# Don't settle for less than the best: use optimization to make decisions

Kimberly Keeton, Terence Kelly, Arif Merchant, Cipriano Santos, Janet Wiener and Xiaoyun Zhu
*Hewlett-Packard Laboratories, Palo Alto, CA*
*firstname.lastname@hp.com*

Dirk Beyer
*M-Factor, San Mateo, CA*
*dirk.beyer@m-factor.com*

## Abstract

Many systems design, configuration, runtime and management decisions must be made from a large set of possible alternatives. Ad hoc heuristics have traditionally been used to make these decisions, but they provide no guarantees of solution quality. We argue that operations research-style optimization techniques should be used to solve these problems. We provide an overview of these techniques and where they are most effective, address common myths and fears about their use in making systems decisions, give several success stories and propose systems areas that could benefit from their application.

## 1 Introduction

Decision-making problems abound in systems research, including questions of resource provisioning [6, 16], resource allocation and scheduling [17, 20, 23, 24, 30, 31], system administration and management [8], and application and system design [3, 5, 15, 25]. These problems are characterized by a large space of potential solutions, with complex tradeoffs between system performance, availability, reliability, manageability and cost. Given the large solution space, it's hard to keep the alternatives straight, let alone find the best solution. Even when a "good enough" or merely feasible solution (i.e., one that meets constraints) is desired, it can be hard to find. Furthermore, getting the answer wrong can be costly (e.g., in the time to recover from a disaster or in the monetary expense of over-provisioning physical resources), so there is a strong incentive to choose wisely.

Traditionally, systems researchers have used ad hoc domain-specific heuristics to solve these decision-making problems. Unfortunately, heuristics don't provide the best solution, nor do they provide any bounds on how close their solution is to the best. Recently, a new paradigm has emerged, where systems decision problems are cast as formal optimization or constraint satisfaction problems, allowing the use of operations research (OR) solution techniques, from mathematical programming to meta-heuristics.

In this paper, we argue that the systems community needs to leverage the more principled approach of formal optimization to solve design, configuration, runtime and management decision-making problems [26]. By optimization, we mean first formally specifying the problem, and then using any of several techniques to solve it. Specifying the problem means explicitly defining the objective, the constraints on a valid solution, and how input parameter values impact the goodness of a candidate solution. Formulating these aspects of the decision-making problem forces us to understand the underlying problem and the tradeoffs that we're trying to capture. This knowledge is useful, whether the problem is ultimately solved by standard OR techniques or by domain-specific heuristics.

Standard OR solution techniques provide many benefits. In many cases, these techniques provide optimal answers, which means that researchers don't need to worry that a heuristic might perform poorly for an as-yet-unseen corner case. The speed of current desktop machines makes it possible to use these techniques on many problems that would have been intractable even ten years ago. OR techniques encourage a clean separation between the problem statement and the solution method. Furthermore, the availability of commercial off-the-shelf solvers means that we as systems researchers can focus on specifying the problem at hand, rather than worrying about how to solve it.

In the remainder of the paper, we provide an overview of popular OR techniques and debunk common myths preventing their usage in the systems community. We discuss when to choose an optimization technique instead of an ad hoc heuristic. Finally, we give several success stories where optimization has been applied to systems problems and list several areas that are ripe for optimization in the future.

## 2 Optimization techniques

The OR community presents a variety of techniques to find optimal and approximate solutions to decision problems. To use any of these techniques, the first step is to describe the problem formally: what decisions must be made, which alternatives are feasible, and what the "goodness" metric is for comparing solutions. Decisions might include which outgoing link to use to transmit a message in a wireless routing environment, or whether to allocate a server to workload A or workload B. Constraints on alternatives may be either hard constraints, which cannot be violated, or soft constraints, where violations of the constraint incur penalties. The specific formalism varies with the technique, as described in the rest of this section. One common thread, however, is that all tradeoffs must be expressed in the same currency, such as execution time, throughput, monetary cost, or a "utility" composed from such metrics.

### 2.1 Techniques to find optimal solutions

Techniques such as mathematical programming (MP) provide an optimal answer to a decision problem. Among the approaches we consider, MP requires the most detailed knowledge of the decision problem. In an MP formulation, *decision variables* correspond to the choices to be made, *objective functions* quantify a candidate solution's "goodness," and *constraints* describe which solutions are feasible [11, 29]. The solver then determines an optimal solution, the top $N$ solutions, a solution within $x\%$ of optimal, the best solution possible within a time budget, or simply a feasible solution.

Mixed integer programs (MIPs) are math programs with linear objective functions, constraints defined by linear inequalities, and decision variables that take on continuous or discrete (often binary) values. MIPs where all decision variables have continuous values are called linear programs (LPs). MIPs are appropriate for problems characterized by contention for additive resources and additive measures of system goodness.

### 2.2 Techniques to find feasible solutions

The OR community also provides techniques for finding feasible solutions to decision problems, such as constraint programming [12]. Constraint programming (CP) is an appropriate technique when the constraints can only be expressed by rules — logical statements such as "if you choose option A, you must also choose B, C and D." A constraint satisfaction problem consists of decision variables, each with a domain of valid discrete values, and a set of constraints governing feasible solutions. A solution is a complete assignment of variables that meets all of the constraints. CP is predominantly used to find feasible, rather than optimal, solutions.

### 2.3 Techniques for approximate solutions

Meta-heuristics are algorithms for finding near-optimal solutions, which are inspired by naturally-occurring phenomena, such as genetic algorithms [21], simulated annealing, and auctions. In a genetic algorithm (GA), an individual represents a feasible solution, and the genes of the individual represent decision variables. The most "fit" individuals are selected for the next population based on a fitness function, which is roughly equivalent to the objective function in a MIP. These techniques require less detailed knowledge of a problem's structure because they rely on a procedure, or "oracle," to determine the feasibility and goodness of a candidate solution. This oracle can be an analytic model, a lookup table, or even a simulation. Meta-heuristics provide few, if any, guarantees on the optimality of their solutions. However, it is sometimes possible to state probabilistically how close to optimal a solution is.

## 3 Myths and realities

Although optimization techniques are powerful, systems researchers are often skeptical about applying them to solve decision problems. In this section, we refute several common myths about optimization.

**Myth: A simple heuristic is "good enough."** If an easy-to-implement and quick-to-run heuristic exists, why not use it? If the problem doesn't require an optimal solution, is formal optimization overkill?

**Reality:** If a simple heuristic provides "good enough" answers, then it may be the appropriate choice. The challenge lies in quantifying what "good enough" means and determining if a solution meets it. In many cases, it's hard to determine what "good enough" is, without knowing the best that can be achieved. Even if the goal is balancing tradeoffs between conflicting goals, rather than finding an optimal solution, we still need to understand the relative costs of the alternatives, so that we know whether the appropriate balance is achieved. Without a formal specification, it can be hard to estimate how close to optimal a solution lies; with ad hoc approaches, it can usually be determined only empirically. Techniques like math programming provide a systematic solution with bounds on how close that solution comes to the optimal one. Furthermore, even if optimization techniques work only for small problem instances, their results can be compared with those of domain-specific heuristics, to help understand the heuristic's behavior for larger instances.

**Myth: Problem formulation takes too much time.** Formulating problems is often challenging, and it requires both domain expertise and knowledge of the optimization technique. Employing ad hoc domain-specific heuristics doesn't generally require such up-front, interdisciplinary effort.

**Reality:** The hardest part of problem formulation is understanding the problem — its goals and tradeoffs, as well as how to capture the underlying system's behavior. This first step is required whether the ultimate solution is a standard optimization technique or a domain-specific heuristic. Unfortunately, in the latter case, the explicit formulation step is often ignored, and researchers end up gradually "discovering" aspects of the problem as they successively refine their heuristic. The effort in formulation is well-spent, because it's easier to adapt the formulation as the decision question or constraints change than to adapt an ad hoc heuristic.

**Myth: Formulating the problem requires too many simplifying assumptions.** If too many simplifications are made, then the decision is not realistic, and the resulting solution may be meaningless.

**Reality:** Our collective experience is that simplifications are problematic only when we try to force a problem into a particular framework (e.g., force non-linear behavior into an LP). If one technique doesn't work, we need to try another one, or to break the problem down so that different techniques can be used for different portions of the problem (e.g., a MIP for resource provisioning and a heuristic for resource scheduling). Ultimately, if no optimization technique works, the formal description is still useful for understanding the problem and developing an ad hoc domain-specific heuristic.

**Myth: Optimization techniques are too slow.** Standard optimization techniques take too long to be useful for runtime management decisions.

**Reality:** The execution times of these techniques are highly dependent on the size of the problem and its structure. (For instance, linear programs can be solved more efficiently than non-linear ones.) Execution times can be under a second. Given that many decisions will be in effect for days or months, many decision-making problems can tolerate the execution times of OR techniques.

**Myth: Inaccurate input data may result in bad decisions.** Variations in the input values may cause variations in optimal solutions.

**Reality:** Sensitivity to the input values is a characteristic of the problem domain, rather than the solution technique. If we can't estimate input values with high accuracy, for example, because they are estimates of business utility, then it's important to do a sensitivity analysis to understand how the optimal solution varies with different input values.

**Myth: Optimal solutions may not be easy to support.** Optimal solutions may use non-standard configurations for a large hardware or software system, which may be hard to maintain.

**Reality:** It's difficult to capture intangible goals such as "manageability" in an objective function. If they can't be represented quantitatively in the objective function, it may be possible to restrict the space of candidate solutions to only those that fit the intangible criteria. Another possibility is to present a family of possible solutions to the user, who can then choose one based on the intangible goals.

## 4 When should I use optimization?

Four criteria must be met for a math programming or constraint solver to be useful. These criteria are: desire for a better solution than an ad hoc heuristic can provide, enough knowledge of the decisions to be made to express them formally, accurate and available input data so that the solver can compare alternative solutions, and sufficient time to run the solver. If only the latter two are met, then meta-heuristics may be appropriate. Otherwise, an ad hoc heuristic may be the only choice. If the answers to all four questions are all yes, then using optimization is the best bet.

**Does this problem need an optimal solution?** The stronger the desire to find the best solution, the more worthwhile it is to employ math programming or constraint programming techniques. These approaches can find feasible or, in the case of math programming, optimal solutions. Meta-heuristics and domain-specific heuristics don't provide any optimality guarantees.

**Can the decisions and constraints be modeled formally?** Math programming is appropriate if the system constraints can be modeled as sets of inequalities. If constraints can be specified only in Boolean terms, then constraint programming is a better choice. If system behavior can only be understood through simulation or black-box measurement, perhaps because of complex interactions between components, then a meta-heuristic or domain-specific heuristic is most appropriate.

**Is enough input data available? Is it accurate enough?** For math programming and constraint programming, complete input data must be available to evaluate all possible alternatives; unavailable data must be estimated with reasonable accuracy. Meta-heuristics may be able to get by with partial input information, because they can incorporate new or changed data at each step of the search space exploration. If input data is arriving continuously or can't be accurately measured or modeled, then an ad hoc heuristic is easier to use.

**Is there enough time to compute an optimal answer?** For optimization to be effective, the time to make

a decision should be shorter than the time frame for revisiting the decision with new data. Storage-related and wide area distributed run-time management decisions require answers in under a second, and MIPs can sometimes provide answers in under a second, depending on the size of the problem and the set of constraints. Configuration and capacity planning decisions that will take hours to days to implement can tolerate much longer decision-making latencies, from minutes to hours. Current commercial MP solvers, such as ILOG's CPLEX solver [13], can solve LP problems with hundreds of thousands of variables in minutes.

Applying formal optimization techniques may not be worthwhile in all circumstances. Other approaches may be more effective if: 1) the decision has only a minor impact on the quality (e.g., the performance, availability, power or manageability) of the overall solution; 2) it's easy to enumerate and evaluate all of the alternatives; 3) the alternatives are roughly equivalent in cost and benefit; 4) a formal technique won't provide the answer quickly enough to be useful; 5) the solution quality or inputs are hard to quantify; or 6) it's easy to change the decision if the result is unsatisfactory.

## 5 Where and how to use optimization?

We believe that many systems decision-making problems should be solved by standard optimization techniques. These problems are complex (and thus not easily solvable by ad hoc methods), the solutions have long-lasting impact (thus permitting longer solution times and requiring good solutions), and in many cases, the system parameters can be measured to provide accurate inputs. We summarize three areas where these techniques have been successfully applied, provide references to additional example success stories, and enumerate several classes of problems where optimization will be useful in the future.

### 5.1 Data recovery scheduling

We have addressed the question of scheduling recovery operations in a dependable storage system after a failure using MIP, genetic algorithm and domain-specific heuristic formulations [17]. Dependable storage systems protect application data by making copies through backup, snapshot and replication techniques. After a failure, applications must decide which copy to use for recovery. Some alternatives (e.g., restoring from a backup) provide fast recovery with non-trivial loss of recent updates, while others (e.g., restoring from a remote replica across a low-bandwidth network) provide minimal data loss at a potentially higher recovery time.

Applications incur financial penalties due to downtime, recent data loss and vulnerability to subsequent failures. Our objective is to minimize these penalties. The inputs to the problem are a set of penalty rates (e.g., dollars per hour for outages) for each application, device resource capabilities, and a recovery graph describing alternate recovery paths for each workload, including their operations, resource requirements and precedence relationships. The techniques choose a recovery path for each workload and determine a schedule for the recovery operations. Constraints govern the choices that can be made: for each application, only a single recovery path can be chosen, and the chosen schedule must satisfy the precedence constraints specified in the input recovery graph. Constraints also govern resource usage: the sum of all resource demands for a given device must not exceed the capabilities for that device.

We began by formulating a MIP, which we solved using ILOG's CPLEX solver [13]. However, we found that the MIP implementation had limited scalability. Even so, the MIP formulation gave us greater insight into the recovery scheduling problem, which we applied to the design of the GA and the domain-specific heuristic. We also used the MIP to establish the optimal solution for small problem sizes. We were even able to define larger problems based on the smaller ones, where we could extrapolate the optimal solution for the larger problem size. We compared the solutions provided by the other techniques against the MIP's optimal solution.

### 5.2 Publish-subscribe system

Corona [23] is a publish-subscribe system that provides asynchronous update notifications to its subscribers. Users register URLs they're interested in, and the system asynchronously sends them updates about changes posted to the URL. Changes are detected through cooperative polling by multiple nodes that periodically check the same URL and share any detected updates. Using more nodes for a URL improves update performance, but increases network load. The precise tradeoff between performance and load depends on several factors, including the number of clients requesting a URL, the content size, the update frequency, etc. Corona resolves the tradeoff by treating the number of nodes per URL as an optimization problem. The authors define several different objectives: optimizing performance while limiting load; minimizing load while bounding update delay; and several other performance metrics that depend on both the update rate and update delay per URL. The resulting non-linear optimization problem is solved quickly through their decentralized Honeycomb optimizer. A distinct advantage of this approach is that all the optimization objectives can be achieved through a common technique, as opposed to an ad hoc approach that would have required a separate heuristic for each case.

### 5.3 Web cache management

Web caches reduce network traffic and downloading latency, and can affect the distribution of Web traffic over the network through cost-aware caching. Web cache replacement policies choose which documents to evict when the cache is full, and this decision problem can be addressed through an explicit objective function. For example, Cao and Irani use objective functions based on combinations of temporal locality, document size, and network latency [5]. Kelly *et al.* propose a cache replacement algorithm that allows users to define the value of cache hits and that strives to maximize aggregate user value [19]. Both approaches can postpone the definition of the objective function until run-time, rather than specifying it at design time. Allowing users or run-time conditions to define the objective has familiar systems analogs, such as the `qsort()` function in the C library, which accepts an arbitrary client-supplied comparison function. These precedents show that adopting an OR-style optimization approach doesn't require us to hard-wire an objective function into our optimization designs; the objective function can instead be a placeholder, to be supplied by users.

### 5.4 Decision problems ripe for optimization

Many systems decision-making problems beg the use of optimization. Here we outline several such areas, providing references to published work that applies optimization and articulating specific open questions.

**Resource provisioning**: Numerous issues arise in provisioning server, network and storage resources to meet service level objectives [2, 6, 7, 16, 22, 27, 28]. For example, how many servers, network links, and bytes of storage are needed for competing workloads to meet their performance goals? How much redundancy (and what kind) is needed to guarantee the desired levels of reliability and availability? How many devices can be turned on, while still meeting power and cooling budgets? If all machines are not in use, which ones should be turned off? When a cooling unit fails, which machines should be turned off so that the current workload is least impacted, but the room doesn't overheat?

**Resource allocation and scheduling**: Which servers and storage devices should be assigned to which workloads and for how long, to meet performance [31] or availability [15] goals? Other goals may include maximizing customer revenue, minimizing energy [30], or meeting scheduling deadlines [3]. In a sensor network [20], which sensors should be powered off, and how should messages be routed to minimize energy consumption?

**System administration and management**: Many questions arise in managing system administration changes [18], migrating data [10], and setting application configuration parameters [8]. For instance, when should servers be upgraded to minimize application performance impact? How should data be migrated to newer storage, given bandwidth and ordering constraints?

**Application and system design**: Interesting questions emerge in contexts such as cache management for distributed data [5, 9, 14], distributed data replication strategies [26], determining checkpoint intervals for long-running computations [4], and database design [1]. For instance, which data should be replicated in web servers or distributed hash tables (DHTs) to minimize access time, minimize write time, or meet reliability guarantees? What is the right tradeoff between storing intermediate results and repeating computations after a failure? Which indexes and materialized views will minimize query execution time for a given query workload?

Although initial work has been done in these areas, many opportunities remain. As systems grow increasingly complex, we expect that the list will grow.

## 6 Conclusions

As system complexity increases, the number of decisions to be made, as well as the number of potential choices, increases. The key to solving these problems is to thoroughly understand the questions they ask – what should be decided, what solutions are reasonable, how to compare the alternatives, and what's important for picking the most appropriate solution. By formally formulating decision problems, the researcher gains greater insight into the problem's tradeoffs, regardless of how the problem is finally solved.

Systems researchers shouldn't settle for less than the best answers to decision-making questions. We should apply the principled approach of operations research techniques like math programming, constraint programming and meta-heuristics to obtain the best solutions. Now that we've described when these techniques are most useful, we hope you'll consider using them for the decision-making problems you face.

## References


[1] S. Agrawal et al. Automated selection of materialized views and indexes for SQL databases. *Proc. of VLDB*, pp. 496–505, Sept. 2000.

[2] G. Alvarez et al. Minerva: an automated resource provisioning tool for large-scale storage systems. *ACM TOCS*, 19(4):483–518, Nov. 2001.

[3] E. Anderson et al. Value-maximizing deadline scheduling and its application to animation rendering. *Proc. of SPAA*, pp. 299–308, July 2005.

[4] J. Bent et al. Explicit control in a batch-aware distributed file system. *Proc. of NSDI*, pp. 365–378, Mar. 2004.

[5] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. *Proc. of USITS*, pp. 193–206, Dec. 1997.

[6] J. Chase et al. Managing energy and server resources in hosting centers. *Proc. of SOSP*, pp. 103–116, Oct. 2001.

[7] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. *Proc. of Supercomputing*, pp. 1–11, Nov. 2002.

[8] Y. Diao et al. Generic, on-line optimization of multiple configuration parameters with application to a database server. *Proc. of DSOM*, pp. 3–15, Oct. 2003.

[9] L. W. Dowdy and D. V. Foster. Comparative models of the file assignment problem. *ACM Computing Surveys*, 14(2):287–313, June 1982.

[10] J. Hall et al. On algorithms for efficient data migration. *Proc. of SODA*, pp. 620–629, Jan. 2001.

[11] F. Hillier and G. Lieberman. *Introduction to operations research*. McGraw Hill, 7th edition, 2001.

[12] J. Hooker. *Logic based methods for optimization*. John Wiley and Sons, 2000.

[13] ILOG, Inc. *CPLEX 8.0 User's Manual*, July 2002. Available from http://www.ilog.com.

[14] S. Jamin et al. Constrained mirror placement on the Internet. *Proc. of INFOCOM*, pp. 31–41, Apr. 2001.

[15] J. Janakiraman, J. R. Santos, and Y. Turner. Automated system design for availability. *Proc. of DSN*, pp. 411–420, June 2004.

[16] K. Keeton et al. Designing for disasters. *Proc. of FAST*, pp. 59–72, Mar. 2004.

[17] K. Keeton et al. On the road to recovery: restoring data after disasters. *Proc. of EuroSys*, pp. 235–248, Apr. 2006.

[18] A. Keller et al. The CHAMPS system: Change management with planning and scheduling. *Proc. of NOMS*, Apr. 2004.

[19] T. Kelly et al. Biased replacement policies for web caches: Differential quality-of-service and aggregate user value. *Proc. of 4th Web Caching Workshop*, March-April 1999.

[20] G. Mainland et al. Decentralized, adaptive resource allocation for sensor networks. *Proc. of NSDI*, May 2005.

[21] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer-Verlag, 3rd edition, 1996.

[22] J. Moore et al. Making scheduling "cool": Temperature-aware workload placement in data centers. *Proc. USENIX Technical Conf.*, pp. 61–75, Apr. 2005.

[23] V. Ramasubramanian et al. Corona: A high performance publish-subscribe system for the world wide web. *Proc. of NSDI*, pp. 15–28, May 2006.

[24] J. Rolia, A. Andrzejak, and M. Arlitt. Application placement in resource utilities. *Proc. of DSOM*, Oct. 2002.

[25] A. Sahai et al. Automated policy-based resource construction in utility computing environments. *Proc. of NOMS*, Apr. 2004.

[26] E. G. Sirer. Heuristics considered harmful: Using mathematical optimization for resource management in distributed systems. *IEEE Intelligent Systems*, pp. 55–57, Apr. 2006.

[27] S. Uttamchandani et al. Chameleon: A self-evolving, fully-adaptive resource arbitrator for storage systems. *Proc. of USENIX Technical Conf.*, pp. 75–88, Apr. 2005.

[28] J. Ward et al. Appia: automatic storage area network design. *Proc. of FAST*, pp. 203–217, Jan. 2002.

[29] L. Wolsey. *Integer programming*. John Wiley and Sons, 1998.

[30] Q. Zhu et al. Hibernator: Helping disk array sleep through the winter. *Proc. of SOSP*, pp. 177–190, Oct. 2005.

[31] X. Zhu et al. Resource assignment for large scale computing utilities using mathematical programming. Technical Report HPL-2003-243R1, Hewlett-Packard Labs, Feb. 2004.