Piece of Pie Search: Confidently Exploiting Heuristics

Nikolaos Pothitos
pothitos@di.uoa.gr

Panagiotis Stamatopoulos
takis@di.uoa.gr

Department of Informatics and Telecommunications
National and Kapodistrian University of Athens
Panepistimiopolis, 157 84 Athens, Greece

ABSTRACT

Search is not a direct path to a solution. While searching for a solution to a problem, *heuristics* consult us to avoid paths with dead ends, but they are not infallible. Many popular search methodologies "disobey" them during critical points of the search. In this work, we found an efficient stochastic methods framework that smoothly combines randomness with normal heuristics. We consider a factor of disobedience to the heuristics and we fine-tune it each time, according to our estimation of heuristic-reliability. We prove mathematically that while the disobedience factor falls, the stochastic methods approximate deterministic methods. Our algebraic evidence is supported by empirical evaluations on real life problems, such as course scheduling and frequency assignment. In this context, we exploit our proposed heuristic-reliability semantics in order to produce a *piece of pie search* (PoPS) method that can outperform other known constructive search processes in hard optimization problems.

CCS Concepts

•**Computing methodologies** → **Randomized search;** Heuristic function construction; •**Mathematics of computing** → *Optimization with randomized search heuristics;* •**Theory of computation** → *Theory of randomized search heuristics;*

Keywords

randomness; stochastic methods; discrepancy; constructive search; confidence; CSP

1. INTRODUCTION

Artificial Intelligence (AI) methodologies aim to tackle with difficult computational and real life problems, such as scheduling [16], radio frequency assignment [5], other NP-hard problems, and also problems stemming from various disciplines, e.g. Bioinformatics [1].

SETN '16, May 18-20, 2016, Thessaloniki, Greece

DOI: http://dx.doi.org/10.1145/2903220.2903242

In these cases, naive algorithms have to explore the whole candidate solutions spectrum, in order to find a real solution. The issue here is that the candidate solution range is exponential in the problem instance parameters, and, as a consequence, an iteration through the candidate solutions becomes infeasible as the problem scales.

Heuristics role in this situation is to change the order of the candidate solutions, so as to favor the "promising" ones. In other words, heuristics make an estimation of the possibility of an incomplete or candidate solution being a real solution, and label it with a priority. A high priority means that the candidate solution should be examined soon.

This reordering cannot make the search space tractable—this is most probably impossible [8]—but it is able to dramatically decrease the time needed to guide a search method toward a real solution. In this direction, we study heuristic properties, such as reliability/confidence, and we propose a generic framework in order to exploit them by incorporating a randomness factor into them.

2. BACKGROUND AND RELATED WORK

We focus on *constraint satisfaction problems* (CSPs) [21] that can be faced via a plethora of available *constraint programming* (CP) solvers [7, 13].

2.1 Constraint Satisfaction Problems

Every single CSP can be stated using commonplace formalizations. It is a triplet of (i) *variables* $X_1, \ldots, X_n$, (ii) their corresponding *domains* $D_{X_1}, \ldots, D_{X_n}$, which are ordinarily finite sets of integers, and (iii) the *constraints* between variables; a constraint contains the tuples of all the valid assignments for a specific pair/set of variables. To put it differently, a constraint is a relation between the variables, such as $X_1 < X_2$.

In the attempt to find a solution to a CSP, we have to make assignments.

Definition 1. We say that a variable X is *assigned* a value $v \in D_X$, if its domain is made singleton, i.e. $D_X \rightsquigarrow \{v\}$.

A *solution* is an assignment that involves all variables and also satisfies all the constraints. The *search* process leads a CSP after consecutive assignments into a solution. The strategic advantage of the overall paradigm is that the CSP description and search phases are independent [9].

2.2 Map-Coloring Problem

There exists a huge list of interesting CSPs [10]. For example, *map-coloring* is a CSP for assigning colors to each

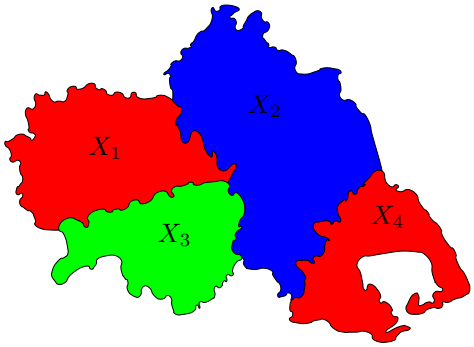

Figure 1: The four Thessaly prefectures

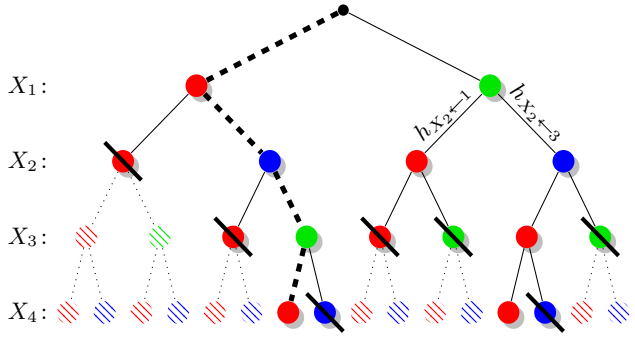

Figure 2: The search tree for Thessaly-coloring

prefecture in a given map, so as no neighbouring prefectures have the same color. Figure 1 illustrates a map of the Greek region "Thessaly", containing four prefectures; the colors in the figure form an indicative solution.

Problem 1. Typically, *"Thessaly-coloring"* is a CSP with:

1. Four constrained variables: X_1, X_2, X_3, X_4. Each one of them designates the prefecture color.
2. The corresponding domains are $D_{X_1} = D_{X_3} = \{1, 2\}$ and $D_{X_2} = D_{X_4} = \{1, 3\}$. Numbers 1 2 3 represent respectively red, green, blue.[1]
3. The constraints are $X_1 \neq X_2$, $X_1 \neq X_3$, $X_2 \neq X_3$, and $X_2 \neq X_4$.

The solution in Fig. 1 is represented by the assignment

$$\{X_1 \leftarrow 1,\ X_2 \leftarrow 3,\ X_3 \leftarrow 2,\ X_4 \leftarrow 1\}\,. \qquad (1)$$

2.3 Search Tree Exploration

A *search tree* is a descriptive way to depict every possible assignment in a CSP, such as map-coloring. Figure 2 displays the search tree for the Thessaly-coloring problem. The struck out nodes have been pruned as no-goods.

Each path from the root (i.e. the uppermost node) represents an *assignment.* If the path from the root ends up into a leaf (lowest node), we have a *complete* assignment. E.g., the dotted path in Fig. 2 is an alternative form of the solution assignment in (1).

2.4 Heuristic Estimation as a Real Number

A heuristic function maps every possible choice in the search tree to a number that corresponds to the estimation that it will eventually guide us toward a solution.

Definition 2. For a specific search tree node, let Choices be the set with the alternative assignments that one may follow. The *heuristic function* h_i maps each alternative assignment $i \in$ Choices to a positive number or zero, i.e. h : Choices $\rightarrow \mathbb{R}^+$.

Example 1. In Fig. 2 uppermost right node, there are two alternative assignments in Choices $= \{X_2 \leftarrow 1,\ X_2 \leftarrow 3\}$. One heuristic function may provide the estimations, e.g. $h_{X_2 \leftarrow 1} = 0.7$ and $h_{X_2 \leftarrow 3} = 2.8$; that is, the assignment $X_2 \leftarrow 3$ is more promising.

[1]We could initially set all the domains equal to $\{1, 2, 3\}$. We used smaller initial domains just to simplify the problem.

The above example is almost ideal, as the heuristic function h favours the assignment $X_2 \leftarrow 3$ over $X_2 \leftarrow 1$. Besides, the latter leads to a dead end, as its two descendants are struck out in Fig. 2, because they violate the constraints.

Unfortunately, this is not always the case, i.e. the heuristic value for an assignment that leads to a dead end (say $X_2 \leftarrow 1$ in Fig. 2) may be overestimated or, even worse, may be greater than the heuristic estimation for an assignment that really leads to a solution (e.g. $X_2 \leftarrow 3$).

A heuristic value h_i is actually a *prediction* whether a specific assignment will ultimately guide us to a solution or not. Being a prediction, it implies an inherent *reliability/ confidence* level.

2.5 Heuristics Exploitation in Related Work

In *constructive search,* one can build a solution either with a deterministic/systematic search method, or by making one-by-one random assignments. Do these methods exploit heuristics and how?

2.5.1 Deterministic Search Methods

To our knowledge, existing search methods such as *limited discrepancy search* (LDS) use heuristics only to *order* the possible assignments and do not exploit the *difference* of the one heuristic estimation to another, but only their *rank* [19]. For example, the *iterative broadening* method explores only a limited children's number for each search tree node [11]. Of course, it chooses to visit only the children with the highest ranks. *Credit search* [2] and *limited assignment number* (LAN) [3] are other deterministic methods that also take into account the rank of the heuristic estimations and not the heuristic values themselves.

Last but not least, there are also methods that make the assumption that *the heuristic function is more reliable as the search tree node depth increases.* E.g., *depth-bounded discrepancy search* (DDS) allows to override a heuristic estimation, only when we have not yet reached a specific search tree depth [22]. Finally, there are some methodologies that take into account two or more heuristic functions and *learn* as the search proceeds, which heuristic is the best to use [23].

2.5.2 Random Search Methods

On the other hand, stochastic search methods completely ignore heuristics, as they choose to make an assignment at random [14]. For example, *depth first search with restarts* traverses the search tree making random choices, and when a specific time limit is reached, it re-starts from the beginning.

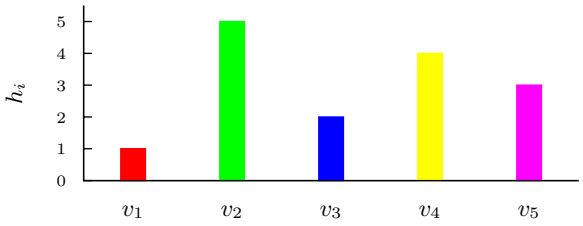

Figure 3: Heuristic estimations h_i for each value v_i

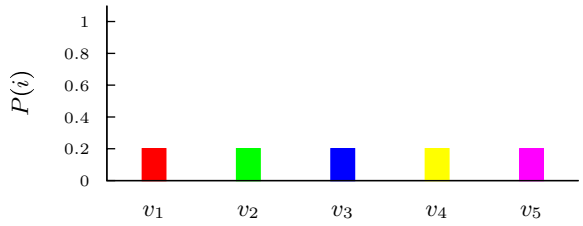

Figure 4: The probability is spread uniformly

2.5.3 *Heuristics and Probabilities*

Thus, the well-known search methods either use heuristics as Choices ranks, or completely ignore them. In 1996, Bresina transformed the heuristic ranks into *probabilities* via the so-called *heuristic-biased stochastic sampling* (HBSS) [4]. He provided a set of various decreasing functions bias(r), e.g. $\frac{1}{r}$ or e^{-r} etc., that take a specific integer choice rank $r \in \{1, 2, \ldots\}$ and return a number that corresponds to the probability of the choice to be selected. Cicirello and Smith improved HBSS by introducing the *value-biased stochastic sampling* (VBSS). The bias function now takes as argument the heuristic *value* itself [6].

On the other hand, Gomes et al. exploit the so-called *heuristic-equivalence* to equate the choices with the highest heuristic values. In this way, we can exclude the choices with the lower heuristic values and select at random amongst the choices with the most prevailing values [12].

3. NEW PROBABILISTIC HEURISTICS

Our contribution lies in the mathematical foundation of a framework that covers the above heuristic categories. In contrast to existing methodologies, we leverage on the *smooth* transition from the total randomness to determinism.

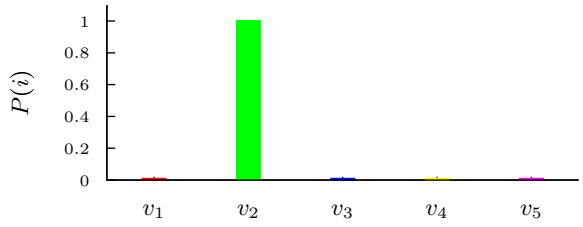

Figure 5: Systematic search favours the highest h_i

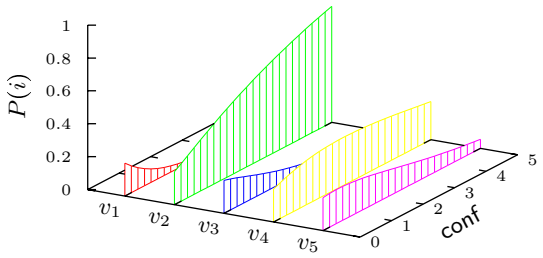

Figure 6: As conf rises, the effect to $P(i)$ is greater

3.1 Heuristics Probabilistic Foundations

Probabilities are a more precise way to depict heuristics than orderings, because heuristics are actually *estimations* whether a choice will guide us to a solution; they are not a strict quality rank.

Definition 3. A function P : Choices $\rightarrow [0, 1]$, namely a *heuristic distribution function,* maps each available choice to a corresponding probability, i.e. $P(i)$.

Property 1. It should hold that $\sum_i P(i) = 1$, as P denotes a probability for each $i \in$ Choices.

Regarding random search methods (Section 2.5.2), the probability is distributed uniformly along the Choices. Conclusively,

PROPOSITION 1. *The heuristic distribution for a random method is always* $P(i) = \frac{1}{|\text{Choices}|}$, $\forall\, i$.

Example 2. Say that Choices $= \{v_1, v_2, \ldots, v_5\}$. Every v_i denotes a possible assignment. Furthermore, in a specific search tree node we can make five different assignments, and their corresponding heuristic estimations h_i are 1, 5, 2, 4, 3 respectively, as in Fig. 3.

Figure 4 depicts the corresponding heuristic distribution function for a random method, that is $P(i) = 1/5$, $\forall\, i$.

On the other extreme, deterministic search methods (Section 2.5.1) always select the choice v_i that corresponds to the h_i with the highest rank.

PROPOSITION 2. *Formally, in deterministic search methods, if* $i = \arg\max_j h_j$, *then* $P(i) = 1$, *otherwise* $P(i) = 0$.

Example 3. The greatest heuristic value in Example 2 is $h_2 = 5$. Hence, a deterministic search method would select v_2 with a certain probability $P(v_2) = 1$. Consequently, the rest of the probabilities are zero, as in Fig. 5.

If there is more than one maximum heuristic value, deterministic methods arbitrarily concern only one of them as maximum. To simplify the following equations we will make the assumption that there is only one maximum. Without loss of generality, we also assume that heuristic values are non-zero.

3.2 Bridging the Two Opposites

We extend our previous formulation of the heuristic distribution function (Definition 3) in order to compromise random and deterministic methods. We introduce a parameter $\mathsf{conf} \in \mathbb{R}^+$, that signifies how much the heuristic estimations will be taken into account; it is the heuristic *confidence*. This confidence pararameter is the basis to define the condition when a heuristic distribution function is "balanced".

Definition 4. A parameterized heuristic distribution function $P_{\mathsf{conf}}(i)$ is *balanced* if and only if:

1. $\forall\, i,\ \lim_{\mathsf{conf}\to 0} P_{\mathsf{conf}}(i) = \frac{1}{|\mathsf{Choices}|}$, and

2a. if $i = \arg\max_j h_j$, $\lim_{\mathsf{conf}\to\infty} P_{\mathsf{conf}}(i) = 1$,

2b. otherwise, $\lim_{\mathsf{conf}\to\infty} P_{\mathsf{conf}}(i) = 0$.

Moreover, the function $P_{\mathsf{conf}}(i)$ must be monotonic and continuous with respect to conf and for fixed i.

Intuitively, conf is the link between random and deterministic search methods, as the above definition covers both Proposition 1 when $\mathsf{conf} \to 0$ and Proposition 2 when $\mathsf{conf} \to \infty$. In other words, conf is the position along the random-deterministic axis.

What happens for intermediate conf values? This depends on the precise parameterized heuristic distribution function instance. We define the following function that gently scales randomness.

LEMMA 1. *The function* $P_{\mathsf{conf}}(i) = \frac{h_i^{\mathsf{conf}}}{\sum_j h_j^{\mathsf{conf}}}$ *is balanced.*[2]

PROOF. We prove Definition 4 three requirements.

1. $\lim_{\mathsf{conf}\to 0} P_{\mathsf{conf}}(i) = \frac{h_i^0}{\sum_{j\in\mathsf{Choices}} h_j^0} = \frac{1}{\sum_{j\in\mathsf{Choices}} 1} = \frac{1}{|\mathsf{Choices}|}$.

2a. Let $n = |\mathsf{Choices}|$. This number is bounded as the possible assignments in a CSP are a finite set. Thus, the distribution function can be analyzed as

$$P_{\mathsf{conf}}(i) = \frac{h_i^{\mathsf{conf}}}{\sum_j h_j^{\mathsf{conf}}} = \frac{h_i^{\mathsf{conf}}}{h_1^{\mathsf{conf}} + h_2^{\mathsf{conf}} + \cdots + h_{\max}^{\mathsf{conf}} + \cdots + h_n^{\mathsf{conf}}}\,.$$

Let $h_{\max}$ be the maximum h_i. If we divide by $h_{\max}^{\mathsf{conf}}$ both the nominator and denominator, we have

$$P_{\mathsf{conf}}(i) = \frac{\left(\frac{h_i}{h_{\max}}\right)^{\mathsf{conf}}}{\left(\frac{h_1}{h_{\max}}\right)^{\mathsf{conf}} + \cdots + 1 + \cdots + \left(\frac{h_n}{h_{\max}}\right)^{\mathsf{conf}}} = \frac{\left(\frac{h_i}{h_{\max}}\right)^{\mathsf{conf}}}{1 + \sum_{j\neq\max}\left(\frac{h_j}{h_{\max}}\right)^{\mathsf{conf}}}\,. \tag{2}$$

Here, max is an abbreviation for $\arg\max_i h_i$. Therefore, $\forall\, j \neq \max$,

$$h_j < h_{\max} \implies \frac{h_j}{h_{\max}} < 1 \implies \lim_{\mathsf{conf}\to\infty}\left(\frac{h_j}{h_{\max}}\right)^{\mathsf{conf}} = 0\,. \tag{3}$$

As a result from (2) and (3),

$$\lim_{\mathsf{conf}\to\infty} P_{\mathsf{conf}}(i) = \frac{\lim_{\mathsf{conf}\to\infty}\left(\frac{h_i}{h_{\max}}\right)^{\mathsf{conf}}}{1 + \sum_{j\neq\max}\lim_{\mathsf{conf}\to\infty}\left(\frac{h_j}{h_{\max}}\right)^{\mathsf{conf}}} = \lim_{\mathsf{conf}\to\infty}\left(\frac{h_i}{h_{\max}}\right)^{\mathsf{conf}}. \tag{4}$$

A direct derivation is that for $i = \max \equiv \arg\max_j h_j$, we have $\lim_{\mathsf{conf}\to\infty} P_{\mathsf{conf}}(\max) = 1$, which is the second prerequisite for a balanced function.

2b. Finally, the last prerequisite of Definition 4 involves $i \neq \max \Rightarrow h_i < h_{\max} \Rightarrow \frac{h_i}{h_{\max}} < 1$, which, combined with (4), gives $\lim_{\mathsf{conf}\to\infty} P_{\mathsf{conf}}(i) = 0$, which had to be demonstrated. □

The above function (in Lemma 1) is balanced and it also moves smoothly from the random extreme to the deterministic one, because it is a *continuous* function, with regard to $\mathsf{conf} \in \mathbb{R}^+$.

Hence, the overall function is a transition from the total randomness to the almost total determinism. This is illustrated in the three-dimensional Fig. 6, which for $\mathsf{conf} = 0$, is equivalent to the two-dimensional Fig. 4, and when $\mathsf{conf} \to \infty$, it is equivalent to Fig. 5.

4. PIECE OF PIE SEARCH

The probabilistic framework founded in the previous section, naturally complies with existing search methods; it affects only the heuristic function and not the methods themselves. But in order to fully exploit the introduced heuristics framework, we built the new constructive search method *Piece of Pie Search* (PoPS).

4.1 The Algorithm Core

Figure 7 describes PopsSample, which is the PoPS core. It is called inside PoPS in order to solve a CSP by providing a complete and valid Assignments set, which is initially empty.

In each PopsSample call we get an unassigned variable returned by the function VariablesOrderHeuristic($\mathscr{X}$), where $\mathscr{X}$ is the set of all the constrained variables. Then, it stores its domain D_X, in order to restore it in a future backtrack. All the above steps are common in constructive search methods.

The crucial and novel part of this function is inside the **while** iteration where we iterate through the different values in D_X. The call ValuesOrderHeuristic(D_X, conf) returns the best value out of D_X, according to the heuristic estimation, using the heuristic function in Lemma 1.

Normal search methods, like Depth First Search (DFS), Limited Discrepancy Search (LDS), and other known deterministic methods explore in their steps a specific *number* of values in D_X or every value in it (cf. Section 2.5.1). In PopsSample, we explore a specific *subset* D'_X of D_X, which corresponds to a proportion of the heuristics pie. The proportion is the argument $\mathsf{PieceToCover} \in [0, 1]$. When PieceToCover becomes 1, then PopsSample is a complete search method as it explores all the D_X set values.

Example 4. Figure 8 demonstrates the heuristics pie for the Example 2: Each h_i corresponds to a value v_i in D_X. In this case, a PopsSample$(0.5, 1)$ invocation would explore

[2]For $\mathsf{conf} = 1$, the function $P_1(i) = \frac{h_i}{\sum_j h_j}$ is equivalent to the *fitness proportionate selection* function—resembling a *roulette wheel*—that is used in Genetic Algorithms [20].

function PopsSample(PieceToCover, conf)
  if Assignments violate any constraint **then**
    return failure
  else if Assignments include every variable **then**
    Record Assignments as solution
    return success
  end if
  $X \leftarrow$ VariablesOrderHeuristic($\mathscr{X}$)
  $D_{X_{\text{init}}} \leftarrow D_X$
  CoveredPiece $\leftarrow 0$
  while CoveredPiece $\leqslant$ PieceToCover **do**
    value $\leftarrow$ ValuesOrderHeuristic(D_X, conf)
    CoveredPiece $\leftarrow$ CoveredPiece $+ \frac{h^{\text{conf}}_{X \leftarrow \text{value}}}{\sum_{v \in D_{X_{\text{init}}}} h^{\text{conf}}_{X \leftarrow v}}$
    Assign value to X and add it to Assignments
    PopsSample(PieceToCover, conf$+\frac{100-\text{conf}}{|\mathscr{X}|}$)
    Undo the assignment
    $D_X \leftarrow D_X -$ {value}
  end while
  $D_X \leftarrow D_{X_{\text{init}}}$ ▷ Restores initial domain
  return failure ▷ All alternative values are exhausted
end function

Figure 7: The recursive PopsSample called by PoPS

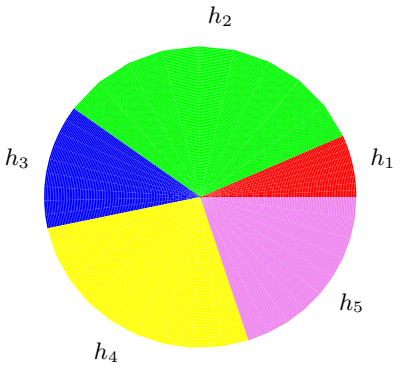

Figure 8: The heuristics pie chart for Example 2

half the pie. E.g., the choices that correspond to the heuristics $h_1^1 + h_2^1 + h_3^1$ or $h_2^1 + h_5^1$ make half the pie and more.

The exponent in the heuristic values has to do with the confidence semantics in our framework.

4.2 Heuristic Confidence vs. Node Level

An important detail in PopsSample appearing in Fig. 7, is the increase in conf as the current search tree node level deepens.

When we make the first recursive PopsSample call (inside **while**), we have already made an assignment. Hence, the current tree level will be augmented by 1 and conf will be increased by $\frac{100-\text{conf}}{|\mathscr{X}|}$.

Each subsequent recursive call deepens search by 1, until the current depth reaches $|\mathscr{X}|$, which means that every variable in $\mathscr{X}$ has been assigned a value. For a specific depth k the conf value is increased by $k \cdot \frac{100-\text{conf}}{|\mathscr{X}|}$. Finally, when $k = |\mathscr{X}|$, the conf argument of PopsSample will become

function PoPS
  for i **from** 1 **to** SamplesNum **do**
    Sample$_i$ is activated
    Cover$_i \leftarrow 0$
    conf$_i \leftarrow 100 \cdot \frac{i-1}{\text{SamplesNum}-1}$
  end for
  while the available time is not exhausted **do**
    for each active Sample$_i$ **do**
      if PopsSample(Cover$_i$, conf$_i$)
        did not return a solution **then**
        Sample$_i$ is deactivated
      end if
      Cover$_i \leftarrow$ Cover$_i + \frac{1}{d}$
    end for
    if every Sample$_i$ is deactivated **then**
      Activate every Sample$_i$ ▷ to keep searching.
    end if
  end while
end function

Figure 9: Piece of Pie Search (PoPS) Method

equal to the marginal value 100.

In the deepest node levels, heuristics are usually more accurate, because even more variables have been instantiated and we have a clearer picture of the problem. In our framework, more accuracy means more confidence, that's why we increase conf as the search method proceeds with the assignments.

4.3 PopsSample: Average Complexity

The PopsSample complexity depends on PieceToCover argument and the heuristic function distribution.

Lemma 2. *Let n be the constrained variables number and let d be the average domain size. Then, the average complexity of a* PopsSample(PieceToCover, conf) *call equals* $\mathcal{O}(d^n \cdot \text{PieceToCover}^n)$.

Proof. An initial PopsSample(PieceToCover, conf) call, iterates through the values of, let's say, the first variable X_1. If the heuristic function numbers for the values in D_{X_1} are uniformly distributed, the expected value for $h_{X_1 \leftarrow \text{value}}$ would be $\mu = \frac{\sum_{v \in D_{X_1}} h_{X \leftarrow v}}{|D_{X_1}|}$.

Thus, to reach the pie proportion $A = \text{PieceToCover} \cdot \sum_{v \in D_X} h_{X \leftarrow v}$, we need $A/\mu = \text{PieceToCover} \cdot |D_{X_1}|$ iterations, i.e. $\mathcal{O}(\text{PieceToCover} \cdot d)$ loops.

The total time needed is $T_1 = \mathcal{O}(\text{PieceToCover} \cdot d) \cdot T_2$, where T_2 is the time for the PopsSample call *inside* the loop. It also holds that $T_2 = \mathcal{O}(\text{PieceToCover} \cdot d) \cdot T_3$, etc., and finally $T_n = \mathcal{O}(\text{PieceToCover} \cdot d)$. In conclusion, the aggregate complexity is $\mathcal{O}(\text{PieceToCover}^n \cdot d^n)$ for the initial call. □

We can observe that PopsSample(1, conf) is equivalent to a complete search space exploration, which has an $\mathcal{O}(d^n)$ time complexity.

4.4 The Motivation Behind PoPS

Finding the best conf is the motivation behind PoPS. Unfortunately, we do not know a priori which conf is the best parameter for PopsSample. However, we can find it by trial and error. In Fig. 9, the PoPS function invokes

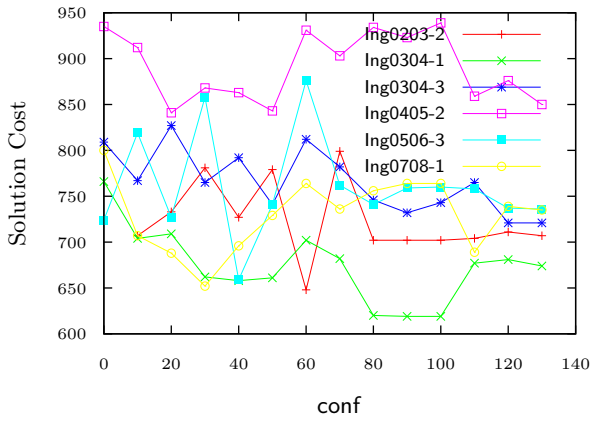

Figure 10: Timetabling solutions costs vs. conf

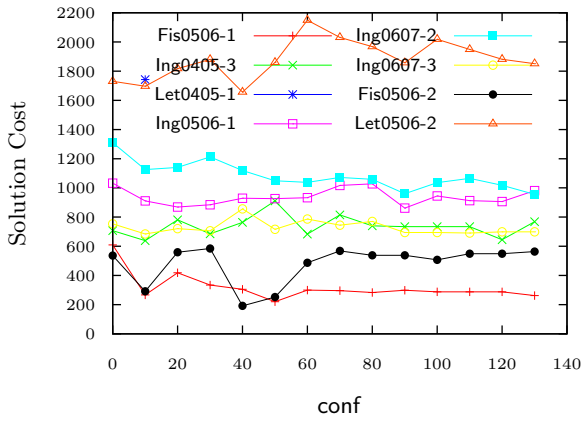

Figure 11: Solutions for the rest of the ITC instances

PopsSample for SamplesNum different conf_i values, including the marginal values 0 and 100.

Each different conf_i is used in turn. Initially, the Cover_i parameter in the PoPS algorithm is zero for every conf_i. When a specific conf_i has been examined, the corresponding Cover_i is increased by $\frac{1}{d}$, where d is the average domain size. When the second iteration over a specific conf_i ends, the Cover_i is increased again by $\frac{1}{d}$ and so on.

In this way, each conf_i is given the same opportunity (search space) to find a solution. If some conf_i does not produce a solution, it is deactivated. It is reactivated only if all other conf_i's fail to produce a solution.

5. EMPIRICAL EVALUATIONS

The gradual switch from randomness to determinism can boost search in demanding CSPs, such as course scheduling and the radio frequency assignment problems. With the help of our free constraint programming C++ library Naxos Solver [17], we solved official instances of these problems for different heuristic distribution configurations.

The source code for our evaluations is freely available at http://di.uoa.gr/~pothitos/PoPS including the problem datasets. The experiments were conducted on an HP computer with an Intel dual-core E6750 processor clocked at 2.66 GHz with 2 GB of memory and a Xubuntu Linux 12.04 operating system.

For the following first three subsections, our confidence framework was used to randomize only the *variables ordering heuristic* (minimum remaining values and degree used for tie breaking), whereas in the last subsection, which refers to PoPS, the randomization affects only the ValuesOrderHeuristic (least constraining value), called by PopsSample inside PoPS.

5.1 University Course Scheduling

Automated timetabling is nowadays a crucial application, as many educational institutes still use ad hoc manual processes to schedule their courses. The International Timetabling Competition (ITC) is an attempt to unify all these processes. We borrowed the fourteen instances of the latest contest track concerning *universities* [15].

In these problems, we have to assign valid teaching periods and rooms to the curriculum lectures. The objective is to distribute them evenly during the week but without having gaps between them, if scheduled on the same day; each gap increases the solution cost [18].

Due to ITC specifications, we had 333 seconds in our machine to solve each instance and minimize the solution cost as much as we could. Figures 10 and 11 display the minimum solution costs found per instance by PopsSample for various conf values. We observe that as conf increases the costs tend to a specific number, whilst for small conf values we have fluctuations because search becomes more random.

It was expected that for high conf values the results would be more stable, as the search process approximates the default depth-first-search (DFS). For the marginal low values, e.g. $\mathsf{conf} = 0$, search is completely stochastic and the results are worse on overage, as we have higher solution costs. However, the evaluations for intermediate conf values, e.g. $\mathsf{conf} \approx 20$, are more promising, but the automatic selection of the best conf is an open question here; in the last sections, PoPS finds automatically appropriate conf values.

In practice, as shown in Fig. 10 and 11, a conf value around 100 actually represents infinity, because search tends to produce the same solutions for $\mathsf{conf} \geqslant 100$.

It is worth to mention that in Fig. 11 the only solution found for the Let0405-1 instance, depicted with an asterisk $*$, was for $\mathsf{conf} = 10$.

5.2 Radio Link Frequency Assignment

Another important real problem is the frequency assignment, in which we have to assign a frequency to each radio transmitter with the objective to minimize the interference. The interference is minimized by assigning different frequencies to every two transmitters that are close to each other.

The Centre Electronique de l'Armement (CELAR) provides a set of real datasets for this NP-hard problem [5]. We chose to solve the five so-called "MAX" problem instances, namely SCEN06–SCEN10, in which, generally speaking, we try to maximize the number of the satisfied soft constraints.

For each of these instances, we had 15 minutes to explore the search space. We recorded the best (lowest) solution costs found so far in Fig. 12 for several conf values. Approximately the same as in course scheduling, the lowest solution costs occur around $\mathsf{conf} \approx 10$, which gives better results on average than the marginal conf values.

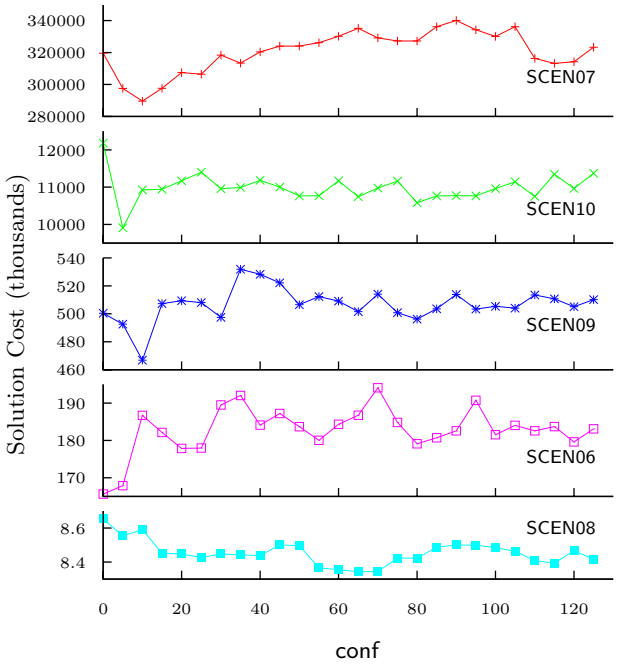

Figure 12: Unsatisfied soft constraints increase cost

5.3 PopsSample during Hard Optimization

The conf parameter can refine any search method that adopts our heuristic framework. The PopsSample method goes a step further: it incorporates our heuristic *confidence semantics* into its search engine.

In order to solve the first university course timetabling instance (Fis0506-1 of Section 5.1), we invoked PopsSample for various PieceToCover and conf values and we plotted the best solution costs found in Figure 13. The third dimension is the *cost* of the solutions found: the lower the solution cost is, the more qualitative timetable is produced.

In the same graphs we include some of the well-known search methods results, such as DFS, LDS, and Iterative Broadening, implementesd in the same solver, with only their best solution cost depicted as a plane grid, in order to make comparisons easily.

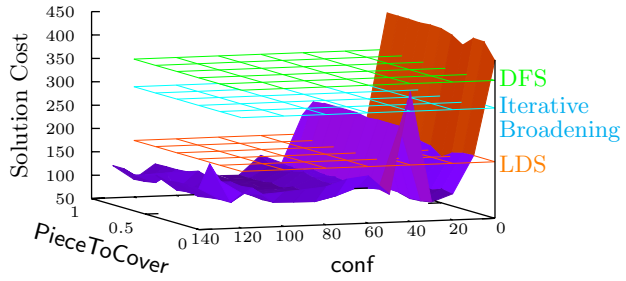

Figure 13: PopsSample for the first ITC instance

Table 1: Solution costs for fourteen ITC instances

Instance	PoPS	LDS	DFS	It. Broad.
Fis0506-1	*105*	171	345	286
Ing0203-2	*241*	288	698	321
Ing0304-1	*279*	307	578	353
Ing0405-3	*195*	215	817	235
Let0405-1	655	*627*	X	X
Ing0506-1	*307*	311	812	342
Ing0607-2	*282*	283	1184	328
Ing0607-3	*223*	239	635	262
Ing0304-3	*288*	294	675	370
Ing0405-2	*265*	284	877	344
Fis0506-2	*12*	33	486	34
Let0506-2	*713*	783	1621	937
Ing0506-3	*231*	256	660	280
Ing0708-1	*223*	227	660	264

5.4 PoPS vs. Other Search Methods

In the above sections, it was not easy to figure out which is the best PieceToCover and conf combination. That is why we employed PoPS to solve the fourteen course timetabling instances. As described in Section 4.4, PoPS uses several conf values and favours the most fruitful ones. We used five conf samples, i.e. 0, 25, 50, 75, and 100, by setting SamplesNum equal to 5. In this way, PoPS constructed solutions with lower costs than the other methods, except for the fifth instance, as illustrated in Table 1. The time limit for all the methods was set to 15 minutes.

6. CONCLUSIONS AND PERSPECTIVES

We presented a well-founded framework to exploit both stochastic and deterministic heuristics. Empirical evaluations showed that our hybrid approach can produce better results than fully random or fully deterministic methodologies.

In order to achieve this, we approached and used heuristics as a *confidence* and *reliability* measure. By exploiting these heuristic semantics, we were able to produce a new efficient search method, namely PoPS, that can outperform other methodologies. In general, our proposed framework gives the opportunity to exploit "on the fly" whichever heuristic confidence fluctuations occur.

In future, it will be challenging to parallelize it, as it supports a whole grid of strategies, by concurrently invoking PopsSample with several PieceToCover and conf arguments.

7. ACKNOWLEDGMENTS

This research was partially funded by the University of Athens Special Account of Research Grants no 10812.

We also thank Foivos Theocharis who initially built the search methods library Amorgos, which is available together with Naxos [17].

8. REFERENCES

[1] P. Barahona, L. Krippahl, and O. Perriquet. Bioinformatics: A challenge to constraint programming. In P. Van Hentenryck and M. Milano, editors, *Hybrid Optimization*, volume 45, pages 463–487. Springer, New York, 2011.

[2] R. Barták. Incomplete depth-first search techniques: A short survey. In *CPDC 2004: 6th Workshop on Constraint Programming for Decision and Control*, pages 7–14, 2004.

[3] R. Barták and H. Rudová. Limited assignments: A new cutoff strategy for incomplete depth-first search. In H. Haddad, L. M. Liebrock, A. Omicini, and R. L. Wainwright, editors, *SAC 2005: Symposium on Applied Computing, Santa Fe, New Mexico*, pages 388–392, New York, 2005. ACM.

[4] J. L. Bresina. Heuristic-biased stochastic sampling. In W. J. Clancey and D. S. Weld, editors, *AAAI 1996: 13th National Conference on Artificial Intelligence, Portland, Oregon*, volume 1, pages 271–278, Menlo Park, 1996. AAAI Press.

[5] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners. Radio link frequency assignment. *Constraints*, 4(1):79–89, 1999.

[6] V. A. Cicirello and S. F. Smith. Enhancing stochastic search performance by value-biased randomization of heuristics. *Journal of Heuristics*, 11(1):5–34, 2005.

[7] ECL^{i}PS^e constraint logic progr. http://eclipseclp.org, 2012.

[8] L. Fortnow. The status of the P versus NP problem. *Communications of the ACM*, 52(9):78–86, 2009.

[9] E. C. Freuder and B. O'Sullivan. Grand challenges for constraint programming. *Constraints*, 19(2):150–162, 2014.

[10] I. P. Gent and T. Walsh. CSPLib: A benchmark library for constraints. In J. Jaffar, editor, *CP 1999: 5th International Conference on Principles and Practice of Constraint Programming, Alexandria, Virginia*, volume 1713 of *LNCS*, pages 480–481, Heidelberg, 1999. Springer. http://CSPLib.org.

[11] M. L. Ginsberg and W. D. Harvey. Iterative broadening. *Artificial Intelligence*, 55(2-3):367–383, 1992.

[12] C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1-2):67–100, 2000.

[13] Ilog Solver. http://ilog.com/products/cp, 2012.

[14] B. Jafari and M. Mouhoub. Heuristic techniques for variable and value ordering in CSPs. In *GECCO 2011: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, Dublin*, pages 457–464, New York, 2011. ACM.

[15] B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. Di Gaspero, R. Qu, and E. K. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 22(1):120–130, 2010.

[16] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, New York, 4th edition, 2012.

[17] N. Pothitos. Naxos. http://di.uoa.gr/~pothitos/naxos, 2013.

[18] N. Pothitos, P. Stamatopoulos, and K. Zervoudakis. Course scheduling in an adjustable constraint propagation schema. In *ICTAI 2012: 24th IEEE International Conference on Tools with Artificial Intelligence*, volume 1, pages 335–343. IEEE, 2012.

[19] P. Prosser and C. Unsworth. Limited discrepancy search revisited. *J. Experim. Algor.*, 16:1.6:1–1.6:18, 2011.

[20] D. Sharma, V. Singh, and C. Sharma. GA-based scheduling of FMS using roulette wheel selection process. In *SocProS 2011: International Conference on Soft Computing for Problem Solving*, volume 131, pages 931–940. Springer, 2012.

[21] E. Tsang. *Foundations of Constraint Satisfaction*. Books on Demand, Norderstedt, 2014.

[22] T. Walsh. Depth-bounded discrepancy search. In M. E. Pollack, editor, *IJCAI 1997: 15th International Joint Conference on Artificial Intelligence, Nagoya, Japan*, volume 2, pages 1388–1393, San Francisco, 1997. Morgan Kaufmann.

[23] Y. Xu, D. Stern, and H. Samulowitz. Learning adaptation to solve constraint problems. In *LION 3: 3rd International Conference on Learning and Intelligent Optimization*, 2009.