changes need to be make to adapt to new class of problems.
The high knowledge approach tries to see typically the relationship between the structure of the problem and algorithm behavior. Forms one important basis to decide what is the best algorithm to use. This approach tries to see how different features of the problem and/or features of the algorithm related to choose the best algorithm.
As an example of high knowledge strategy we can see in Roberts et al. [16] work that was apply to planner's algorithms. They use a lot o features, initial they had 57 features, but they could see that reducing this number (using fastest features) they could obtain similar results; nevertheless continue to have many features.
The low-knowledge use a different type of approach, tries to use less information than the high knowledge approach; normally the information is the evaluation or progress of the evaluation of the principal feature of the problem. In optimization problems, the major features typically relate with the evaluation function. In decision problems, can be the time required to answer yes or no to the decision problem.
In Carchirae et al.[10] they used the low-knowledge approach in the Job-shop scheduling problem. One of the techniques used was to run a set of algorithms during a prediction phase then a Bayesian classifier chose one to run the remainder of time T. They only used the performance of the algorithms to predict what will be the best with that particular instance.
The table 4 resumes the taxonomy of the type knowledge used in the field.

Table 4. Type knowledge

Knowledge	High-Knowledge	Based on the proprieties of the problem/algorithm, i.e., in their parameters.
	Low-knowledge	Based in indirect information, like speed of the algorithm or evaluation, not the intrinsic proprieties of the problem and algorithm

1.2 Typical Learning strategies used

Offline learning strategy clearly separate the learning moment from the run-time moment. Typically, the learning process use huge information that relates the behavior of the algorithms with many features (algorithm, problem and space). This process can use a lot of time. The data used to learn what the best algorithms are, is divide typically in train data and test data. Sometimes the train divides in two groups: train and validation data. The strength of this approach is to try to see all the relevant relations between the algorithms and features.

In Lagoudakis et al. [2], one of the solution they propose to resolve which of the two algorithm do run, was a construction of function that decide what was the best algorithm to use in the Order Statistical Selection Problem. They gather the data necessary to learn this function using many offline experiences (in some experiences they used 2400 randomly generated instances).
Online learning process tries to learn the model in run-time. The selection algorithm learns and decide in run-time, typical do not use so many features as the offline strategy. The strength of this approach is to choose the best algorithm in present space conditions; the selected algorithm can have a poor median performance but a good performance in the present conditions.
In Loonel Lobjois et al. [21], they used instances of a variant of the Value CSP problem (Σ-VCSP). Given an instance to solve and a list of algorithms, they want to know what the best algorithm is. They estimate the running time of each candidate algorithm on the particular instance by running each algorithm a time previous defined; then they select the algorithm that gives the smallest expected running time.
Life learning strategy tries to use offline learning (huge information) and online learning (low information, but more instance information focus), both the strategies are good. Using the two can lead to a better performance, is a strategy that needs more studies.
Offline learning is typical used in high knowledge approach, because of the number of feature needed to learn the models. Online learning approaches are typical related with low-knowledge approach, as they have to use less computational power to learn. The table 5 resumes the taxonomy of the learning types used in the field.

Table 5. Learning types

Learning	Offline	The process of learning which of the algorithms to run is made in previous tests.
	Online	The process of learning which of the algorithms to run is made in run-time.
	Life Learning	The process of learning which of the algorithms to run is made using offline and online learning.

1.3 Type of Reactivity

Reactivity is a feature that enables the algorithm to change its behavior in run time. In the scope of the selection algorithm problem is a quality that can help choosing the best algorithm in the present conditions. We have two types: systems that can change the run-time algorithm based in new information's acquired in run-time and systems that do not change the run-time algorithm based in run-time information.
In Lagoudakis et al. [4], they focus in sorting algorithms. They learn a function that relates the dimension of array and the best algorithm in offline manner. If the instance to be resolve is divided (perhaps using MergeSort), two new instances are created and the algorithm has to react to this new instances and decide what is the best algorithm

to use in each instance. We could have many different algorithms running in sub-instances helping resolving the main instance.
Almost all online learning systems use reactivity to change its behavior in run-time, they are compelled to react as they learn, but reactivity can be used in offline systems to – if space features or other features used in the training phase change during run-time.
The Table 6 resumes the taxonomy of the reactivity used in the field.

Table 6. Type of reactivity

Reactivity	Reactive	Reacts in run time, can change the algorithm that is running
	No reactive	System that decides what algorithm to run and cannot change in run time

1.4 Type of Architecture

The architecture is very important to understand the type of algorithm evolved in the selection problem algorithm. Many authors do not give much importance to this classification, but is important to future search about related works. In the isolate architecture we have always only one algorithm that is chose to run, normally this type architecture is associated with offline learning. Given the fact that we have to decide just one time, it is important to have all the information that we can gather to make the best decision.
In Beck et al. [8] one of their initial experiences was in an isolated architecture. They chose the algorithm to run using simple rules based in the initial performance of the algorithms ("training" phase) and then chose one algorithm to run in rest of the time. We have only one algorithm running in run-time (test phase).
The sequential architecture is less used, but is very interesting also. In this type of architecture, we have only one algorithm running at each time, but the algorithm that is running can change. To be a sequential architecture is also necessary that the algorithm that runs influences the next chosen algorithm. Typical this influence can be change of the search space, new leanings or transmission of the seed to the next algorithm, normally the seed is the best solution obtained by the previous algorithms.
One example of sequential architecture is in Carchrae et al. [9] work. In their work exists a switch algorithm that will decide what will be the new algorithm that runes. The seed of the new algorithm is the solution of the last algorithm.
The parallel architecture, use many algorithms at same time – many authors simulate this propriety. The most popular parallel architecture in this field is the portfolio [24], that we will explain in detail in the next section. The Table resumes the taxonomy of the architectures used in the field.
In Roberts et al. [16] the portfolio begins with a set of planners and ranks them according to the success models – the planners that have more success in the training phase (tree learning) have more chance to be in the initial portfolio.

Table 7. Types of architectures

Architecture	Parallel	Many algorithms running at the same time
	Sequential	The algorithms are ordered to run in sequential order
	Isolated	Just one algorithm running in each run

1.5 Type of Cooperation

The cooperation gives a new dimension to the problem, is not only the selection of the algorithm problem that is the focus but also the problem of achieve an overall better performance (better solution or faster answer). To have some sort of cooperation is also necessary to have many algorithms evolved in sequential or in parallel architecture.
Typically, we can separate cooperation in two main areas: cooperative and competitive, of course it is possible to have both in different moments, what can become difficult to classify with confident. The most used type of cooperation in this field is the competitive way. All the algorithms compete for the computer resources, the more fit receive more computational resources. One classical type is the portfolio architecture, in this architecture we have many algorithms running independently, the first to end, ends the meta-algorithm. When they compete for resources, they do not share efforts.
When exist cooperation, exist sharing of efforts, this can be sharing of best solution (island model), sharing of space areas to search (divide-conquer), etc. One example of cooperation is in Carchrae et al. [9] work, uses a structure similar with Beck et al. [8], but with some machine learning techniques not heuristic rules. When the switch algorithm occurs, the last algorithms share information to the new algorithm.
The table 8 resumes the taxonomy of the type of cooperation used in the field.

Table 8. Types of cooperation

Cooperation	Cooperative	Different algorithm that cooperate, and share efforts to obtain the final result
	Competitive	Different algorithm that run as isolated algorithms, don't share efforts

5. Classification of the field papers

In table 9, we try to classify some papers typically referenced in the field. Many others could be in this study, but we had to make some choices. Give the diversity of interesting works we tried to show some of them. A few classifications were very difficult, some because of difficulty to understand the author idea, but mainly because the authors could make more rich combination than our suggestions.

Table 9. Paper references by taxonomy field

Type	Name	Papers	Nº
Knowledge	High-Knowledge	[19],[20],[6],[11],[12], [16],[17]	7
	Low-knowledge	[21],[22],[2],[3],[4],[23], [5],[7],[8],[9],[10], [14],[15],[18],[13]	15
Learning	Offline	[19],[20],[22],[2],[3],[4], [23],[6],[14],[16],[17]	11
	Online	[21],[5],[7],[8],[9], [10],[11],[12],[15],[18], [13]	11
	Life		0
Reactivity	Reactive	[19],[2],[4],[23],[5], [8],[9],[10],[11],[12], [14],[15],[18],[13],[7]	15
	No reactive	[20],[21],[22],[3],[6],[16], [17]	7
Cooperation	Cooperative	[19],[21],[2],[4],[23],[9], [10]	7
	Competitive	[20],[22],[3], [5], [6], [7],[8],[11],[12],[14], [15],[16],[17],[18],[13]	15
Architecture	Parallel	[2],[4],[5],[6],[7], [8],[9],[10],[11],[12], [14],[15],[16],[17],[18], [13]	16
	Sequential	[19],[23],[9],[10]	4
	Isolated	[20],[21],[22],[3],[8]	5

One conclusion of this table is that the use low knowledge approach is very often used but is not classify by the authors as a low knowledge approach, maybe because this term is new in the field or because is difficult to classify. Neither the offline nor the online strategy is dominating the field. Seems that exist a tendency that the reactive approach is becoming the standard.

The competitive strategy continues to be the more used. This type of strategy is easier to understand the implication of the work done, but the cooperation between algorithms is a more realistic choice in real problems in the opinion of the authors. Once more is more easy to use machine learning techniques in this strategy than using in cooperation. In recent years as computer (or networks) are become less and less expensive the parallel architecture (implicit or explicit) are become the standard.

The use of the life learning as we purpose is an area to explore, as we did not see a work that fits our definition of life learning.

In Table 10, we have done some cross-reference study. We can make some conclusions – if we consider our sample a good sample of the work done in the field. When we use low-knowledge approach, it is 88% (if we consider [21] as an offline system dedicated to a particular instance– it's a possible view to the same system,

then the number will be 100%) more provable that we are using a reactive system. This conclusion it is expected because the use of low-knowledge needs less computer resources to achieve a relation between algorithms and problem instances, the need of less computational resources can be used to give more flexibility to the system – reacting to new information's. We can see that 82% of the online learning systems are of the low-knowledge type. The offline learning uses 71% of the time high knowledge as expected. The most common systems (36%) in our sample are the online systems, which use low-knowledge approach and are capable of reacting

Table 10. Cross-Reference by Knowledge/Learning/Reactivity

	High-Knowledge		Low-knowledge	
	Reactive	No reactive	Reactive	No reactive
Online	[11], [12]	-	[5],[7],[8],[9],[10], [15],[18], [13]	[21]
Offline	[19]	[20], [6], [16],[17]	[2],[4], [23],[14]	[22],[3]

6. Conclusions

We can see that the systems are becoming more complex as the computer power increases. Nowadays the trend is to use parallel systems in the form of portfolio architecture or other related forms. This is a natural path because of the cheap number of computer networks available - and the grid is a reality waiting to use in this type of problems. One recent trend that is also catching the eyes of the researchers is using low-knowledge approach in their meta-algorithms. The problems are becoming more complex (see the SAT problem) the need of better expertise by the researcher are pushing the low-knowledge approach a strong path to the future research. We can see that the isolated architecture is being abandoned, not because is not a good idea, but just because the research community as realize that is important to have many algorithm running in complex problems [24]. Another recent trend is the use of reactivity; the communities become conscious that a particular algorithm that has median lower performance – discard in a no reactivity strategy – can be the best in a particular stage of the meta-algorithm. The Life learning strategy as we defined need some research as can see with our sample of works, but seems to be a cleaver way to use different approaches. In offline approach, we can use huge statistical knowledge and in online approach we have to use less statistical knowledge – but more dedicated to the present instance. A good future research is to try to build a reactive system that can incorporate these two types of approaches – life-learning approach.
In resume, for the author, the future research is in building a meta-algorithm that uses many heterogeneous algorithms (example local search and global search) in a parallel architecture. This meta-algorithm has to promote cooperation between algorithms in the form of sharing efforts (sharing solutions, sharing regions of the search space), can react and change the algorithms if sees an opportunity – certain places in the search space. The future is in using life- learning approach, maybe using the high

knowledge for the first stages of the algorithm and low knowledge for the latter stages.

7. References

1. J. R. Rice, “The algorithm selection problem,” In M. V. Zelkowitz, editor, Advances in computers, vol. 15, pp. 65--118.(1976)
2. M. G. Lagoudakis and M. L. Littman, “Algorithm selection using reinforcement learning,” ICML, Morgan Kaufmann, San Francisco, CA, pp. 511--518. (2000)
3. Pavel B. Brazdil and Carlos Soares, “A comparison of ranking methods for classification algorithm selection,” ECML, (2000)
4. M. G. Lagoudakis and M. L. Littman,”Selecting the right algorithm,” Proceedings of the 2001 AAAI Fall Symposium Series: Using Uncertainty within Computation, Boston, MA(2001)
5. Mauro Birattari, Thomas Stützle, and Luis Paquete, Klaus Varrentrapp, ”A Racing Algorithm for Configuring Metaheuristics,” GECCO 2002, pp. 11--18.(2002)
6. Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham, “A Portfolio Approach to Algorithm Selection,” IJCAI 2003,pp. 1542--1542. (2003)
7. Yuan B. and Gallagher M., “Statistical racing techniques for improved empirical evaluation of evolutionary algorithms,” Proceedings of the 8th International Conference on Parallel Problem Solving from Nature, Springer-Verlag, pp. 172--181. (2004)
8. Beck J.C. and Freuder E.C., “Simple Rules for Low-Knowledge Algorithm Selection,” Proceedings of the First International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CPAIOR'04) (2004)
9. Carchrae T. and Beck, J.C., “Low knowledge algorithm control,” Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI04) (2004)
10. Tom Carchrae and J. Christopher Beck, “Applying machine learning to low-knowledge control of optimization algorithms,” Computational Intelligence 21 (4), pp. 372--387 (2005)
11. Gagliolo, M., Schmidhuber, J., “A neural network model for inter-problem adaptive online time allocation,” W. Duch et al., eds., ICANN 2005 (2005)
12. Gagliolo M. and Schmidhuber, J., “Learning dynamic algorithm portfolios,” AI & MATH 2006 Special Issue of the Annals of Mathematics and Artificial Intelligence, 47(3-4), pp. 295--328 (2006)
13. Yuan, B. and Gallagher, M., “Combining Meta-EAs and racing for difficult EA parameter tuning tasks,” Lobo et al. eds. Parameter Setting in Evolutionary Algorithms, Series of Studies in Computational Intelligence (SCI), vol. 54, Springer (2007)
14. Marek Petrik and Shlomo Zilberstein, ”Learning static parallel portfolios of algorithms,” Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida (2006)
15. Gagliolo M. and Schmidhuber J., “Gambling in a computationally expensive casino: algorithm selection as a bandit problem. online trading of exploration and exploitation,” NIPS 2006 Workshop, Whistler, British Columbia, Canada (2006)
16. M. Roberts and A.E. Howe, “Directing a portfolio with learning”, AAAI 2006 Workshop on Learning for Search (2006)
17. M. Roberts, “Exploiting portfolio strategy to explore the interaction of problems and algorithms in AI planning,” International Conference on Automated Planning and Scheduling (ICAPS-06) Doctoral Consortium (2006)
18. Gagliolo M. and Schmidhuber J., “Learning restart strategies,” M. Veloso, ed., IJCAI 2007, Hyderabad, India, vol. 1, pp. 792--797, AAAI Press (2007)

19. C. Brodley. “Addressing the selective superiority problem: Automatic algorithm/model class selection," Proceedings of the Tenth International Conference on Machine Learning, pp. 17--24. (1993)
20. John A. Allen and Steven Minton., “Selecting the right heuristic algorithm: Run time performance predictors,” GordonMcCalla,editor, Intelligence: The Eleventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence. Springer-Verlag, Berlin, Germany, pp. 41--53.(1996)
21. Lionel Lobjois and Michel Lemaitre, “Branch and bound algorithm selection by performance prediction,” Proceedings of the Fifteenth National Conference on Artificial Intelligence, Menlo Park: AAAI Press, pp 353--358.(1998)
22. E. Fink. “How to solve it automatically: Selection among problem solving methods,” R. G. Simmons, M. M. Veloso, and S. Smith, editors, Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, pp. 128--136 (1998)
23. M. G. Lagoudakis and M. L. Littman, “Learning to select branching rules in the DPLL procedure for satisfiability,” Electronic Notes in Discrete Mathematics (ENDM) (2001)
24. Carla P. Gomes and Bart Selman, “Algorithm portfolio design: theory vs. practice,” UAI 1997, pp. 190--197 (1997)
25. E. Horvitz, Y. Ruan, C. Gomes, H. Kautz, B. Selman, and D. M. Chickering,”A Bayesian approach to tackling hard computational problems,” Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (2001)
26. Leyton-Brown K., Nudelman, and E., Shoham, Y., “Learning the empirical hardness of optimization problems: The case of combinatorial auctions,” In: ICCP: International Conference on Constraint Programming (CP), LNCS (2002)
27. Stuart J. Russell and Peter Norvig. Artificial Intelligence, A Modern Approach, Prentice-Hall, Englewood Cli_s, 2nd edition (2003)
28. Frank Hutter, Youssef Hamadi, Holger Hoos, and Kevin Leyton-Brown, “Performance prediction and automated tuning of randomized and parametric algorithms,” Twelfth International Conference on Principles and Practice of Constraint Programming, 2006
29. Tom M. Mitchell, Machine learning, McGraw-Hill (1997)