The Ultimate Solution Approach to Intractable Problems

Abdellah Salhi

University of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK,
as@essex.ac.uk,
WWW home page: http://www.essex.ac.uk/maths/staff/profile.aspx?ID=1273

Abstract. There is now strong belief that $P \neq NP$. This means that some very common problems cannot be solved efficiently under current and so called Von Neumann type computer architectures including parallel configurations. And, this will remain the case even in relatively low dimensions. What one may hope to achieve is the best possible solution given the available facilities within the allowed time. This makes the current definition of the optimum redundant for practical purposes. Therefore, a new definition of the optimum is required as well as appropriate approaches to find it.
This paper will put forward a definition for the practical or sensible optimum, the s-optimum, consider its consequences and suggest what can be the ultimate approach to finding it. Although this approach is generic and can be applied in any context, optimisation and search are the specific contexts we will be concerned with here.

1 Introduction

One of the Clay Mathematics Institute's seven Millennium Prize Problems is whether $P \neq NP$, where P stands for the class of problems for which an answer can be found in polynomial-time and NP for the class of problems for which a guessed solution can be checked in polynomial-time. The NP-Complete subclass of NP contains those problems, in decision form, i.e. a simplified form requiring a "yes" or "no" answer, for which the solution cannot be found in polynomial-time, [8]. This means that even in relatively low dimensions, the solution cannot be found in realistic times. They are therefore considered to be difficult (intractable) as opposed to problems in the class P, which are considered easy.

The problem would be of little importance if it only concerned a few and not relevant situations. Unfortunately, NP-Complete problems are very common. The Travelling Salesman Problem (TSP), perhaps the most famous intractable problem, is one of them. So are Satisfiability, Set Covering, Set Packing, cases of Machine Learning and Clustering, to name a few.

Although establishing that $P = NP$ could have serious consequences for internet security for instance, it will also make life easier and cheaper for all of us. In fact, the positives will almost certainly outweigh the negatives. There is, however, consensus among experts, despite a lack of proof, that the contrary is likeliest, i.e. $P \neq NP$.

This belief is being reinforced quite frequently. For instance, it has recently been reported (New Scientist Series, Guardian of 10 August 2010) that, a mathematician at Hewlett-Packard Labs, Palo Alto, California, claimed that he has shown that the Satisfiability Problem, a famous version of which, the 3-SAT, is NP-Complete, cannot be solved in polynomial-time. That in itself would not have made it to the popular press if it was just a claim like many others before it. But, it seems as if people like Stephen Cook who showed first that 3-SAT is NP-Complete, take the claim rather seriously.

It is still a leap of faith to accept that $P \neq NP$, before a proof has been produced and accepted. But, I take a pragmatic stance here, and trust that it is unlikely that an efficient algorithm will be discovered for any of the NP-Complete problems in the near future. Consequently, I suggest that, we be realistic when we face some instances of any of these problems and only expect to find approximate solutions. However, we should make sure that the best approximate solution is found given the means we have at our disposal.

The means we have at our disposal are of two types: hardware and software. The hardware aspect is outside the scope of this paper. It is, therefore, assumed that the hardware facilities are set up in the best way they can be and that the user has no leverage on changing them or resetting them except using them to run his/her programmes. "Software" side is considered, here, to be synonymous with the algorithmic side in a given application.

Having limited ourselves to the algorithmic side, assuming that the available hardware is optimally set for use, we will proceed in this paper to define a pragmatic optimum solution which we call s-optimum. The consequences of this definition will be considered and an approach that will guarantee to find the r-optimum will be suggested.

This approach will rely on Game Theory (GT), and particularly the Prisoners' Dilemma (PD) and its iterated version (IPD), for implementation purposes.

The paper is organised as follows. Section 2 introduces the s-optimum. Section 3 deals with the issue of problem formulations and their importance. Section 4 suggests a comprehensive classification of approaches to the optimisation problem. This will help understand what we mean by the ultimate approach. Section 5 describes how to achieve the s-optimum. It also gives the approach in algorithmic form. Section 6 discusses implementation issues and explains why the IPD is a suitable paradigm for implementing the approach. Section 7 is the conclusion and other suggestions.

2 On *s*-optimality and the redundancy of absolute optimality

2.1 Is absolute optimality redundant?

Modelling real world applications taking into account nonlinearity and/or the combinatorial aspects of solution sets almost always leads to intractable optimisation problems of the global type, [4]. In some cases, even checking a given solution for optimality is intractable. Consequently, it is not reasonable, to expect the global optimum to be found in acceptable times. In other words, there is no point in aiming for what is not achievable. That is why, in fact, heuristics are now the favoured approaches to the solution of these intractable problems. The consequence of this is that the standard definition of the (absolute) optimum (min problem), i.e.

$$x^* | f(x^*) \leq f(x), \forall x \in X,$$

where X is the search space, is rather redundant.

The same can be said of the Pareto-optimum in the case of multiple objectives.

Recall that a solution is Pareto-optimal (or efficient) if there is no other solution for which at least one objective has a better value while values of the remaining objectives are the same or better.

2.2 *S*-optimality: definition

Since, for large and practical instances of intractable problems of the optimisation type the absolute optimum is out of reach with the available means, we should aim for a realistic, sensible and usable approximate solution. Call it the *s*-optimum.

Definition: A solution is *s*-optimal if there is no other solution found by all available means, and within the required time, that gives a better value to the objective function to be optimised. In other words, it is the best solution found with the available means, within the required time.

Note 1: "Available means" will be defined later.

Note 2: The time constraint is already implicit in most stopping criteria of heuristic approaches (max number of iterations, max number of generations, cycles, reducing a parameter value to a certain level, such as temperature in SA, for instance).

Remark 1: The s-Pareto-optimum can be defined similarly.

With this definition in mind, it can be stipulated that:

Proposition 1: The s-optimum of any intractable problem is achievable.

Proposition 2: The optimum use of available facilities results in the s-optimum.

Proposition 3: The converse of Proposition 2 is not true.

3 Problem formulations

In the last two to three decades it has become clearer that problem formulation is an important aspect of problem solution. In mathematical programming this has been championed by H.P.Williams, [9]. Indeed, it is not difficult to see that most problems can be written down in their general form in different ways. Although these different formulations may capture all aspects of the problem, they may emphasise some more than others, they may be more or less compact than others and, crucially, they may be more suitable for an algorithm than an other. The latter may be a result of chance, but could be intended since the formulation may be constructed with a specific algorithm in mind. This may make other algorithms potentially not suitable for solving the problem in this particular form. Consider the following formulations of TSP, [6, 9].

TSP formulation 1:

$$Min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

Subject to:

$$\sum_{i:(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V \tag{2}$$

$$\sum_{j:(i,j)\in A} x_{ij} = 1 \qquad \forall i \in V \tag{3}$$

$$\sum_{i\in S, j\in S} x_{ij} \leq |S| - 1, \qquad 2 \leq |S| \leq |V| - 2 \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A \tag{5}$$

where

$$\mathrm{x}_{ij} = \begin{cases} 1 \text{ if city j succeeds city i in the tour} \\ 0 \text{ otherwise} \end{cases}$$

$$c_{ij} = \text{cost of travelling from city } i \text{ to city } j$$
$$S = \text{any subset of V with } |S| = \{2, 3, .., |V| - 2\}$$
$$V = \text{set of cities} = \{1, 2, .., n\} \text{ and}$$
$$A = \text{set of arcs} = \{(i, j) \ni i, j \in V, i \neq j\}$$

Formulation 1 has an exponential number of constraints. This is because all possible subtours have to be eliminated. Subtours can involve between 2 and $|V| - 1$ cities. Therefore, a constraint must be written for each one of them. Since the number of subsets of $|V|$ is the power set of $|V|$, which has cardinality $2^{|V|}$, the number of constraints is $O(2^{|V|})$. The model is therefore not suitable to write explicitly even for moderately sized instances of the problem. A more compact form can be as follows.

TSP formulation 2:

$$Min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{6}$$

Subject to:

$$\sum_{i:(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V \tag{7}$$

$$\sum_{j:(i,j)\in A} x_{ij} = 1 \qquad \forall i \in V \tag{8}$$

$$u_i - u_j + |V|x_{ij} \leq |V| - 1 \qquad \forall i, j \in V, i \neq j. \tag{9}$$

In this formulation, variables $u_i, i \in V$ are continuous although they will take integer values in the solution. The u_i variable can be interpreted as the sequence number in which city i is visited. This formulation has $O(|V|^2)$ variables and $O(|V|^2)$ constraints.

Formulation 1 is obviously not suitable for algorithms of the Cutting Plane type, [6], which add new constraints to the model, since the model has already an exponential number of constraints. Formulation 2 is more suitable.

In fact, Formulation 1 is is not suitable for all algorithms that require the full model from the start. Having said that, it is commonly used when applying Branch-and-Bound. However, not all constraints are entrerd initially; they are appended to the model as the solution progresses.

Some problems may be known only with a single formulation. This does not mean that other formulations do not exit. Indeed, there are problems for which there are many different formulations. TSP, for instance, has numerous formulations, [6].

As illustrated above, formulations may have different dimensions, different objectives and different constraints. How good is a formulations is, however, decided generally, at least in integer programming, by how sharp it is, [9]. In other words, it is decided by how big (or small) its the continuous i search space is when the integrality constraints are dropped.

Although some obvious incompatibilities may rule out the use of a given algorithm on a given formulation, in general it is not easy to tell. Only with experience can do so. For this reason, it is not recommended to discard algorithms or formulations outright.

Note that not all solution methods, particularly heuristic-based one, require a closed form formulation. They, often, work on the basic definition of the problem and tap into its search space directly. For example, the Genetic Algorithm (GA), [2], works on tours represented as permutations of city indices.

In the presence of many algorithms and many formulations of a given problem, the issue of matching algorithms to formulations is far from trivial particularly when the user is not an expert. In fact, it is a fundamental decision making problem, at the heart of the approach to be suggested here. This is because, on top of the available hardware (computers and operating system), we consider as integral to the facilities available to solve a problem, all potential solution algorithms (and their codes) as well as all formulations available to the user. In other words, the sum total of what we consider here as facilities are: operational hardware, coded algorithms and formulations. The problem is to use these facilities in an optimal way. A suboptimal solution results from applying a less suitable algorithm to a given formulation.

4 Solution approaches classification

Traditionally, solving a problem boils down to matching an algorithm to some formulation and executing it on the available hardware. This is what we are used to do. However, it is not all one can do and certainly not the best one can do. Given a set of m algorithms and a set of n formulations, four possible configurations are possible.

1. **Single Algorithm Application to a Single Formulation (SASF):** As said above, this is the way we commonly solve problems. Provided the available algorithm is applied properly to the instance of the available problem formulation, the s-optimum is guaranteed, although it may be very remote

from being usable.

2. **Single Algorithm Application to Multiple Formulations (SAMF):** In the presence of more than one formulation, deploying the algorithm on all of them is a better approach, unless the expertise of the user can help to choose the most appropriate formulation. SAMF is particularly suited for a parallel/distributed processing environment, [3].

3. **Multiple Algorithms Application to a Single Formulation (MASF):** In the presence of many algorithms and one formulation, it is better to solve each formulation with the available algorithm, unless the user is capable to choose the most appropriate formulation for the given algorithm. Here too, MASF is best implemented in a parallel/distributed environment.

4. **Multiple Algorithms Application to Multiple Formulations (MAMF):** In the presence of many algorithms and many formulations, the best solution strategy is to deploy every algorithm on every formulation. This is ideal in a parallel environment with at least mn processors. MAMF is the ultimate approach in that it is the one that can achieve the highest quality s-optimum. However, this may depend on its implementation. The same can be said of the other approaches.

Proposition 3: The s-optima delivered by the different strategies are not necessarily the same.

5 Achieving *s*-optimality: optimum use of facilities

By virtue of Proposition 2 above, the s-optimum of any intractable problem can be achieved, provided available facilities are optimally used. The optimum use of available facilities can be reduced to the ideal matching of available algorithms and formulations. Clearly, this is a decision making problem that is not easy to solve even for the expert. This is because even if some algorithm has been designed with some formulation of a problem in mind, problem instances vary in difficulty depending on the data involved. For example, an algorithm that is very efficient on a problem with well-behaved data may be completely unsuitable when the problem data are ill-conditioned. There is a trade-off between efficiency and robustness within most algorithms. Therefore, when facing a new instance of a problem under some formulation, it is not wise to assume that the algorithm which we think is most suitable will do well on that particular instance. In other words, a given algorithm will have different performances on different instances of the same problem formulation. Hence, optimum allocation of algorithms to

formulations instances can only be achieved after these algorithms have been tried on the data for a few iterations/generations/cycles, depending on the algorithm. This is really a case of real-time decision making, i.e. decision making during the solution process.

If only one algorithm and one formulation were available, then the s-optimum can be achieved if the algorithm was run for the allowed time. To fulfil the potential of configurations SAMF, MASF, and MAMF an appropriate implementation is required.

The aims of this implementation are:

- Find the most suitable (algorithm, formulation) pair(s)s and run them on all available processors;
- Exploit the synergy between pairs by making them work together (co-operate) towards the solution;
- Implement competition: allow pairs (algorithm, formulation) to compete for the computing facilities;

These three aims if achieved will deliver the s-optimum.

5.1 Algorithm

1. Allocate each (algorithm, formulation) pair an equal amount of run time (CPU time, iterations, generations etc...) on the available computing platform;
2. Run each pair (algorithm, formulation) for a fraction of its allocated time;
3. Allow algorithms to co-operate by exchanging their current solutions;
4. Rank them according to performance over each period of run time and reward them by increasing their quota of time or punish them by reducing it;
5. Drop algorithms with empty quotas.

This algorithm guarantees that after few iterations/generations/cycles etc... of each algorithm, only the best performing ones remain running on the computing platform. In other words it guarantees the best use of available facilities. Therefore, it finds the s-optimum.
Remark 2: The s-optimum is equal to the classical absolute optimum in the limit.

6 A game theoretic multiple agent system for intractable problems

The above algorithm requires an exchange of information between individual algorithms in order to improve the overall solution process. Unfortunately, poor

algorithms may not have much to offer and therefore may be a burden on the overall system. Competition for the facilities is necessary to allow the most successful algorithms to remain in use while unsuccessful ones are dropped.

An appropriate set up is to consider each pair (algorithm, formulation) as a player in a co-operative/competitive game the aim of which being to find the solution to the problem in hand as quickly as possible.

There are many co-operative/competitive game paradigms of which Stag Hunt, [7], and the Prisoners' Dilemma (PD), [1], are the most prominent. PD, in particular, epitomises the social games where trust is an important ingredient for fostering co-operation, and greed is what guarantees the big returns. PD has been popularised by Al Tucker in the 50's and Axelrod in the last two decades, [1]. The reward/punishment actions are implemented through appropriate pay-off tables.

In Salhi and Toreyen, [5], a Game Theory Multi-Agent System (GTMAS) has been developed and tested with two algorithms on a set of TSP problems. The results show that the approach is effective.

7 Conclusion and further work

We have defined what a "realistic optimum" is and justified the need for it. We have argued the case for the importance of problem formulation and illustrated it on the TSP. We have looked at all possible strategies for achieving the s-optimum depending on the computing facilities (hardware, software) available. These strategies are in four categories, SASF, SAMF, MASF and MAMF. Each of these strategies will deliver an s-optimum if implemented properly. The s-optima found by different strategies are not necessarily the same. MAMF, for obvious reasons, has the potential to deliver the highest quality s-optimum. To fulfil its potential, however, it has to be implemented in such a way that every advantage is taken of all that participating algorithms and formulations have to offer. This can be achieved through co-operation. Moreover, since some (algorithm, formulation) pairs may not have much to offer, they should be discarded once this becomes obvious. Competition for facilities must, therefore, also be implemented. The co-operation/competition aspects of the implementation of MAMF can be realised within a game theory framework of the the Prisoners' Dilemma type. In other words, MAMF can be implemented as a game of the PD-type played by algorithms cast as players. Moreover, these players can have autonomy and be enhanced with some intelligence, making the resulting implementation a multi-agent system, [10]. This has already been achieved as GTMAS in [5].

Consequently, we believe that we have presented what can be seen as the ultimate approach to solving intractable problems and any problem fro that matter. It is generic and should deliver the s-optimum for any problem.

It must be said that this approach is different from standard hybridisation. MAMF and its realisation as GTMAS constitute a dynamic and flexible loosely coupled hybrid system.

GTMAS is a very limited realisation of MASF in that it involves only two algorithms and one formulation. It remains to implement MAMF with more algorithms and formulations to see the extent of its advantages or otherwise. The consequence of having more than two algorithms (players) is that designing appropriate payoff tables for cooperative/competitive games involving a large number of players can be problematic. Perhaps, by allowing coalitions between agents to form, the size of the table may be made more manageable. One can also exploit the preference order that may exist between algorithms based on performance:

$$A1 \succeq A2$$

$$A2 \succeq A3$$

$$\Rightarrow A1 \succeq A3.$$

So, the encounter between A1 and A3 is redundant. However, the fortunes of agents may change over time!!

References

1. Axelrod, R. *The Evolution of Cooperation*. Basic Books, New York, 1984.
2. Holland, J.H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, USA, 1975.
3. Salhi, A. and Glaser, H. and De Roure, D. Parallel Implementation of a Genetic-Programming based Tool for Symbolic Regression Information Processing Letters, 66(6), pages 299-307, 1998.
4. Salhi, A., Proll, L.G., Rios Insua, D., and Martin, J. Experiences with stochastic algorithms for a class of global optimisation problems. *RAIRO Operations Research*, 34(22), pages 183-197, 2000.
5. Salhi, A. and Töreyen, Ö. A game theory-based multi-agent system for expensive optimisation problems. In Y. Tenne and C.-K. Goh, editors, *Computational Intelligence in Optimization*, Chapter 9, pages 211-232. Springer-Verlag, 2010.
6. Nemhauser, G.L., and Wolsey, L.A. Integer and Combinatorial Optimization. Wiley-Interscience Publication, Wiley & Sons, 1988.
7. Battalio, R., Samuelson, L. and Va Huyck, J. Optimisation Incentives and Coordination Failure in Laboratory Stag Hunt Games. Econometrica, 69(3), pages 749-764, 2001.
8. Garey, M.R. and Johnson, D.S. Computer and Intractability: A Guide to the Theory of NP-Completeness. W.H.Freeman, ISBN 0-7167-1045-5, 1979.
9. Williams, H.P. Model Building in Mathematical Programming. Wiley, 3rd Edition, 1990.
10. Park, S. and Sugumaran, V. Designing Multi-Agent Systems: A Framework and Application. Expert Systems with Applications, 28, pages 259-271, 2005.