A model of costs and benefits of meta-level computation

van Harmelen, F.A.H.

published in
Proceedings of the Fourth Workshop on Meta-programming in Logic ({META'94})
1994

document version
Publisher's PDF, also known as Version of record

Proceedings of the Fourth Workshop on Meta-programming in Logic (META'94)
LNCS 883, pp. 248-261, Springer Verlag, June 1994

A model of costs and benefits of meta-level computation

Frank van Harmelen

SWI
University of Amsterdam
frankh@swi.psy.uva.nl

Abstract. It is well known that meta-computation can be used to guide other computations (at the object-level), and thereby reduce the costs of these computations. However, the question arises to what extent the cost of meta-computation offsetts the gains made by object-level savings. In this paper we discuss a set of equations that model this trade-off between object-savings and meta-costs. The model shows that there are a number of important limitations on the usefulness of meta-computation, and we investigate the parameters that determine these limitations.

1 Introduction

One of the most often stated aims of meta-programming is *search-control*: a meta-program is used to guide the computation at the object-level. Often, this takes the form of a meta-program choosing among multiple applicable object-level computations. A large body of literature exists on this type of meta-programs, in areas like knowledge-representation, (logic-)programming and theorem proving.

Although many other types of meta-programs are both possible and useful, this paper will only consider meta-programs that are used to guide the computation at the object-level. This type of meta-program gives rise to a trade-off situation, in which costs should be compared with benefits. The benefit of meta-computation is that it leads to a better choice among object-computations, and therefore to savings at the object-level, since useless or expensive object-computations can be avoided (see e.g. [1] for results in the area of theorem proving). On the other hand, meta-computations themselves often have a considerable cost, and this cost might offset any savings that are obtained by that very same computation.

This trade-off (between savings made by meta-level choices and the costs of having to make these choices) has been recognised in the literature: [5], [2] and [4, chapter 7] report on *experimental results* on measuring the size of the meta-level overhead, and the large literature on partial evaluation tries to reduce the size of this overhead.

The goal of this paper is to investigate a *theoretical model* of the costs and benefits of meta-computation. After setting out the formal assumptions that underlie this work (Sect. 2), we present in Sect. 3 a quantitative model developed by [6]. In the context of this model, we postulate some reasonable properties of

meta-computations (Sect. 4), and illustrate the model with some examples (Sect. 5). In Sect. 6 we extend and generalise the basic model from Sect. 3.

2 Assumptions

We will assume that there are two independent methods for solving a particular object-level problem, and we will call these methods x and y[1]. We also assume that each of these methods has a certain expected cost, which we will denote by c_x and c_y. Furthermore, we assume that x and y are heuristic methods, i.e. they are not guaranteed to solve the object-problem. Instead, we will assume that each method has a specific chance of solving the given object-problem, which we will write as p_x and p_y.

The goal of the meta-computation is to choose among the two object-methods x and y in order to solve a given problem in such a way that the overall cost of the object-computation is minimised. Because in general x and y are not guaranteed to solve the problem (p_x and p_y might be smaller than 1), the meta-computation must not choose between either x or y, but it must choose the ideal ordering of x and y: first try x, and if it fails, then try y, or vice versa. We will write $c_{x;y}$ to denote the expected cost of first executing x followed by y if x fails, and similarly for $c_{y;x}$.

The meta-computation that determines this choice will again have a certain cost, which we will write as c_m. Again, we will assume that this meta-computation is heuristic, i.e. it will make the correct choice of how to order x and y only with a certain chance, which we will write as p_m.

We assume that without meta-level computation, the system will try to use the two methods in a random order to solve the object-problem. The goal of the model will be to compute the savings (or losses, i.e. negative savings) that are obtained by using meta-computation to choose the ordering of x and y instead of making a random choice[2]. These expected savings will be denoted by s.

All of this notation (plus some additional notation used in later sections) is summarised in Table 1.

3 The Savings Function

In this section, we will derive the expression for the expected savings obtained by making a correct meta-level choice concerning the optimal order in which to execute two object-level methods.

Given the assumptions about x, y, c_x, c_y, p_x and p_y, the expected cost of executing x before y, $c_{x;y}$ is:

$$c_{x;y} = c_x + (1 - p_x)c_y \tag{1}$$

[1] In Sect. 6, we will show how the model can be extended to deal with an arbitrary number of methods.

[2] In Sect. 6 we will show how the model can be adjusted to accommodate the more realistic assumption that the system will execute the methods in some fixed order if no meta-computation is done.

Table 1. summary of notation

Notation	meaning
x,y	object-methods
c_x,c_y	expected cost of object-methods
$c_{x;y}$	expected cost of first executing x then y
p_x,p_y	chance that object-method will solve problem
c_m	cost of meta-computation
p_m	chance that meta-computation makes the correct choice
s	expected savings made by meta-computation
Δ	difference in expected costs between the two object scenarios
$\phi(x)$	utility of method x

namely the expected cost of executing x plus the expected cost of executing y, but reduced by the chance that y is not executing because x has succeeded in solving the problem. An analogous expression holds for $c_{y;x}$. Notice that (1) reduces to simply the expected cost of c_x when x is a complete method (i.e when $p_x = 1$).

The decision to try x before y should be made when

$$c_{x;y} < c_{y;x} \tag{2}$$

or equivalently, using (1):

$$p_y/c_y < p_x/c_x. \tag{3}$$

The quantity $\phi_x = p_x/c_x$ can be seen as a measure of the *utility* of method x. The utility of a method x increases with its success rate p_x and decreases with its expected costs c_x. The above inequality (3) says that the method with the highest utility should be tried first.

However, the values for success rates and expected costs of x and y (and therefore the values of $\phi(x)$ and $\phi(y)$) will not in general be available to the system, and will have to be computed at the cost of some meta-level effort, c_m. Once the meta-level has estimated $\phi(x)$ and $\phi(y)$, the optimal ordering of the two methods can be determined. We can now derive the expteced savings s made by executing the methods in this optimal order as follows: we assume that without any meta-level effort, the system chooses a random ordering of x and y. The expected savings are then the cost of executing a randomly chosen ordering minus the cost of executing the methods in the optimal ordering, increased with the cost of finding the optimal ordering:

$$\begin{aligned} \text{savings} = \text{cost-of-random-ordering} &— \\ &(\text{cost-of-chosen-ordering} + \text{meta-level-cost}) \end{aligned}$$

and the expected cost of executing the methods in a random order is:

$$\frac{c_{x;y} + c_{y;x}}{2}. \tag{4}$$

If the system spends c_m on meta-level effort and then chooses x before y as the optimal ordering, the expected execution cost would be:

$$c_{x;y} + c_m. \tag{5}$$

The expected savings would then be the difference between these two formulae:

$$\frac{c_{y;x} - c_{x;y}}{2} - c_m. \tag{6}$$

This would be the expected savings if the system preferred x over y on the basis of its estimates of $\phi(x)$ and $\phi(y)$. In general, we cannot expect that the meta-level will always succeed in computing the true values of $\phi(x)$ and $\phi(y)$. We can adjust our model to the assumption that the meta-level prefers x over y (i.e. it claims $\phi(x) > \phi(y)$), but that this decision is only correct with a probability p_m. In this case, the expected execution costs would be

$$p_m c_{x;y} + (1 - p_m) c_{y;x} + c_m. \tag{7}$$

If we subtract this from the costs of executing a random ordering, and simplify the result, we get:

$$\frac{(2p_m - 1)(c_{y;x} - c_{x;y})}{2} - c_m \tag{8}$$

These are the expected savings when the meta-level prefers x over y. An analogous expression holds for the reverse case. We can combine these two expressions to obtain the following expression for the expected savings s made by a meta-computation:

$$s = \frac{(2p_m - 1)\Delta}{2} - c_m = (p_m - \tfrac{1}{2})\Delta - c_m \tag{9}$$

where Δ is notation for $|c_{x;y} - c_{y;x}| = |p_y c_x - p_x c_y|$. The intuitive interpretation of Δ is that it represents the difference in costs between the optimal and the non-optimal object-scenario's (executing first x and then, if necessary, y, or vice versa). This concludes the derivation of the savings function (which closely followed [6]).

An alternative derivation for s is as follows: the expected gains made by making the correct choice between $x; y$ or $y; x$ are $|c_{x;y} - c_{y;x}| = \Delta$. If the chance of a correct meta-decision is only p_m, the maximal expected gains are reduced to $p_m\Delta$. The savings made by a random choice would already have been $\frac{1}{2}\Delta$, reducing the expected savings made through meta-computation to $p_m\Delta - \frac{1}{2}\Delta$. The costs of the meta-computation must also be subtracted from this, yielding a total of:

$$s = p_m\Delta - \tfrac{1}{2}\Delta - c_m = (p_m - \tfrac{1}{2})\Delta - c_m \tag{10}$$

From this value for the savings function s we can already draw some conclusions. The form $s = (p_m - \frac{1}{2})\Delta - c_m$ shows that even an ideal meta-level computation (with perfect results, $p_m = 1$ and no costs, $c_m = 0$), can at best only save half the difference in expected object-costs ($s = \frac{1}{2}\Delta$). This is already a severe limit on what meta-level computations can gain.

This maximal savings of $\frac{1}{2}\Delta$ also puts an upper limit on the maximum amount of meta-effort that we should invest: because the expected savings can never amount to more than $\frac{1}{2}\Delta$, it will certainly never be useful to make c_m larger than this same amount. Thus, any potential losses by spurious meta-computation could be limited given an estimate of Δ.

Furthermore, we see that s is monotonically increasing with Δ, and this makes sense: the larger the difference in expected costs between the different object-computations, the larger the expected savings to be made by meta-computation. In the boundary case, when $\Delta = 0$ and the ordering of object-computations is irrelevant, the expteced savings will be negative (since in that case, $s = -c_m$), and meta-computations will lead to a loss in overall efficiency.

Finally, we see that when $p_m = \frac{1}{2}$, i.e. when the meta-level decision does not improve over a random choice, meta-computation will again only lead to a loss in overall efficiency ($s = -c_m$), as expected.

4 Properties of Meta-computations

In realistic situations, the value of p_m, the probability of making a correct choice between methods, will be dependent on the amount of meta-level effort spent, i.e. $p_m = f(c_m)$. Placing this in (9) above, we get:

$$s(c_m) = (f(c_m) - \tfrac{1}{2})\Delta - c_m \tag{11}$$

Obviously, we want to maximise s as a function of c_m. Exactly what shape $s(c_m)$ will have will depend on how the accuracy of the meta-computation (i.e. p_m) depends on the meta-level effort spent by the system, i.e. the shape of $f(c_m)$. The following are reasonable assumptions to make about $f(c_m)$:

Monotonicity: We expect of a meta-level that the quality of its results does not decrease with increased effort, making $f(c_m)$ non-decreasing:

$$\frac{df}{dc_m}(c_m) \geq 0. \tag{12}$$

Lowerbound: With no meta-level effort, i.e. $c_m = 0$, we assume $p_m = \frac{1}{2}$ (because of the random ordering of x and y):

$$f(0) = \tfrac{1}{2}. \tag{13}$$

Upperbound: Since $f(c_m)$ is a probability, we certainly expect $0 \leq f(c_m) \leq 1$. Together with (12) and (13) this gives:

$$\tfrac{1}{2} \leq f(c_m) \leq 1. \tag{14}$$

This upperbound on $f(c_m)$ immediately makes clear that any meta-level computation is eventually doomed to lead to losses:

$$\lim_{c_m \to \infty} s(c_m) = -\infty \tag{15}$$

since the positive term in (9) is limited to $\frac{1}{2}\Delta$, the negative term $-c_m$ will eventually dominate.

Diminishing returns: Finally, although this is not strictly necessary, we can expect some effect of diminishing returns, giving a smaller increase in p_m for every further increase in c_m:

$$\frac{d^2 f}{d(c_m)^2}(c_m) \leq 0. \tag{16}$$

5 Examples

For illustrational purposes, it is interesting to look at a number of example functions for $f(c_m)$ which have the above properties, and to see what the actual shape of the savings curve $s(c_m)$ would be for these functions. The first two examples are taken from [6].

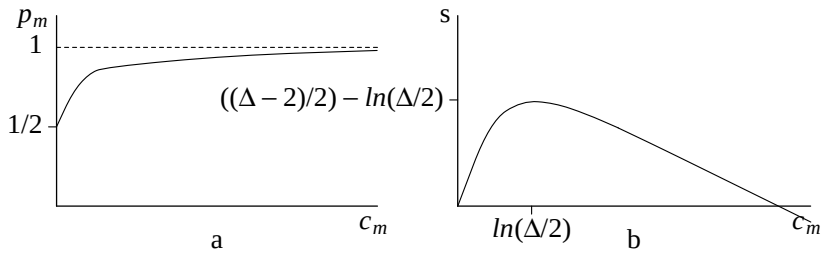

Fig. 1. example of diminishing returns in p_m

Example 1: Figure 1a shows the case for

$$p_m = f(c_m) = 1 - \tfrac{1}{2}e^{-c_m}. \tag{17}$$

The only significance of this function is that it obeys conditions (12)–(16), but otherwise it is arbitrary. It serves to illustrate the case where a relatively small amount of meta-level effort results in substantially improved performance, while there are diminishing returns for subsequent effort. Its convergence to $p_m = 1$ for $c_m \rightarrow \infty$ is arbitrary, and does not influence the qualitative shape of the savings curve for $s(c_m)$ shown in Fig. 1b. In this figure, we can see that at a certain point, the expected savings achieved by meta-computation reach a maximum, and any further meta-level effort will only reduce the overall savings. With even more effort spent on meta-computation, the system will eventually behave worse than without any meta-level effort at all.

It is instructive to notice that at the point of maximal expected savings, the value of p_m is $1 - \frac{1}{\Delta}$ (simply substitute $ln(\Delta/2)$ in (17)). This shows that for large Δ, it pays to spend effort trying to get p_m close to 1, whereas for smaller Δ,

a small value of p_m is all that we can afford to compute. The costs of computing any better approximation of p_m will then not be paid back by the corresponding savings in object-computations.

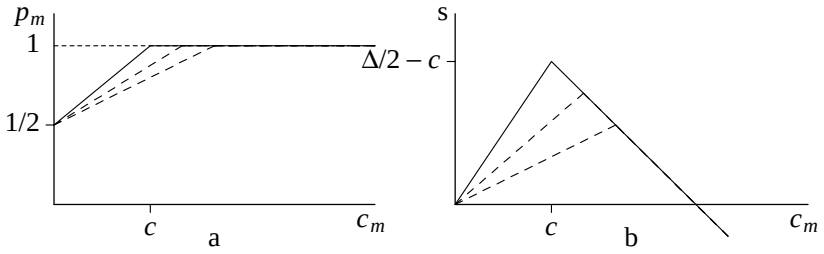

Fig. 2. example of maximised p_m

Example 2: Similarly, Fig. 2a shows an example where increased meta-level effort initially improves the reliability of the choice among object-methods, but after some point ($c_m = c$) the decision is made with maximum reliability (in this case perfect reliability, $p_m = 1$), and any further meta-level effort does not contribute to a more effective control decision. The function used in this figure (shown as a solid line) is:

$$p_m = f(c_m) = \begin{cases} \frac{1}{2}(c_m/c) + \frac{1}{2} & \text{if } c_m \leq c \\ 1 & \text{if } c_m > c \end{cases} \tag{18}$$

Again, the only crucial properties of this example for $f(c_m)$ are conditions (12)–(16),. Other properties, such as the slope of $f(c_m)$ on $[0, c]$, or the fact that $f(c_m) = 1$ on $[c, \infty)$ are irrelevant to the qualitative shape of the saving curve shown as the solid line in Fig. 2b, where again the expected savings reach a maximum at some point, beyond which further meta-level effort will only degrade the performance of the system.

Example 3: Whereas the previous examples showed meta-computations which improved the behaviour of the control decision gradually (in accordance with (16)), a final example shows what happens if meta-level reliability improves suddenly after some initial effort.

Figure 3a shows the case of a meta-level which needs some initial effort to compute a good choice between object-computations, but whose quality does not improve after having made its decision:

$$p_m = f(c_m) = \begin{cases} \frac{1}{2} & \text{if } c_m \leq c \\ 1 & \text{if } c_m > c \end{cases} \tag{19}$$

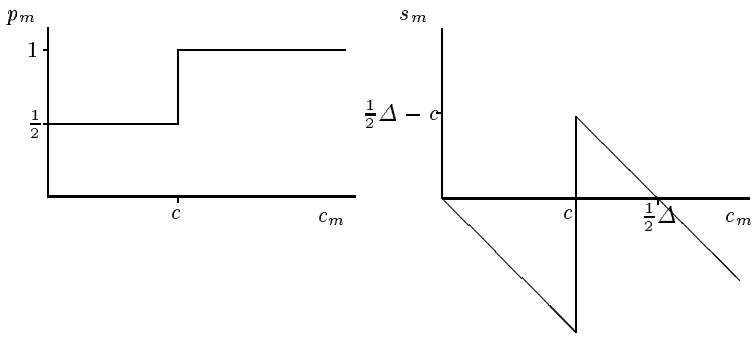

Fig. 3. example of suddenly increasing p_m

The corresponding savings curve is shown in Fig. 3b. The interesting property of this curve is that the expected savings are initially negative ($0 \leq c_m < c$), and only become positive after a required minimum amount of effort. This is an important difference with the previous two examples: there, insufficient meta-effort might lead to non-maximal savings, but would never lead to actual losses, whereas the meta-computation of this third example does indeed yield losses at insufficient meta-effort. This third example shares with the previous two examples the property that too much meta-effort leads to reduced gains, and eventually to losses over no meta-computation at all.

6 Extensions

The simple model above can extended in a number of ways.

6.1 Complete Methods

The first extension is not really an extension, but rather a special case: our current model deals with heuristic methods which would have to be tried in sequence and we therefore have to take into account the expected costs of executing the second method in those cases where the first method failed. We might also consider choosing between complete object-methods, which always succeed. The choice is then not between executing first x and then y or vice versa, but between executing either x or y. The model deals with this situation as a simple special case. We simply take $p_x = p_y = 1$. Then Δ reduces to $|c_x - c_y|$, i.e. the difference between the expected costs of executing *either* x or y as expected .

6.2 Harder Meta-level Problems

We call one meta-level problem *harder* than another if for the same amount of meta-level effort c_m, the system achieves a lower value of p_m (i.e. the choice between the applicable methods is made less reliably).

In the case of the definition for $f(c_m)$ used in example 2, harder meta-level problems are represented by an increasing value of c, and are indicated by the family of dashed lines in Fig. 2a. The corresponding behaviour of the saving function $s(c_m)$ is shown by the dashed lines in Fig. 2b. We see that when the meta-level problem gets harder, the optimum meta-level effort is found for a larger value of c_m, and the corresponding savings are reduced.

This behaviour illustrates a phenomenon often observed in developing meta-level systems, namely that the usefulness of meta-level inference cannot be illustrated adequately on very simple toy problems. For those problems, c will be very small, and any significant amount of meta-level effort is likely to be larger than c, and will thus overshoot the point of maximum expected savings.

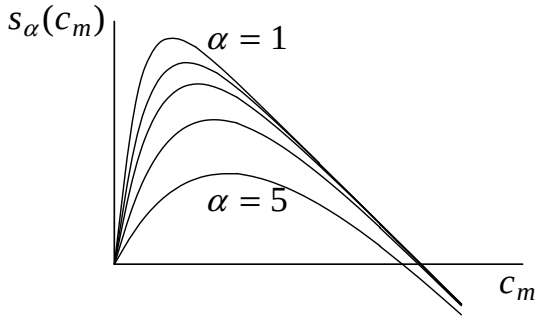

Fig. 4. savings for harder meta-problems

A similar set of curves can be drawn for the example used in Fig. 1. If we take

$$p_m = f(c_m) = 1 - \tfrac{1}{2}e^{-c_m/\alpha} \tag{20}$$

(instead of $p_m = f(c_m) = 1 - \frac{1}{2}e^{-c_m}$), then increasing values of $\alpha \geq 1$ will represent harder meta-level problems. Again, the maxima for $s(c_m)$ for different values of α lie at increasing values of c_m, and the corresponding savings are reduced. An example of this behaviour is shown in Fig. 4 which displays different curves $s(c_m)$ for values $\alpha = 1, 1\frac{1}{2}, 2, 3, 5$.

The same phenomenon occurs in example 3, where again, increasing values of c represent harder meta-level problems, which again lead to lower maximum expected savings, obtained at higher values of c_m. An interesting additional effect is shown in this example: harder meta-level problems not only lead to reduced maximum expected savings, but the savings function is also positive on a smaller interval, making the balance for meta-computation even more delicate.

6.3 Number of Methods

Rather than modelling just two methods x and y, we can adjust the model to choose between n methods (a suggestion also made by [6]). Expression (1) can be generalised from 2 to n methods, so that the expected cost of executing methods $x_1, \ldots, x_n$ in that order is:

$$c_{x_1;\ldots;x_n} = c_{x_1} + \sum_{i=2}^{n}(\prod_{j=1}^{i-1}(1 - p_{x_j}))c_{x_i} \tag{21}$$

(i.e. the total cost is the sum of the cost of each method multiplied with the chance that all earlier methods failed). The optimal order for executing the methods would of course be some order $x_{i_1}; \ldots; x_{i_n}$ such that for all j, $1 \leq j \leq n-1 : \phi(x_{i_j}) \geq \phi(x_{i_{j+1}})$

Before we can derive the savings function for this generalised case, we must introduce some notation. The methods $x_1, \ldots, x_n$ can be executed in $n!$ different orders, and we shall denote each such sequence by some σ_j, $1 \leq j \leq n!$. Then c_{σ_j} is the expected cost associated with executing such a sequence (written as $c_{x_k;\ldots;x_m}$ in (21)). Furthermore, let us assume that σ_1 is the optimal order (i.e. it satisfies $\phi(x_k) \geq \phi(x_l)$ whenever x_k occurs earlier in σ_1 than x_l). With this notation, we can derive the savings function analogously to the case $n = 2$ in Sect. 3. The expected cost of executing the methods $x_1, \ldots, x_n$ in a random order (i.e. the generalised version of (4)) is:

$$\frac{\sum_{j=1}^{n!} c_{\sigma_j}}{n!} \tag{22}$$

Equation (7), the expected cost of executing the optimal method (σ_1), chosen by the meta-level with probability p_m, now becomes:

$$p_m c_{\sigma_1} + (1 - p_m)\frac{\sum_{j=2}^{n!} c_{\sigma_j}}{n! - 1} + c_m \tag{23}$$

namely: the cost of σ_1 times the probability that σ_1 was chosen, plus the chance that any of the other methods was chosen times the average cost of one of the other methods, plus the cost of making the meta-level choice.

As before, the expected savings of a meta-computation is the difference between these two formula. Subtracting (23) from (22) gives, after simplification:

$$s = (p_m - \frac{1}{n!})\Delta - c_m \tag{24}$$

where

$$\Delta = \frac{\sum_{j=2}^{n!} c_{\sigma_j}}{n! - 1} - c_{\sigma_1}. \tag{25}$$

Equation (25) shows us that the proper interpretation of Δ is the difference between the cost of the correct choice and the average cost of an incorrect choice.

In the case $n = 2$, this reduces to the difference in expected costs between the two possible sequences, $c_{\sigma_2} - c_{\sigma_1}$, written as $c_{y;x} - c_{y;x}$ in Sect. 3.

From the generalised savings function shown in (24), we see that the maximal expected savings are $(1 - \frac{1}{n!})\Delta$. If we assume that Δ stays more or less constant with increasing n (since there is no reason why the average cost of a set of methods must increase or decrease for larger sets), we can conclude that for increasing n, the maximal expected savings will increase and will rapidly approach Δ (within 1% for $n = 5$). This gives a slightly rosier perspective on the maximal expected savings to be made by meta-computation than originally derived in Sect. 3, where the expected savings were limited to $\frac{1}{2}\Delta$, namely $(1 - \frac{1}{n!})\Delta$ for $n = 2$.

6.4 Initial Ordering of Methods

The model assumed that with no meta-level effort, the system would make an arbitrary choice for the order in which it executed its object-methods x and y. A more realistic assumption would be that the system would apply x and y in some fixed order, say first x and then y, on the basis of some a priori knowledge that the system's designer has about $\phi(x)$ and $\phi(y)$. Suppose that with no meta-effort, the system chooses x before y, and that this choice is indeed the right one with a chance p_0. In other words, the value $f(0)$, the quality of the meta-level decision at no meta-effort, is no longer $\frac{1}{2}$, as specified in (13), but is now p_0. Presumably, $\frac{1}{2} < p_0 \leq 1$, since the fixed ordering programmed into the system will be better than a random ordering. Because (13) has changed to

$$f(0) = p_0 \tag{26}$$

Formula (14) must also change, into

$$p_0 \leq f(c_m) \leq 1. \tag{27}$$

To derive the new value of the savings function, we follow the derivation of (10) above. In this derivation, we subtracted the savings made by a random choice ($\frac{1}{2}\Delta$). This term must now be replaced by the expected savings made by an informed initial choice, which are $p_0\Delta$. This leads to the following new savings function:

$$s(c_m) = f(c_m)\Delta - p_0\Delta - c_m = (f(c_m) - p_0)\Delta - c_m \tag{28}$$

Notice that this reduces to (9) for the case $p_0 = \frac{1}{2}$ (the random choice case).

Because of the new boundary condition (26), this again implies $s(0) = 0$, as before. Furthermore, if $p_0 = 1$ is (and the initial ordering is already perfect), then (27) implies $f(c_m) = 1$, and the savings function reduces to $s(c_m) = -c_m$, and any meta-computation only leads to losses, as expected.

Finally, the value of (28) is always smaller than the value of (9), because $p_0 > \frac{1}{2}$. For the same reason, the maximum expected savings that can be obtained by a meta-computation are reduced from $\frac{1}{2}\Delta$ to $(1 - p_0)\Delta$. This is just what one would expect, since without meta-effort, the system performs better than it did before, and as a result, the potential savings that can be made by the meta-computation are smaller.

6.5 Cost Dependent on Success or Failure

The assumption above was that a method, say x, had some associated expected cost c_x, which was independent of whether the method succeeded or failed. This assumption can be lifted by introducing two costs, namely c_x^s, the expected cost of x if it succeeds, and c_x^f, the expected cost of x if it fails. The expected cost of executing method x is then:

$$c_x = p_x c_x^s + (1 - p_x) c_x^f \tag{29}$$

namely the cost of x succeeding times the probability that it will succeed plus the cost of x failing, times the probability that it will fail. We can then uniformly substitute this new expression for any c_x in the above, and similarly for y. If we do this in (1), representing the expected cost of executing first method x and then method y, the resulting expression can be rewritten to the following:

$$c_{x;y} = p_x c_x^s + (1 - p_x)(c_x^f + p_y c_y^s + (1 - p_y) c_y^f) \tag{30}$$

which is exactly what we expect, namely the cost of x succeeding times the probability that x succeeds plus the sum of x failing and executing y times the probability that x fails. Notice that this equation reduces to (1) when the distinction between success and failure costs is irrelevant (i.e. when $c_x^s = c_x^f$).

If we substitute (29) into the definition of the utility $\phi(x)$, we obtain:

$$\phi(x) = \frac{p_x}{p_x c_x^s + (1 - p_x) c_x^f} = \frac{1}{c_x^s + \frac{(1-p_x)}{p_x} c_x^f} \tag{31}$$

From this we can see that the utility of x decreases if either failure costs or success costs increase, or when the success chance decreases. This is again as expected, and similar to the old value of $\phi(x)$.

Furthermore, we see that when $p_x > \frac{1}{2}$, then an increase or decrease in c_x^s has a larger effect on $\phi(x)$ than an equal increase or decrease in c_x^f. Thus, for a method which often succeeds, $\phi(x)$ is dominated by the success costs, and similarly the utility for mostly failing methods is dominated by the failure costs.

More surprisingly, (31) also shows that for methods with very low failure costs c_x^f, the utility $\phi(x)$ becomes simply inversely proportional to the success costs, and independent of the success rate. This is surprising because in the case of two methods with zero failure cost (thus: $c_x^f = c_y^f = 0$), but different success rates (say $p_x > p_y$) we would expect x to be preferred over y, but (31) fails to capture this.

A final, and perhaps somewhat counterintuitive implication from (31) would be the following. Suppose that method x fails by running into an infinite loop (e.g. because of a loop in the search space). If we model this by $c_x^f = \infty$, this immediately means $\phi(x) = 0$, implying that x is always to be selected as the last method. The counterintuitive aspect of this is that this conclusion is again independent of the success chance.

7 Discussion

The problem of choosing the correct amount of meta-effort in order to optimise the total savings is very similar to the problems tackled in classical decision theory (e.g. [7]). A number of the effects that we have observed in the models above are also well known in decision theory. In particular, the fact that the maximal savings are limited to $\frac{1}{2}\Delta$ (in the case $n = 2$) is known as the *base-rate effect*: the gains of an informed decision are limited if a random choice already performs well (when the "base-rate" is already high), as in the case of $n = 2$. The reduction of the base-rate and the corresponding increase in maximal savings for increasing n (to $(1 - \frac{1}{n!})\Delta$) is also a known effect.

The *law of diminishing returns* (in our case modelled by assumption (16)) is also familiar from decision theory. Decision theory also gives us an explanation of the counterintuitive results at the end of the previous section for the case of zero or infinite failure costs. In decision theory this is called the *boundary effect*: if one of the dimensions in a multi-dimensional decision problem becomes 0 or ∞, then the model is no longer valid because this single dimension will entirely dominate all the other dimensions. In such cases, we should move to a new model, rather than simply substitute 0 or ∞ in the old model.

An interesting application of our theoretical model can possibly be found in the area of explanation-based learning. A well-known problem in that area is the so-called *utility problem* [3]: learning more control knowledge does not always lead to more efficient problem-solving behaviour, and learning too much may actually adversely affect efficiency. At first sight, it would seem that this effect could be explained by a model similar to ours. Further research is required to shed lighton this issue.

Finally, a remark is in order on the usefulness of our model. One might object that the above results are not very useful since in practice it will be very hard, if not impossible, to obtain the actual values for the costs and probabilities involved in a particular system. The main value of our model is therefore a different one: rather than using it to compute numerical values in a concrete system, it teaches us general lessons about the qualitative behaviour of a whole class of meta-systems in general, lessons which are sometimes surprising and perhaps not obvious at first sight.

8 Conclusions

We can summarise the main conclusions from this model as follows:

- The model shows that a *trade-off* does indeed exist between the costs of meta-computation and the savings obtained by it. This results in a situation where savings are maximised at a certain amount of meta-effort. Both more and less meta-effort will lead to non-maximal savings.
- Meta-computation may also lead to a *loss* in efficiency (when compared to doing only object-computation). This will always happen if the meta-effort

becomes greater than a certain maximum (too much meta-effort), but might also happen in certain cases if meta-effort is less than a certain minimum (not enough meta-effort). The loss of overall efficiency at high amounts of meta-effort explains the phenomenon that the usefulness of meta-computation is often hard to illustrate on the basis of small examples.
- An important quantity in the analysis of meta-computation is the difference in the costs of alternative object-computations (Δ in the above). The savings to be made by a meta-computation are proportional to Δ, and the maximum savings (and consequently an upperbound for the amount of meta-effort) are Δ, and even less than that in the case of an informed initial ordering.

The overall conclusion from all of this is that meta-computations are not an unqualified blessing, and that the savings to be made from such computations are often less than expected, and only occur within a narrowly defined interval of meta-effort. Before one decides to use meta-computation as a means to control object-computation, a careful analysis of crucial quantities like Δ, ϕ and $f(c_m)$ is required.

Acknowledgement: I am grateful to Annette ten Teije for her detailed comments and suggestions. She also provided the derivation for the case $n > 2$.

References

1. J. Hesketh A. Bundy, F. van Harmelen and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991. Earlier version available from Edinburgh as DAI Research Paper No 413.
2. H. Lowe. Empirical evaluation of meta-level interpreters. Master's thesis, Department of Artificial Intelligence, University of Edinburgh, 1988. (M.Sc. Thesis).
3. S. Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42:363–392, 1990.
4. R. A. OKeefe. *The Craft of Prolog.* MIT Press, Massachussetts, 1990.
5. S. Owen. The development of explicit interpreters and transformers to control reasoning about protein topology. Technical Memo HPL-ISC-TM-88-015, Hewlett-Packard Laboratories Bristol Research Centre, Bristol, U.K., 1988.
6. J.S. Rosenschein and V. Singh. The utility of meta-level effort. Report HPP-83-20, Stanford Heuristic Programming Project, March 1983.
7. D. von Winterfeldt and Ward Edwards. *Decision Analysis and Behavioral Research.* Cambridge University Press, 1986.