Improving Dorm Room Assignments Using Simulated Annealing

by

Elena Settanni

B.A., Mathematics, University of Florida, 1977
M.B.A., Business Administration, Pepperdine University, 1987

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Science

The University of New Mexico

Albuquerque, New Mexico

December, 2000

Dedication

To my husband for his support, encouragement and questions

Acknowledgments

Both as a computer science student and UNM Business Services staff, dorm assignments has been an interesting and challenging problem for me. I am pleased to have had the opportunity to work on it.

Thanks to the support of the following people, the resulting software is now used at UNM:

My principal end user, Dianne Ranville, UNM Housing Reservations Supervisor, whose willingness to try something new is largely responsible for the profound satisfaction of being in production. Conversations with Sally Jaramillo, Housing Reservations Accounting Clerk, were most helpful in learning specific details about the existing system and procedures. U. Stephen Allison, Manager of Business Operations, has balanced the need for new answers and reliable product support.

Discussions with the following professors at UNM have offered many insights: The awareness of 'MCMC' and related scheduling problems from my advisor, Barak Pearlmutter; the significance of 'listening to the zeros' from Jim Ellison of the Math Department; the confidence programming in C after Barney Maccabe's Compiler Construction class; and the hacks in response to Dave Ackley's incisive questions.

Thanks to Dr. Sheldon Ackley for his helpful comments as an early reader, and my committee members, Professors Barak Pearlmutter, Arthur B. Maccabe, and Charles P. Crowley, for their time reviewing this manuscript.

Improving Dorm Room Assignments Using Simulated Annealing

by

Elena Settanni

ABSTRACT OF THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Science

The University of New Mexico

Albuquerque, New Mexico

December, 2000

Improving Dorm Room Assignments Using Simulated Annealing

by

Elena Settanni

B.A., Mathematics, University of Florida, 1977

M.B.A., Business Administration, Pepperdine University, 1987

M.S., Computer Science, University of New Mexico, 2000

Abstract

Assigning dorm rooms to 2,000 students with complex and interdependent preferences is a difficult problem. User dissatisfaction with an existing proprietary commercial matching system afforded an opportunity to consider fresh approaches. This project, begun August 1998, has focussed on using simulated annealing to optimize dorm room assignments. The project goals are to improve usability and resource utilization of the assignment process, offering better, faster matches for happier students and staff. Having completed its third semester of production use at UNM, results show that the Dorm Assignment Optimizer (DAO) works well producing assignments far superior to the previous system, in a fraction of the time.

Contents

List of Figures **xii**

List of Tables **xiv**

Glossary **xv**

1 Choices, Chances and Results **1**

1.1 Choices: The Dorm Assignment Problem 2

1.1.1 Background 2

1.1.2 The Environment 4

1.2 Chances: The Simulated Annealing Solution 6

1.2.1 Introduction to Annealing and Simulated Annealing 7

1.2.2 Optimizing Dorm Assignments 11

1.3 Results: Fall 1998 – Fall 2000 13

2 Choices: The Dorm Assignment Problem **14**

2.1 The Objective Function 14

2.1.1 Neighborhood Landscapes 15

2.1.2 Constraint Costs 20

2.1.3 Modifying the Objective Function 22

2.2 Choices in Policy 24

2.2.1 Inflationary Costs 25

2.2.2 Multiple Assignment Runs 26

2.3 Choices in the Data 27

3 Chances: The Simulated Annealing Solution **30**

3.1 Simulated Annealing Schedules 30

3.2 Dorm Assignments: Five Cooling Schedules 33

3.3 Guaranteed Local Minimum 38

3.4 Discussion: Adaptive Schedules 38

3.5 Handling Ties 40

4 Results: Fall 1998 – Fall 2000 **42**

4.1 The Artifact 42

4.1.1 Choosing Constraint Coefficients 43

4.1.2 Between Two Systems 44

4.1.3 Batch Management 45

4.1.4 Checkpoint Restart 45

4.1.5 Security Issues 46

4.1.6 Efficiency Considerations 46

4.2 Case Studies 48

4.2.1 Fall 1998, versions v0.01 - v0.39 49

4.2.2 Fall 1999, versions v0.40 - v0.85 50

4.2.3 Spring 2000, versions v0.90 - v1.03 51

4.2.4 Fall 2000, versions v1.04 - v1.15 52

4.3 Experimental Results 55

4.3.1 Characteristics of the Data 56

4.3.2 Methodology 57

4.3.3 Random Optimization 59

4.3.4 Taking Uphill Moves 59

4.3.5 Importance of Lateral Moves on a Mesa 62

4.3.6 Varying Probabilities for Better Moves 63

4.3.7 Reaching a Local Minimum 64

4.3.8 Comparing the Schemes: Scores and Attempts 66

5 Conclusions **69**

Appendices **73**

A HMS Rules for Assignment with the DAO **74**

B The DAO Objective Function **75**

References **80**

List of Figures

1.1 Boltzmann Probability Factor (the heart of annealing) 9

1.2 System Overview 12

2.1 Neighborhood Landscape at a Random Point 15

2.2 Neighborhood Landscape of the Most-senior and Least-senior Students at a Random Point 16

2.3 Neighborhood Landscape at a Single (local optimum) Point 17

2.4 Neighborhood Landscape of the Most-senior and Least-senior Students at an Optimized Point 18

2.5 Neighborhood Landscape of the Least-senior and First-new Students at an Optimized Point 19

3.1 Temperature vs Attempts by Scheme — Samples from Fall 2000 . . 37

3.2 Temperature vs Attempts by Scheme — Samples from Fall 1999 . . 37

3.3 Temperature vs Attempts by Scheme — Samples from Fall 1998 . . 37

4.1 Dorm_View Screenshot 54

4.2 Mean Rank By Method — Fall 2000 60

4.3 Mean Rank By Method — Fall 1999 60

4.4 Mean Rank By Method — Fall 1998 60

4.5 Average Score By Method — Fall 2000 61

4.6 Average Score By Method — Fall 1999 61

4.7 Average Score By Method — Fall 1998 61

4.8 Score vs Total Attempts by Scheme — Fall 2000 67

4.9 Score vs Total Attempts by Scheme — Fall 1999 67

4.10 Score vs Total Attempts by Scheme — Fall 1998 67

List of Tables

2.1 Constraint Classifications 20

3.1 Percentage Results from Sample and Actual Performances 36

3.2 Probability of Accepting Moves by Methods 40

4.1 DAO Process Flow 47

4.2 Key to Methods 55

4.3 Statistics (in millions) from Fall 2000 Tests 58

4.4 Statistics (in millions) from Fall 1999 Tests 58

4.5 Statistics (in millions) from Fall 1998 Tests 58

4.6 Lateral Test Statistics (in millions) 63

Glossary

Accept	A move that changes the current state.
Annealing	The thermal process for obtaining low-energy states of a solid [AKvL97].
Attempt	Repeated occurrence of the event under study, also termed trials [DM84] (i.e. A bed and student chosen to switch in the DAO).
Batch	Method of managing DAO runs by assigning unique *batch* numbers.
Coefficient	The value associated with a constraint, such that, the more important the constraint, the greater its value if violated.
Constraint	A rule to deter mismatches from occurring by limiting the desireable possibilities.
Cost	The value of the objective function. Same as 'grade'.
Cycle	A series of states that repeat over and over.
DAO	Dorm Assignment Optimizer.
Downhill	In a minimization problem, when the change in cost of an accepted move is negative, the overall score improves.
Equilibrium	Indifference to previous states in a process, such that an independent sample can be drawn.

Glossary

Grade — The total costs of mismatches in a given state, according to the DAO objective function. Same as 'score' and 'cost'.

Hillclimbing — Improving moves are always accepted, deteriorating moves are never accepted, and lateral moves may or may not be accepted.

Landscape — The solution space, where the *height* is the overall cost at a single data point or state.

Lateral — A move that changes the state, yet leaves the score unchanged.

Markov chain — Named after the Russian mathematician A.A. Markov who formalized the theory concerning events whose current condition depends solely on their condition one period before, Markov chains are special Markov processes that also assume finite states exist, constant transition probabilities exist, and equal time periods occur [DM84].

Monte Carlo sampling — First employed by J. von Neumann and other researchers in conjunction with military efforts during World War II, it is a technique used to select sample values randomly from a probability distribution for subsequent use in a simulation process [DM84].

Move — Rearrangements of the elements in a configuration (i.e. switching a student and bed occupant in the DAO).

Neighborhood — All the possible states that could result after a single move.

Nobetter — The number of rounds to pass without seeing improvement in DAO.

Objective function — The sum of constraints, each with assigned costs, resulting in a total score.

Phase	The number of moves that must be attempted or accepted before there is a change in temperature. Also known as the length of the Markov chain in the simulated annealing algorithm.
Random walk	An example of a Markov chain, it is not a very fast way to get anywhere: on a coin flip, take one step to the right on heads, to the left on tails, after 25 steps you'll be about five steps from where you started [Sha]. The chance of returning to the start is sharply reduced as the problem dimension is increased.
Round	Based on a single cooling schedule, a configuration, or state, produced by the DAO.
Scheme	A cooling schedule to govern the convergence of the simulated annealing algorithm.
Score	Same as 'grade'.
State	A single data point (i.e. the assignment of each student to a bed).
Temperature	The control parameter in simulated annealing representing the willingness to accept a bad move. When the value is large the willingness is greater.
Uphill	As a minimization problem, when the change in cost of an accepted move is positive, the overall score worsens.
Virtual bed	In the DAO, a waiting list of beds not existing in physical reality.

Chapter 1

Choices, Chances and Results

The Dorm Assignment Optimizer (DAO) project focuses on using simulated annealing to optimize dorm room assignments. Juxtaposing the choices and chances underlying the process of assigning dorm rooms, in this thesis, we present the essence of the problem, our solution, and results of this two year project.

Assigning dorm rooms can be viewed as a collection of choices made by the students and the Housing Reservations staff, and a series of chances taken by our algorithm to produce a guaranteed local optimal assignment of students to beds. Choices and chances is a useful distinction even though there will be some chance in the problem and choice in the solution.

DAO production results have demonstrated superior performance in terms of student placement and use of resources during the last three semesters of use at UNM, over the previously used vendor-supplied mechanism.

1.1 Choices: The Dorm Assignment Problem

The choices that describe the dorm room assignment problem are a combination of student requests and management decisions. Students specify their preferences for halls, room type, non-smoking environments, music tastes, study habits, and requests for up to five specific roommates on their application forms. Residency seniority and prompt application submission are used to prioritize these requests — some chance is involved in the order applications are received at the Housing office.

The Housing Reservations staff is responsible for assigning students to their beds each semester. As proxy for the students, the Housing staff strives to minimize complaints and maximize those who choose to reside on-campus. This one-level of indirection provides an opportunity to gain a global perspective and an overall better final arrangement.

In addition to previously established procedures, Housing Management makes critical choices regarding the placement of co-eds by designating room attributes such as gender and smoking, and prioritizing student preferences.

These human choices define the problem we describe here in further detail.

1.1.1 Background

Originally, the manual effort of assigning beds took over two weeks and many late hours. Seven years ago, UNM began using HMS, a proprietary, commercial housing management system[1] that included an assignment module. Based on the FoxPro database system, HMS is a full-function campus housing management system that catalogs every room and occupant, assigns beds, applies charges to student accounts, offers work order management capabilities, and interfaces with their dining module

[1] http://www.cbord.com/

(Figure 1.2 delineated by the dotted box on page 12). It resides on a Novell file server for multi-user accessibility. The automated bed assignment module is rule-based with scores for successful matches and additional logic to process mutual roommate requests.

While an improvement over doing it by hand, the HMS assignment process over the network, took as long as 48 hours to process 700 returning students. Due to the lengthy processing times, Dianne Ranville, the Housing Reservations Supervisor, found it necessary to use a variety of workarounds to complete her task. For example, she would run new and returning student assignments separately, and the process had to be run over weekends to avoid blocking other system users.

Ranville found the rules used to make the assignments in HMS often complex and unintuitive. Furthermore, with turnover of the support staff at CBORD, the answers to her questions changed, and as a result, the 'current' set of rules never worked exactly as desired. For example, student roommate requests would override all the other preferences no matter what she did, often resulting in new students incorrectly assigned to the better dorms, no regard for roommates requested by students returning to their same room, and hall requests demoted in importance.[2] In addition, HMS could not handle some specifics of the UNM dorm system, such as assigning roommates in a co-ed dorm, popular among residents because of its adjoining rooms. Handling such *suite situations* was one of Ranville's specific requests for an improved system.

HMS scores for the user-provided rules are restricted to powers of two, which tended to stifle Ranville from changing their values, so instead she would deactivate them. A rule intended to put students of closer ages together was very hard to understand (shown on page 43), and was consequently deactivated.

[2]An as-yet-unreleased rewrite of HMS reportedly allows the user to adjust the importance of the roommate rule.

Although HMS is proprietary, based on its behavior, its assignment system appears to be strictly sequential, considering students in seniority and then reservation order, and assigning them to the best remaining bed. As rooms fill up, the process takes less time, according to Ranville. Also, as fewer rooms remain available, she has to inactivate rules to allow the assignment process to complete.

Thus, user dissatisfaction with the existing proprietary matching system afforded an opportunity to consider fresh approaches to the problem resulting in the DAO.

1.1.2 The Environment

In the UNM dorm system, there are over 2,000 beds and typically nearly 2,000 students that must be matched. Housing Management policy shapes the environment of the dorm system. These policies cover early student notification, first-come first-placed, preference priority, as well as room attributes, such as gender, non-smoking, and room type.

Early Student Notification To reduce anxiety and minimize phone call inquiries, students are notified of their fall semester hall assignment by midsummer. After the first cutoff in June, assignments are made as applications arrive and are entered into HMS. As students cancel and free up the more desirable spaces, Ranville blocks these prize locations from the automatic assignment process until she can manually move the more senior students into such better beds. Students are kept apprised of improvements to their assignment due to cancellations thoughout the summer and up until the day they move in. This time-consuming 'upgrading' task, requiring lots of intuition and good will on Ranville's part, attempts to place students into their preferred hall. Further consequences of this policy are presented in Section 2.2.2.

First-come, First-placed Deeply entrenched in the marketing of on-campus housing is the first-come, first-placed basis for assigning rooms. Besides being intuitively easy to understand and explain to students and parents, it encourages students to get their paperwork in early. Although originally intended for office use only, these 'lottery numbers'[3] are now an open secret, and students become very upset upon learning that someone behind them in line received their requested dorm but they didn't! Students who have higher seniority, that is, more consecutive semesters in the dorms, also get a significant advantage. Student priority order is discussed further in Section 2.2.1.

Preference Priority Each year students are surveyed to learn their feelings regarding the importance of hall choice over roommate requests. The hall usually wins over a friend, and Ranville adjusts the rules attempting to reflect this perception the following year. It becomes the delayed broad brush-stroke with the highest penalty. Many fine-brush strokes are also involved to minimize age differences, separate students with conflicting music tastes and study habits, with smaller impact on the overall score. The challenge of setting these values is discussed in Section 4.1.1.

Gender Designating rooms to specific genders involves subtle issues and so are not done automatically. Room gender designations are arranged to avoid conflicts and preserve fairness. For example, a dorm arranged as three floor buildings with two apartments per floor accessed by outdoor stairs is given a 'checkerboard' gender designation to prevent one gender from taking over a building or blocking passage to the top floors.[4] Such room attributes outweigh student seniority considerations.

[3]Known internally as *lottery number*, it is the number stamped on the student's application when their deposit is paid. New and returning students have separate accumulators — lower is better.

[4]Allison, S. Personal communication, Spring 1999.

Smoking Trends With the increasing trend of students not smoking, and objecting to traces of tobacco odor in their room, entire buildings have been designated non-smoking, leaving fewer rooms where the smoking attribute is determined by its occupants. Students can stress their objections to smoke on their application forms, however, this information has yet to be leveraged.

Room Type A distinction between room capacity and room type exists. For example, depending upon enrollment levels, bunk beds may temporarily be installed in some rooms making three to a double room possible, or a bed may be removed creating a 'doubles-as-singles' option.

Interdependent constraints — hall preferences, roommate requests, room attributes, adjoining suites, cancellations, turn-in order, multiple runs and upgrades — makes automatic dorm room assignments a difficult global optimization problem. The deterministic approach apparently used by HMS suffers from myopia — there may be thirty wonderful choices for the first student, but HMS's decision ignores the effect on the hundreth student who has fewer possible good alternatives, and so forth down the line. Simulated annealing's holistic approach seemed better-suited to the task of discovering an overall better set of assignments without hurting those who benefit from seniority. Initially, our goal was to see if that was true.

1.2 Chances: The Simulated Annealing Solution

Starting with a seed of chance, DAO considers millions of random swaps. Using a simulated annealing algorithm, better, worse and lateral moves are considered to avoid possible entrapments and uncover better overall configurations. The DAO usually produces a different result each time it is run — therefore, we say, the solution is based on chance.

1.2.1 Introduction to Annealing and Simulated Annealing

Physical annealing is a three stage process that has been known and used for shaping metals since about 5000 B.C. [Enc00b]. Any annealing process consists of three stages: heating to the desired temperature, holding at that temperature, and cooling, usually to room temperature [Ame]. Glass, crystals [Enc00a], and other materials are also annealed to render them less brittle and more workable [Col94].

Sequences of times and temperatures, called 'annealing' or 'cooling' schedules, are critical in two ways: If the rate of the temperature change between the outside and inside of the piece is too great, temperature gradients and internal stresses may be induced that may lead to warping or cracking. Also, the actual annealing time must be long enough to allow for any necessary transformations to take place [Ame].

Using *Monte Carlo sampling techniques*, the physical annealing process has been modeled successfully by computer simulation methods. The analogy between a physical many-particle system and a combinatorial optimization problem is based on the following equivalences [AKvL97]:

- Solutions in a combinatorial optimization problem are equivalent to states of a physical system.
- The cost of a solution is equivalent to the energy of a state.
- A control parameter plays the role of *temperature*, such that: At large values, changes in energy are accepted; as it is reduced, only decreases or smaller increases are accepted, and as it approaches zero, no increases are accepted at all. Furthermore, there is no limitation on the acceptable size of an energy increase, a characteristic feature of simulated annealing.

The Metropolis algorithm for Monte Carlo is the grandfather of simulated annealing algorithms, proposed in 1953 in [MRR+53]. Its significance is highlighted by its selection as the top algorithm with the greatest influence on the development of science and engineering in the 20^{th} Century for offering "an efficient way to stumble towards answers to problems too complicated to solve exactly" [DS00].

Kirkpatrick and Gelatt first tested simulated annealing on the travelling salesman problem, finding locally optimal solutions for N up to 6000 sites [KGV83]. At that time the exact solution had been obtained for 318 sites. Annealing was also used to optimize the design of complex integrated circuits by arranging hundreds of thousands of circuit elements to minimize chip space requirements and to reduce interference among their connecting wires [KGV83].

For certain NP-hard problems, where poor local extrema abound but are not far from good local extrema, the simulated annealing technique outperforms straightforward iterative improvement, at the cost of much longer running times [MS91].

It has been applied to many problems, ranging from the practical to theoretical [AKvL97]. A related, yet more complex, example is academic course scheduling at Syracuse University [EFC98]. Other NP applications include signal and image processing, task allocation, network design, graph coloring and partitioning [JAM91, JAM89], and molecular analysis.

Four ingredients are needed to apply simulated annealing to new problems [KGV83]:

- A concise description of the system configuration;
- A random generator of moves or rearrangements of the elements;
- A quantitative objective function representing all the trade-offs;
- An annealing schedule of temperatures and length of times by which the system evolves.

Annealing generally begins by reading in an existing solution to the problem or generating a random solution. Once the starting temperature and termination criterion have been established (perhaps by examining a large number of solutions) the actual annealing begins. At each iteration, the current solution is randomly perturbed to create a new solution (this step is often referred to as a move). The change in cost ΔC from the previous to the new state is calculated, and the move is accepted with probability

$$\frac{1}{1+e^{\Delta C/T}}$$

its Boltzmann probability factor (or sigmoid), where T is temperature.

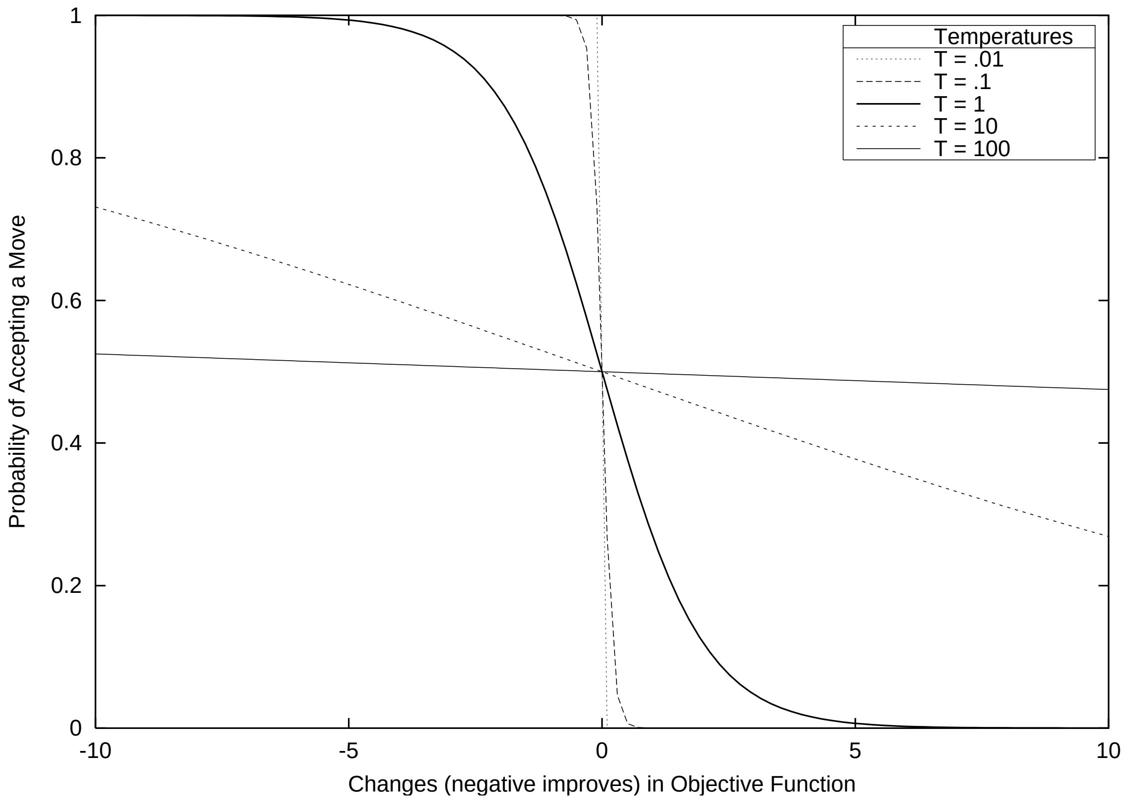

Figure 1.1: Boltzmann Probability Factor (the heart of annealing)

The probabilities for different changes in cost at different temperatures intersect at the center point (0,.5) as shown in Figure 1.1, where there's a 50–50 chance of accepting a move when no change in score occurs whatever the temperature. When the temperature is high, the solid $T = 100$ line shows there is almost always a 50–50 chance of accepting a move regardless of the cost. As the temperature decreases, the bold $T = 1$ line approaches a probability of one as the change in cost decreases and a probability of zero as the change in cost increases, showing we are very likely to take improving moves and less likely to take deteriorating moves.

In general, temperature is used to scale differences in height of the landscape. Raising the temperature flattens a rugged landscape by a greater willingness to accept worse moves. Small irregularities on a smooth landscape are accentuated by lowering the temperature and accepting mostly improving moves. Landscapes are depicted graphically in Chapter 2.

The inherent problem with only accepting better moves, *hillclimbing*, is that we might get stuck in a valley that can't be escaped by taking downhill moves. The willingness to move uphill to a worse position gives simulated annealing the advantage of possibly escaping from local minima in hope of finding a deeper valley farther away.

If the move is not accepted, the previous state is restored. After a certain number of moves, the temperature is decreased — thus decreasing the probability of accepting uphill moves. When no downhill (better) moves exist in the neighborhood, we have reached a *local minimum*. This does not imply we've reached a global minimum solution, however.

Finally, the algorithm terminates when some specified stopping criterion is met — e.g., when no improvement has been found for some number of moves.

One challenge with annealing is the determination of the parameters controlling the execution of the algorithm [Lee94]. It is still an art [AKvL97].

1.2.2 Optimizing Dorm Assignments

The Dorm Assignment problem can readily be structured for simulated annealing:

- The dormitory system can be described concisely in terms of students and beds;
- The students can be rearranged in different beds randomly;
- An objective function based on student preferences can be devised;
- Through trial and error, and a bit of luck, an effective annealing schedule can be found.

The existing HMS system provides us a convenient description of the problem domain: The beds and the students (Figure 1.2). The student data is entered into the system from the application forms manually, and more recently through a trial scanning system.

In the DAO, each rearrangement attempt, or *move*, is based on switching a randomly chosen student with the occupant of a randomly chosen bed (though it may be empty).[5] Available beds are selected directly instead of first selecting a room and then a bed, eliminating any dependency upon room capacity. Virtual beds, though nonexistent in physical reality, are offered as possible temporary alternatives — amounting to a kind of waiting list.[6] Students who have already been assigned a room by the Housing Reservations staff or a previous run of the DAO, are considered unmoveable and are never selected, but their preferences continue to be considered in the placement process of their roommates.

The objective (cost) function is a sum of per-student error terms. Used to calculate the overall cost, Housing chooses the coefficients for these terms — an art

[5] Other notions of a 'move' could be defined (e.g. a group together, or splitting a suite).

[6] The number of virtual beds is a configurable option with a minimum of one room (six beds) per gender.

in itself. As the primary repository of preference and policy choices, this objective function will be the focus of Chapter 2.

To minimize the cost function, five annealing schedules: Algorithmic, table-driven, and three adaptive algorithms, each with its own criteria for reducing temperature and phase and determining completion, were devised and studied for this project. To use simulated annealing effectively to sample the search space efficiently, a good cooling schedule is crucial [EFC98] — this is the focus of Chapter 3.

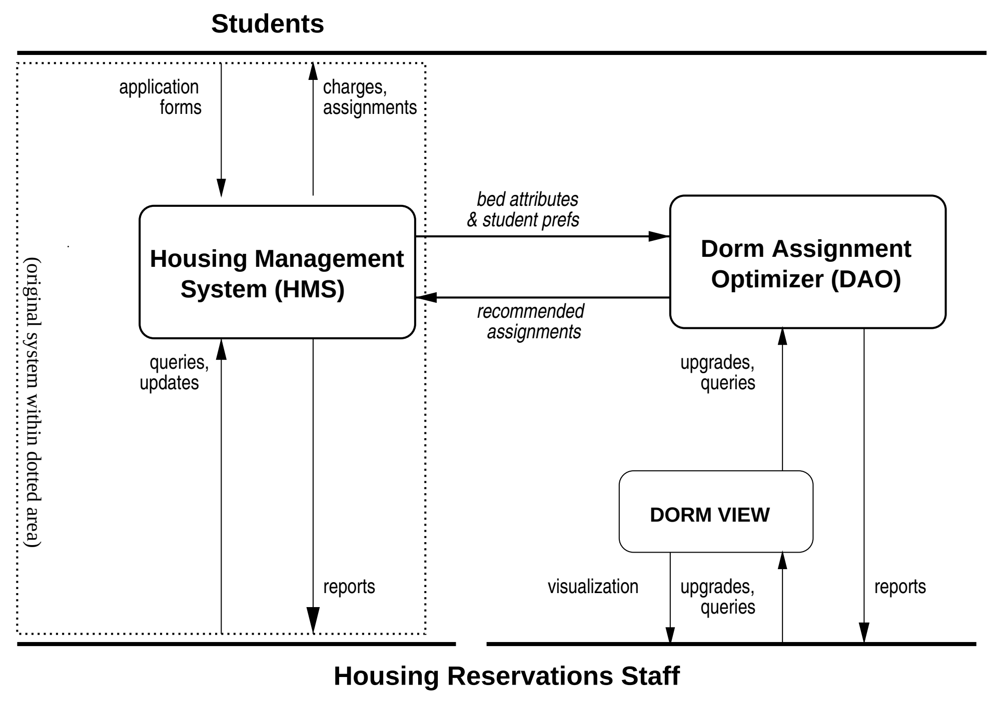

Figure 1.2: System Overview

1.3 Results: Fall 1998 – Fall 2000

The placement of students by this stochastic technique has been the alternative of choice at UNM since Fall 1999. We summarize our results here by semester:

Fall 1998 Tests comparing the DAO and HMS showed promising results.

Spring 1999 Our approach was presented to Housing Management, gaining approval for a side-by-side trial run for the upcoming fall semester.

Fall 1999 First production run was successful achieving 87.3% student satisfaction — one of three top dorm choices granted. Unfortunately for the science, the HMS comparison run was not performed due to time constraints, however, many valuable lessons were learned along the way.

Spring 2000 Enhancements were made to the DAO to accommodate aspects unique to the spring semester — the majority of the students stay in their same rooms with only a few hundred students newly assigned.

Fall 2000 The automated upgrading facility was available for the first time. Dorm_View, a graphical user interface (GUI) we developed to give the Housing Reservations user the ability to: Visualize assignment results using colored lens for different student and room attributes; customize reports based on halls displayed; and move or switch students with more complete knowledge surrounding the change, was used for the first time (Figure 4.1 on page 54).

Production and experimental uses of the DAO have provided many insights into the dorm assignment problem and the simulated annealing solution. These results are covered in Chapter 4.

Chapter 5 concludes with some ideas for future work, from the nearer-term to the speculative.

Chapter 2

Choices: The Dorm Assignment Problem

Without trying all 2,000! possible combinations of dorm room assignments, we would like to find the combination where everyone is the happiest. To do this we need a way to quantify how bad a configuration is, that is, how unhappy the students are likely to be. We define an objective function, f, to capture as much knowledge as possible regarding preferences and policy choices. Our goal is to find the set of assignments with the smallest value of f, thereby minimizing mismatches.

2.1 The Objective Function

The Objective Function for this application is all the contraints used to match students and dorm rooms. The domain is all the students in beds. A single data point, or *state*, is the assignment of each student to a bed. f maps from a state to a non-negative integer score, where a zero score (if you could get there) would be 'perfect' with no errors, and everything else is worse. Starting at some state, we try to improve it by moving to nearby states. By switching a randomly chosen student with the occupant of a randomly chosen bed, while all else stays constant, we can reach a nearby state. Each state has some four million possible neighbors, since any of almost 2,000 students can be moved to any of 2,000 beds. To better illustrate the concepts of *neighborhood* and *landscape*, we present graphs showing a single data point from both random and optimized configurations.

2.1.1 Neighborhood Landscapes

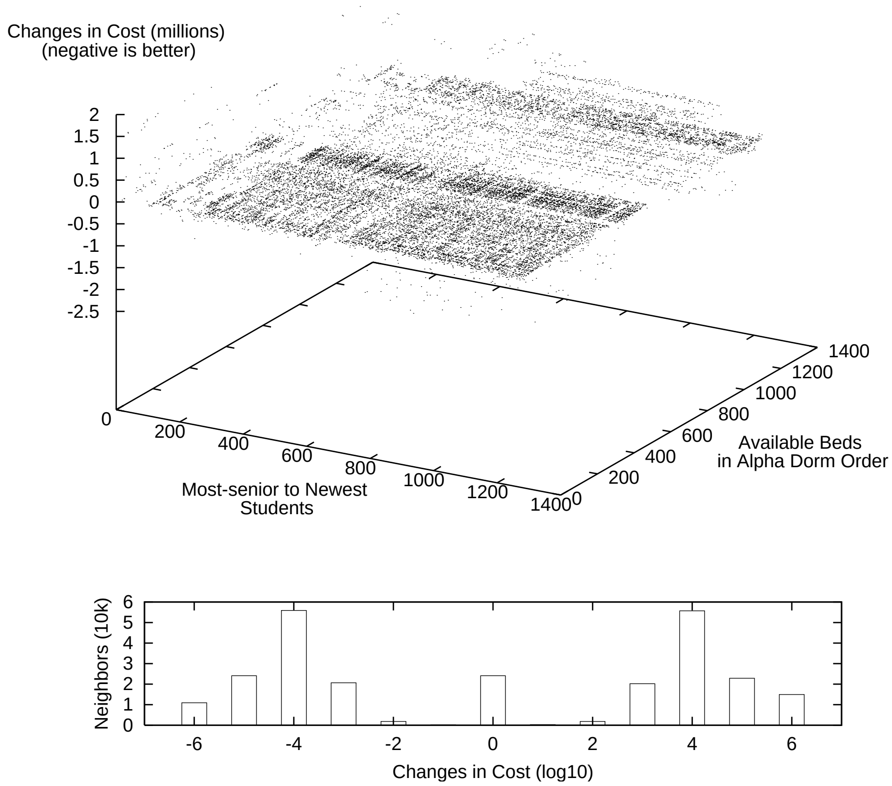

Figure 2.1: Neighborhood Landscape at a Random Point

The landscape of the neighborhood of a single random point is represented in Figure 2.1.[1] The value of the objective function was 520 million, achieving only 37% satisfaction, leaving 803 out of 1267 students nowhere they wanted to be, and only 202 students receiving a perfect score. The histogram shows the nearly symmetric distribution of the cost changes of this neighborhood. The significantly better moves are below the cloud line (negative change in cost), while those above the cloud line are worse.

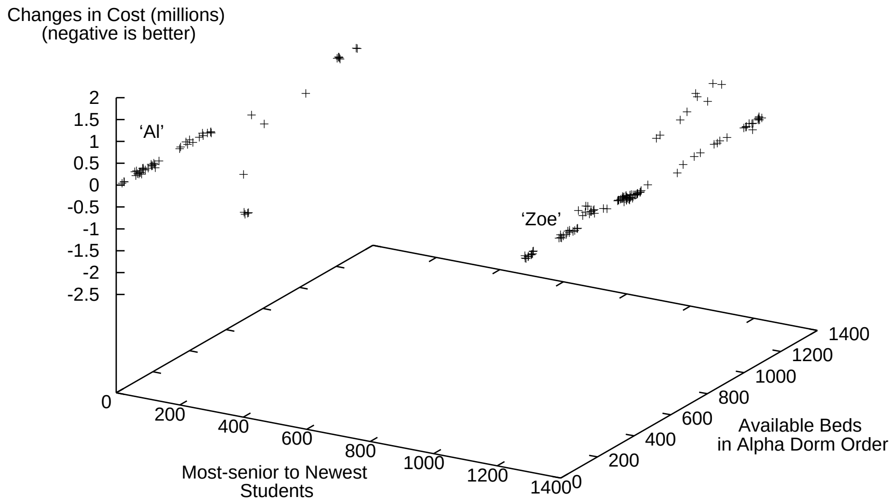

Figure 2.2: Neighborhood Landscape of the Most-senior and Least-senior Students at a Random Point

In Figure 2.2 we isolate the most senior student, who we'll call '*Al*' (at position 10), a resident for eleven consecutive semesters, and the least senior student, '*Zoë*' (at position 1267), a first time resident, at the single random data point to see how they would fare in any other available bed. As it happens, *Al* objects to smoke

[1]To reduce clutter, an arbitrary 25% of the histogram data is plotted.

and requested the apartment-style dorm for all three choices, with one roommate request, and a special request to be in the graduate area; *Zoë* is a smoker with no hall preferences or roommate requests. They were both randomly placed in virtual beds.

The best place for *Al* is the apartment-style dorm (near bed 700) which matches the special living option he requested and switches with someone who has no desire to be there. *Zoë* improves her score pretty much anywhere we place her; the worst places for her are non-smoking apartment-style rooms with the non-smoking special living option (near bed 1000).

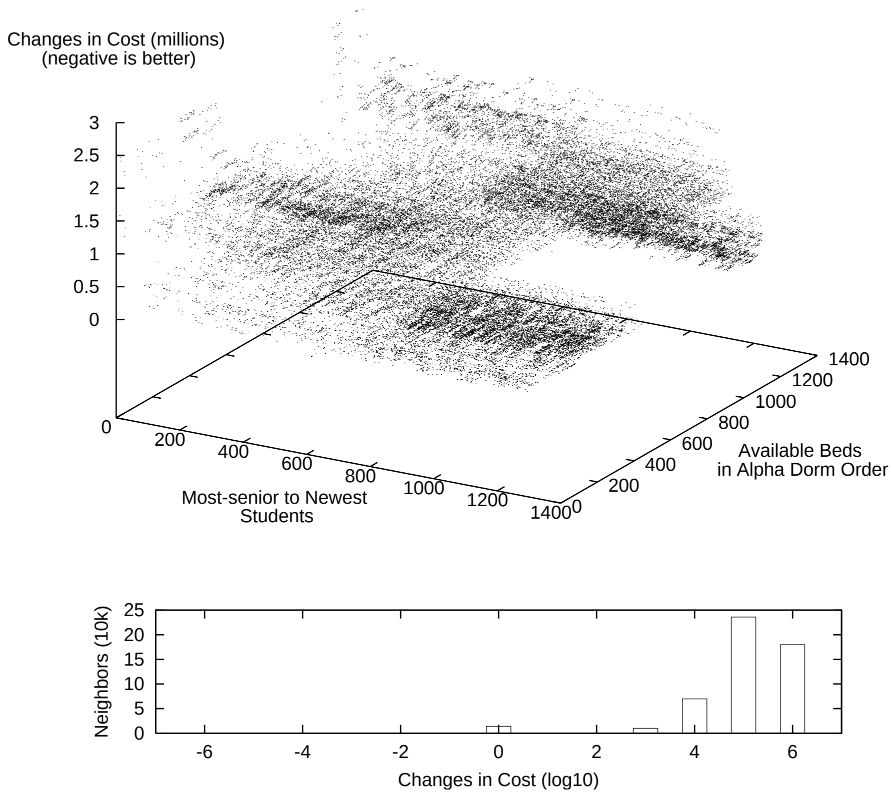

Figure 2.3: Neighborhood Landscape at a Single (local optimum) Point

Using the same constraint values and data as in the random point, after six rounds taking approximately two hours, the global score declined to 45 million, achieving 97% satisfaction with only 32 out of 1267 students nowhere they wanted to be, and 697 students receiving a perfect score. The result of this complete run, as we can see in Figure 2.3, is a local minimum — there are no better single moves (below zero) for any student.

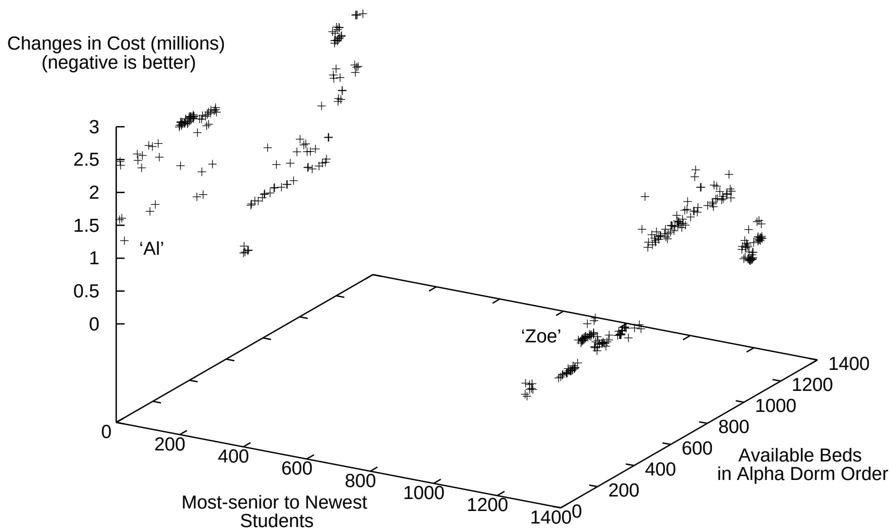

Figure 2.4: Neighborhood Landscape of the Most-senior and Least-senior Students at an Optimized Point

Two slices of the neighborhood landscape for our students, *Al* and *Zoë*, at the optimized point can be seen in Figure 2.4. *Al* got his first hall choice in the graduate area, but didn't get his roommate request because it was not a reciprocal request and the special grad area wasn't mutually specified. *Zoë*, who was so easy to please, got a perfect score. Her set of possible alternative moves exhibits the end of a phenomenon that looks like a 'hole' in the cloud.

Figure 2.5 pinpoints the start of this obvious ‘hole’ between the last second semester resident, $Mary$ (at position 718), and the first new student (at position 721) $Nick$. $Mary$ got her second hall choice, but not her requested roommate, a more senior resident who wanted and got into the apartment-style dorm. $Nick$, assigned to his first hall choice with the scholar’s wing special option and no requested roommates, has a perfect score. Attempting to move him into any of the coveted apartment-style dorms (beds 673 through 1161) severely hurts his grade with increased costs — this begins the vacant region.

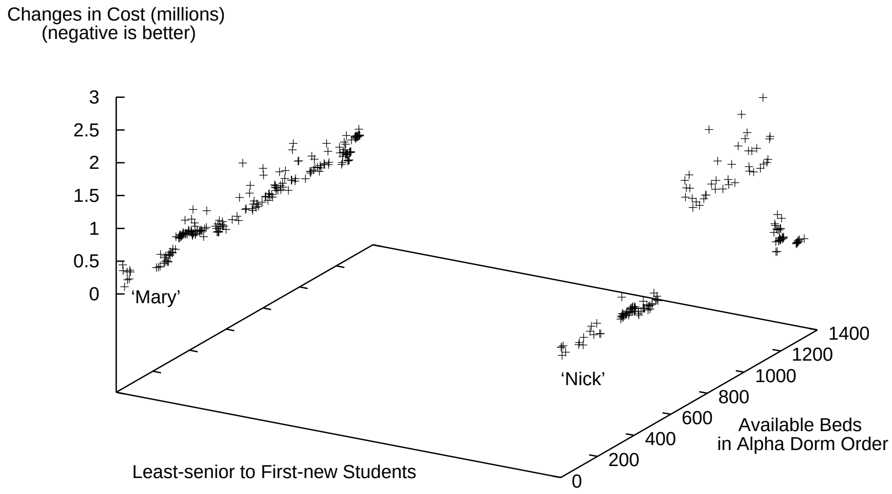

Figure 2.5: Neighborhood Landscape of the Least-senior and First-new Students at an Optimized Point

The Objective Function has terms that include dependencies *between* students, not just students and beds independently. As we’ve seen with Al’s assignment, a student requests other students as roommates, and depending upon the locations of the requested roommates there may be a ‘mismatch’ or ‘error’.

2.1.2 Constraint Costs

The basic structure of the DAO's objective function iterates over all the students and sums up their 'unhappinesses'.

There are two main types of constraints in the DAO's objective function: *Room-student* and *student-consensus* related errors (Table 2.1). Room-student constraints take student and room attributes into account. For example, a room-student mismatch occurs when a student's first choice hall does not match the hall of their room. Student-consensus constraints are concerned only with the students in the room (or suite). Student-consensus errors occur when the occupants of the room do not agree, for example, a student who prefers classical music is in the same room as someone who objects to the classics. Student-consensus constraints can also be used to determine the attribute of the room, such as non-smoking and gender[2].

ROOM-STUDENT Unweighted (RS)	STUDENT-CONSENSUS Unweighted (SC)
bed	**gender**
smoke	**smoke**
underage	**music**
different hall	**study late**
different bed	**age difference**
worse hall	**fill (partial & triple)**
better bed	**first timer**
ROOM-STUDENT Weighted (RSw)	STUDENT-CONSENSUS Weighted (SCw)
dorm preferences	**group (roommates)**

Table 2.1: Constraint Classifications

Additionally, a compiled-in table determines whether constraints are weighted by students' seniority and reservation order. Priority increases the constraint penalty if there is a problem, thereby encouraging the system to find a better solution for

[2]Gender consensus applies to undesignated male or female rooms. The gender attribute is generally predetermined at UNM (see page 4).

the individual and the whole. 'Dorm Preferences' and 'Roommate Requests' are the weighted constraints in the DAO. A closer look into weighted and unweighted constraints follows in Section 2.2.1.

The Objective Function is the sum of all per-student constraint violations, weighted and unweighted, as they apply to each student in a bed, given all the other students in their beds. Housing sets the penalties to indicate the relative importance of the errors (e.g. no bed at all is much worse than a older roommate). The grades and the error breakdowns are kept on both individual and overall bases for reporting purposes.

The Objective Function (details in Appendix B), f, is computed as:

$$f(e) = \sum_{s=1}^{m} \left(slw(s) \sum_{j \epsilon RSw} v_j RSerr_j(e,s) + \sum_{j \epsilon RS} v_j RSerr_j(e,s) \right) +$$

$$\sum_{r=1}^{n} \sum_{s=1}^{t} \left(slw(s) \sum_{j \epsilon SCw} v_j SCerr_j(e,s,r) + \sum_{j \epsilon SC} v_j SCerr_j(e,s,r) \right)$$

where e is the state, m is the number of students, n is the number of rooms, t is the number of students in a room, slw is the student's calculated seniority-lottery-weight[3], v is the user-specified value of the constraint coefficient, RSw is the set of weighted room-student constraints, RS is the set of unweighted room-student constraints, $RSerr$ is a room-student error, SCw is the set of weighted student-consensus constraints, SC is the set of unweighted student-consensus constraints, and $SCerr$ is a student-consensus error.

The overall cost (the global grade or score) for all the students, computed initially and as sanity checks, is linear in the number of students. Most of the time, however,

[3]The slw is the number of consecutive semesters as a resident in the dorms multiplied by the seniority weight, plus the lottery benefit (see page 25).

we can update it incrementally in constant time by adding the change in cost resulting from a single move to the global grade.[4] Our goal is to minimize the global grade.

Sometimes the Whole Must Suffer
A first time student for Fall 2000, seventh in line by lottery number, had three different hall choices, the second of which was the less popular all-female hall. The DAO placed her in her second choice for the overall benefit of those students who would not be happy with anything less than this student's first choice. Regardless, she was manually upgraded to her first choice because she was at the head of the line.

Seniority weights, constraint error values, and other configurable parameters assigned by Housing, are read from an options file as the program begins, and are used to compute the overall cost of the objective function. As management policy changes, these choices can be amended without recompiling the DAO. However, choosing values accurately to reflect these policies becomes the real challenge, as shown in Section 4.1.1 (page 43).

2.1.3 Modifying the Objective Function

The DAO objective function, written in C, requires a programmer and recompilation to modify it, unlike the HMS rules. However, we provide an understandable way for Housing to adjust the grading function: Seniority weights, constraint coefficients,

[4]Worthy of note is the significance of the maximum number of beds in a room (a constant) to the worst case complexity of the group, gender, and smoking student-consensus constraints across suites. For an intermediate grade calculation, the complexity in the worst case, based on *consensus_across_suites* = # *rooms_in_suite* * # *students_in_room* * # *roommate_requests_per_ student* * # *rooms_in_suite* * *maximum_beds_in_room*, is $O(b^5)$, where b is the maximum number of beds in a room. Since MAX_BEDS is a constant, and a constant raised to a power is constant, the complexity is still $O(1)$. At UNM, the maximum number of beds per room, found in the apartment-style dorm, is six. Furthermore, suites comprise approximately 46% of the total bed space.

and other configurable parameters. Housing also designates the finality of a student's assignment, and room attributes (e.g. gender, smoking and blocked), all of which effect the quality of assignments.

Changing the Objective Function by adding or modifying constraints is an important topic with many subtleties. Constraints must:

- reflect the real-world;
- apply to the entire system equally;
- report errors on an individual basis;
- support gradations where possible, rather than all-or-none, to help produce a smoother landscape;
- be linear time since they are checked in the inner most loop.

These design choices are illustrated in the following examples:

Reflect the Real World To reduce manual consolidation — combining single students assigned to double rooms at the start of fall semester, the *partial_fill* constraint gives each student in the room, regardless of the type of room, the minimum of the empty and occupied beds. When occupancy is at the room's capacity there is no problem.

Apply Uniformly The apartment-style dorms can be configured as a room of six beds or as a suite of six adjoining single rooms. The roommate constraint generally applies across adjoining rooms because we want groups to be together across the suite situations as well as within rooms. Non-smoking consensus does not generally apply across adjoining rooms, however, a shared living area keeps smoking an issue in these dorms regardless of their interpreted configuration. The music, study late and age difference errors do not effect suites — the real world view says these constraints are less critical when students can easily shut their doors, but they would matter in a multi-bed room. All these constraints apply uniformly, whether a consensus of six or the trivial case of one.

Smoother Landscapes To keep the landscape smoother, we want the constraint functions to return more gradual results. For example, spring semester we added a constraint to deter the system from placing new students with returning students, as suggested by Area Coordinator Rob Burford, who often saw this happen with subsequent room changes. The *first_timer* constraint returns the difference in the consecutive semesters between first-timers and old-timers. The more senior the old-timers, the worse the grade for the first-timers – this is more desireable than just counting the old-timers for example, because it is less discrete. A relatively small coefficient sufficed for this constraint to have the desired effect.

Linear Time The *age_difference* constraint works similarly, returning the maximum difference in ages beyond the configured acceptable years apart. To find the largest unacceptable age difference, instead of comparing each student in the room with every other student in the room, we find and compare just the oldest and the youngest, requiring only linear time.

2.2 Choices in Policy

The Objective function is shaped by the priorities set forth by Housing Management policy. As described in Section 1.1.2, the First-come, First-placed and Early Student Notification choices have a particular impact on the problem and the results.

2.2.1 Inflationary Costs

Although position in line, hence lottery number, has little to do with making good matches, we accommodate this traditional practice by allowing Housing to adjust the lottery number coefficient. We ask Housing to consider how important this chance order is compared to residence seniority, such that the first person in line gains some fixed percentage of a semester's seniority benefit, the last person has no benefit, and all the others are scaled in between.

A *non-weighted constraint* has the value specified by the user, such that the more important the effect, the greater its value. A *weighted constraint* has a base value specified by the user, which is multiplied by the student's calculated seniority-lottery-weight (slw) to determine its overall value. To provide greater granularity in lottery we use a seniority weight close to the number of students.[5]

The greater the seniority weight, the greater the non-weighted constraints must be to have the desired effect. To accommodate these inflationary costs, we converted the $GRADE$ in DAO versions v0.6x to an unsigned integer for a global score maximum of approximately 4.2 billion.[6] Weighted contraint values must still be relatively small. Negative values do not exist in the unsigned world, and limit us in two ways: We are unable to support negative constraint values (or credits), and the hardware cannot help us detect overflows. An example of the former is given below. To compensate for the latter, we systematically use software overflow checking whenever we adjust the global grade. If an overflow occurs, the run is aborted, and constraint values must be reduced by the user.

[5]Doubling the seniority weight to 2000 produced more defendable results, according to Ranville's analysis in early Fall 2000 tests.

[6]In limits.h, ULONG_MAX = 4,294,967,295

A 'Good' Constraint
Ranville asked if we could assign the larger rooms in the apartment-style dorm to the more senior students. Since we do not support negative constraint values, we cannot give 'bonuses' when senior students are in these 'better beds', so instead we give an error to all other cases, including students in all the other dorms. In effect, we are raising the sea-level of our landscape. This error destroys the majority of perfect zero grades because it is not really an error — it's a credit!   *Applying the error just to the apartment-style dorm that actually has the better beds, though initially plausible, results in the DAO choosing to move students out of the apartment-style dorm. Applying the constraint globally avoids that faulty behavior.*

To further conserve error bandwidth, in DAO version v0.93, we converted gender into a 'hard constraint', guaranteeing the randomly-chosen bed will always be the appropriate gender, without requiring a large constraint weight to ensure it.

2.2.2 Multiple Assignment Runs

The Management policy of early notification to the students, requiring multiple assignment runs with incomplete information, has consequences to the overall process and results. Tests have shown the overall score suffers after submitting the returning and new students separately.[7] Getting student preferences into the first assignment run is a major priority — this run has the maximum degrees of freedom for making good matches.

Notified only of their hall, however, students assigned a room early may still be moved. We are free to change their bed within the same hall, and possibly their hall if a higher choice becomes available. Were anybody to be moved to a worse hall — not one of the student's requested halls, or a lower ranked available hall — the whole

[7] *Simulated Annealing in Dorm Assignments*, CS451 Project, October 1998.

run would be unusable. To try to prevent this from happening, *automatic upgrading* is implemented as a constraint with an extremely large 'worse hall' penalty.

Preliminary tests show that secondary runs with automatic upgrading can help to compensate for the disadvantage of doing multiple runs, but still they do not compete with the quality of results from a single larger run. This upgrading feature has yet to be used in production, as discussed in Section 4.2.4 (page 52).

2.3 Choices in the Data

The student's primary contribution to the assignment process is filling out and turning in the Residency Application form. Students who remain dorm residents over their college career increase their chances of getting the room they want. Keeping the same room is an overriding choice — essentially a hard constraint for DAO. Students also improve their chances of getting their choices by reserving a place in line early. To be placed with a friend in the same room, roommate requests should be mutual and compatible.

Baffled Roommates
Baffled as to why two friends, less than a month apart in age, were not placed together after requesting each other, the same halls, same music preference, same study habits, and turning in their applications at the same time, the Area Coordinator discovered one was a smoker and the other objected to smoke. Clearly, the smoking error (60,000) vastly outweighed the incomplete group error (4,588) for these two first timers. A smoke compatibility verification, added in version v1.15, revealed over a hundred roommate requests with the same problem. Reported as exceptions, Ranville is now alerted to such conflicting data. Understanding why this happened, future orientations will stress the importance of compatible choices when requesting a roommate.

Since constraints are cumulative, the more choices a student makes, the better their assignment tends to be because they can produce larger errors on mismatches. We found cases where a student who requested the same hall for all three preferences had a better chance of success than a more senior student who had only filled in their first choice. We surmised a similar phenomenon existed with roommate requests — a student with five roommate requests was more often satisfied than the student with only one best friend. Attempting to level the playing field, we added configurable options to duplicate students' hall choices and roommate requests in DAO version v0.45. When the roommate duplication option was tried in early Fall 2000 tests, Ranville noticed students with mate requests and less seniority were given preference over students with more seniority and no roommate requests. Roommate duplication was not used.

Multiple roommate requests also come into play in a new way as adjoining rooms are now considered when placing students. For the student who chooses a double room, the *bestfriend* option tries to place their first valid roommate listed in the same room rather than split them across a suite.

Spending Choices Foolishly
In Fall 2000, we saw first-time residents requesting the prize apartment-style single rooms (about 28% of the total bed space) for their first and second choices and the other popular hall with adjoining rooms (about 18% of the total bed space) as their third choice, get a single in an unrequested dorm; students behind them in line who requesting the adjoining double rooms as their first and second choices and the less desireable hall for their last choice actually got their first choice. The person who gets their paperwork in early should at least get their third choice, according to Ranville, even if they choose to spend their first choices foolishly, at the expense of the wiser student behind them. Rather than leaving some of these students' bad choices to chance, Ranville fixed their requests to get them into the hall they 'deserved'.

Two conflicting important constraints could hurt the student's chances of getting into their requested area. For example, students who request the apartment-style dorms should also request a single room so the two choices do not conflict. Sometimes students fail to indicate they are a non-smoker while requesting a non-smoking special living option (see table on page 56). Occasionally students designate the same music type for both their preference and objection, or request a roommate of the opposite gender — usually a data entry problem. All these exceptions are reported to and remedied by the Housing Reservations staff before assignment processes are run.

Chapter 3

Chances: The Simulated Annealing Solution

The simulated annealing technique used in the DAO was regularly able to find a local minimum within 20 million attempts switching students and beds at random — a vanishingly small sample compared to the 2000 factorial possible combinations of dorm room assignments. At the same time, compared to the HMS system, DAO reduced the processing time from days to hours.

3.1 Simulated Annealing Schedules

Simulated Annealing (SA) has advantages and disadvantages compared to other global optimization techniques, such as genetic algorithms, tabu search, and neural networks. Among its advantages are the relative ease of implementation and the ability to provide reasonably good solutions for many combinatorial problems. Though a robust technique, its drawbacks include the need for a great deal of computer time for many runs and carefully chosen tunable parameters [EFC98].

With the existence of uphill moves, comes the possibility of cycles and random walks, both which stymie SA and are addressed in Section 4.1.6 (page 46). Like hillclimbers, simulated annealing can fall into a local minima far from the global optimum solution (like premature crystalization) [Rib].

Without attempting to provide a comprehensive comparison of global optimization techniques, we present our approach and findings within the specific problem domain of improving dorm room assignments, using simulated annealing.

Given a neighborhood structure, simulated annealing can be viewed as an algorithm that continuously attempts to transform the current configuration into one of its neighbors [vLA87].

Simulated annealing, a Monte Carlo method that can be modeled mathematically using the theory of finite Markov chains [AKvL97]. By definition, a *Markov chain* is a sequence of trials, where the probability of the outcome of a given trial depends only on the outcome of the previous trial. In the case of SA, a trial corresponds to a move, and the set of outcomes is given by a finite set of neighboring states. Each move depends only on the outcome of the previous attempt, so the concept of Markov chains applies. Since the number of the trial does not affect the probabilities, it is considered *homogenous* [AKvL97].

To govern the convergence of the algorithm, a set of parameters, known as a *cooling schedule*, are specified by:

- an initial value of the control parameter (i.e. temperature)
- a decrement function for lowering the value of the control parameter
- a final value of the control parameter specified by a stop criterion
- a finite length of each homgeneous Markov chain.

Determining these parameters is one challenge with annealing. There is much research on the topic, dealing mostly with heuristic schedules [Pic87, BGNZ96, AKvL97]. There are two main categories of heuristic schedules: Static and dynamic. In a *static* cooling schedule, the parameters are fixed and cannot be changed during the execution of the algorithm. With a *dynamic* cooling schedule the parameters are adaptively changed during the execution. The *Mathematical Optimization* chapter of the Computer Science Education Project [CSE96] provides an introduction to two simple static cooling schedules:

> The simplest and most common temperature decrement rule is: $T_{k+1} = \alpha T_k$, where α is a constant close to, but smaller than, 1. This *exponential cooling scheme* (ECS) was first proposed by Kirkpatrick et al. [KGV82] with $\alpha = .95$.
>
> Randelman and Grest [RG86] compared this strategy with a *linear cooling scheme* (LCS) in which T is reduced every L trials [while avoiding negative temperatures]: $T_{k+1} = T_k + \Delta T$. They found the reductions achieved using the two schemes to be comparable, and also noted that the final value of [the objective function] f was, in general, improved with slower cooling rates, at the expense, of course, of greater computational effort. Finally, they observed that the algorithm performance depended more on the cooling rate, $\Delta T/L$, than on the individual values of ΔT and L.

Some general guidelines exist when choosing an annealing schedule, for instance, there is a trade-off between large decrements of the control parameter (T) and small Markov chain lengths (L) — usually small decrements of T, or a ceiling for L, polynomial in the problem size, are chosen to avoid long chains. When the Markov chain length is fixed, it may be related to the size of the neighborhoods in the problem instance. Apparently, when a sufficently long schedule is used, simulated annealing replaces iterative improvement as the optimal schedule [vLA87, AKvL97].

Researchers have proposed more elaborate, mostly adaptive, annealing schedules, using statistical measures to modify the control parameters. Unexpectedly, non-monotonic schedules have been observed as optimal in some situations [vLA87, AKvL97].

3.2 Dorm Assignments: Five Cooling Schedules

Since there seemed to be no basis for deriving a single optimal schedule, we developed five annealing schedules: Two static types, one algorithmic and one table-driven; and three dynamic adaptive algorithms, each with its own criteria for reducing temperature and phase, and determining completion. An algorithm options file facilitates experimentation with the SA parameters.

We began our study in DAO version v0.09 with an ECS variant — starting with a temperature and a phase, after a phase worth of attempts, the temperature is decreased by a configurable factor and the length of the phase doubled: $T_{2k} = \alpha T_k$, where $\alpha = .20$ by default. The process stops when the temperature reaches a configurable value. Running at high temperatures for a short time and at lower temperatures for longer times produced excellent results at the expense of longer than desired elapsed times — taking several hours for a single round. This 'cadillac' algorithmic schedule (denoted **sa0**) remains our baseline for comparison.

DAO version 0.10 introduced a table-driven version (**sa1**) that disassociates the phase and the temperature. Up to sixteen phase-temperature pairs are configurable in the algorithm options file. The extreme flexbility of this table approach has been useful in configuring various test situations, though challenging to refine for production.

In search of easier-to-configure approaches, work on adaptive cooling began in November, 1998 with DAO version v0.17, resulting in two approaches: An '*attempt-based*' scheme (**sa2**) and an '*accept-based*' scheme (**sa3**). Most recently, as of version v1.05, a '*reheating*' scheme (**sa4**) has been developed as well.

The three adaptive versions, similar in structure, try to sense when to change the temperature. In each, a running average of the most-recent one thousand cost differences is kept, where an improved score has a negative cost difference. The attempt-based scheme (**sa2**) uses all attempts, including the unaccepted attempts that produce no change in the overall score, in the history buffer, while the accept-based approaches (**sa3** & **sa4**) keep only the accepted attempts.[1]

When a phase completes (initially, 2,000 attempted or accepted moves, by default) and the temperature hasn't changed for a configurable number of phases, temperature is computed, based on comparing the standard deviation of the last 1,000 attempted or accepted cost differences to the previous standard deviation. The schemes differ as follows:

<table>
<tr><th>STD <=> LastSD</th><th>sa2</th><th>sa3</th><th>sa4</th></tr>
<tr><td>less than</td><td>cool TEMP by TEMP_DIVISOR</td><td>cool TEMP by TEMP_DIVISOR</td><td rowspan="3">the average of STD and LastSD by TEMP_DIVISOR</td></tr>
<tr><td>equal to</td><td>cool TEMP by TEMP_DIVISOR</td><td>-</td></tr>
<tr><td>greater than</td><td>-</td><td>-</td></tr>
</table>

Inspired by the method of **reheating as a function of cost** (RFC) [EFC98], the adaptive scheme (**sa4**) uses the standard deviation of the cost changes as the basis for the temperature. To compensate for the extreme variability of the standard deviations, we use the average of the standard deviations for the current and previous phases and divide this average by TEMP_DIVISOR to maintain a downward momentum.

[1] The choice of 1000 for the history buffer is questionable. Arguably, the size of the history buffer should be adjusted to the phase, the number of students, the number of beds, or both. With the accept-based approaches (**sa3** & **sa4**), when the phase is 2000, there is complete turn around in the 1000 cost changes used to calculate the standard deviation. As the phase is reduced to less than 1000 in **sa3**, there is possible and probable overlap in the history at these later decision points. With the attempt-based approach (**sa2**), all the zero difference attempts are included in the history, providing little possibility of overlap.

The TEMP_DIVISOR is a configurable parameter set to ten by default.[2] A phase count is used as a delay mechanism to determine the number of phases to wait before trying another temperature adjustment.[3] If the temperature is changed, the phase count starts over at zero, otherwise, each scheme does one of the following:

STD <=> LastSD	**sa2**	**sa3**	**sa4**
less than	-	'ease' phase test	incr phase-count
equal to	-	'ease' phase test	incr phase-count
greater than	incr phase-count	incr phase-count	incr phase-count
zero	incr phase-count	-	-

The accept-based adaptive scheme (**sa3**) also considers '*easing*' down the phase length — reducing the number of accepts required before recomputing parameters — to compensate for the increase in attempts required per accepted move at cooler temperatures. This 'ease' test compares the percentage of improved grade differences over the number of accepts to DECIDE_TO_EASE (set to 10 percent by default). If the percentage is less, we reduce the phase length by the ACCEPT_REDUCER (in half by default), so the next decision point is reached after fewer accepts and attempts. The other accept-based schedule (**sa4**), uses a cost-sensitive temperature instead of 'easing', while the attempt-based approach (**sa2**) also increments the phase count when the standard deviation is zero, as noted in the previous table.

The base case is a temperature less than ENDTEMP (.001 by default). Additionally, the phase value can terminate a round in the accept-based adaptive scheme (**sa3**), and the table-driven scheme (**sa1**).

[2] We found a value of five for the TEMP_DIVISOR is better, though slower.

[3] The delay can be extended by setting the value of the DECIDE_TO_COOL configuration option (zero by default) higher. We used a value of one, which gave us at least three phases at each temperature.

The cooling schedule for each round, per batch, can be limited to one scheme, or rotated among all the schemes beginning with any specified scheme. We keep track of the best grade over a series of rounds until we fail to see a better grade within a pre-configured number of rounds (NOBETTER).[4] At this point, we restore the best state found and switch to a hillclimbing algorithm, as discussed in Section 4.3.7.

Figures 3.1 through 3.3 show the 'Temperature vs Attempts' graphs for sample rounds that achieved the lowest overall results per scheme. The two static approaches were the same each semester, while the three dynamic schedules varied. The smooth curve, produced by **sa0**, took the greatest number of attempts each semester, while the steepest graph with the fewest total attempts each semester was made by **sa1**. The stair-step curves made by the two adaptive schedules, show **sa3** quit before **sa2**, due to the 'easing' process. The reheating adaptive schedule, **sa4**, was non-monotonic and stopped abruptly after nearly half the attempts taken by the cadillac.

	Fall 2000		**Fall 1999**		**Fall 1998**	
	Test	Actual	Test	Actual	Test	Actual
Perfect Score (%)	53.7	48.3	45.6	32.5	49.3	22.7
Satisfaction (%)	94.6	94.0	92.4	87.3	90.7	81.4

Table 3.1: Percentage Results from Sample and Actual Performances

Percentage indicators for each semester, from the best of our sample test rounds along side the actual performances, are presented in Table 3.1. Since changes in the objective function and data between DAO runs voids any score comparisons, other indicators, such as 'perfect score', students without an error, and 'satisfaction', percentage of students receiving one of their requested halls, as well as error counts, may be used instead. Statistical test scores using the five cooling schedules are compared in Section 4.3 (page 55).

[4]The number of rounds can be pre-determined with the LOOP_CONTROL option.

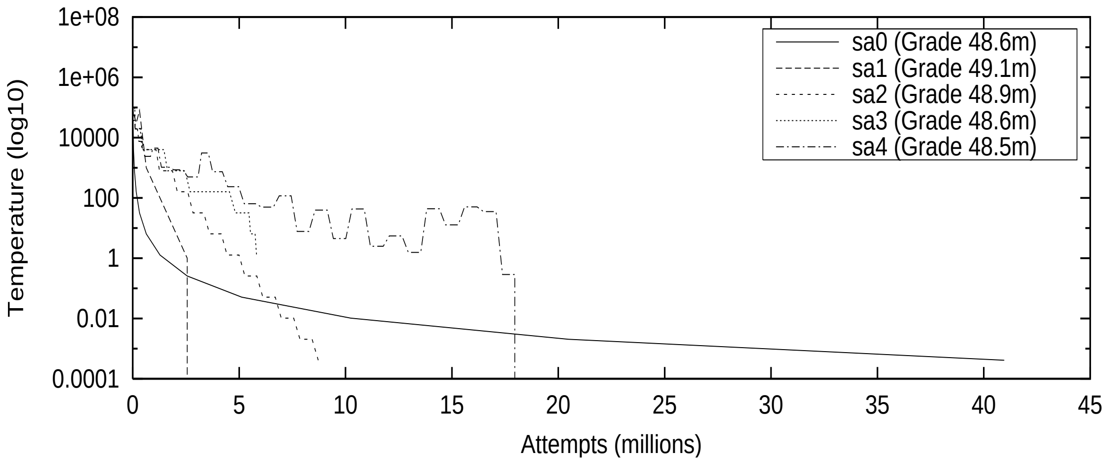

Figure 3.1: Temperature vs Attempts by Scheme — Samples from Fall 2000

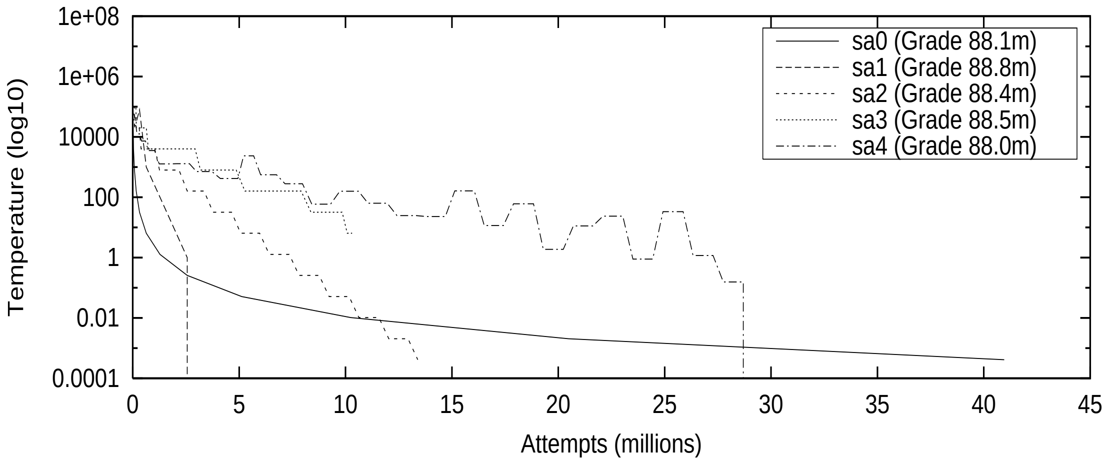

Figure 3.2: Temperature vs Attempts by Scheme — Samples from Fall 1999

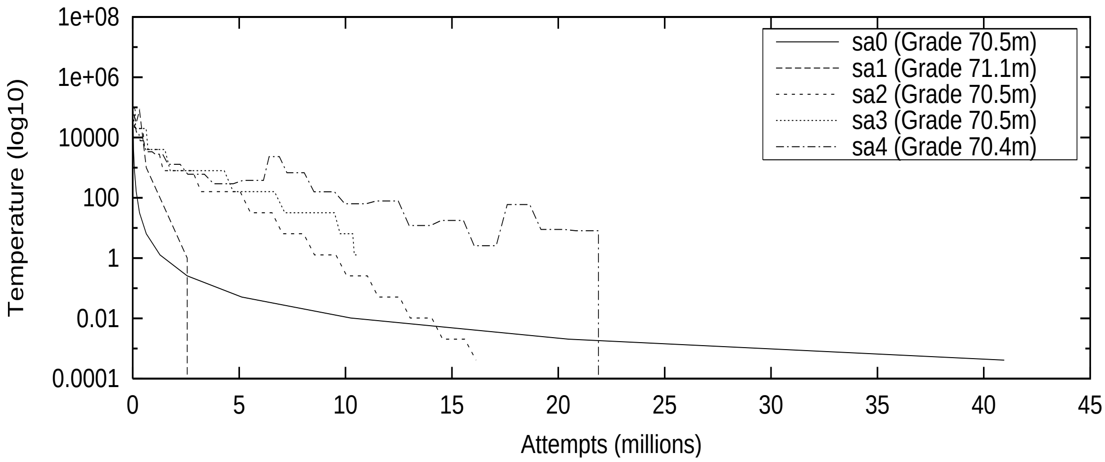

Figure 3.3: Temperature vs Attempts by Scheme — Samples from Fall 1998

3.3 Guaranteed Local Minimum

Simulated annealing is guaranteed to find a global optimal solution, a result in [AKvL97]. Unfortunately, to get that guarantee requires an exponentially long cooling schedule, and is therefore impractical. We'd settle for a local minimum, where no better moves exist in the neighborhood, but reasonable length annealing schedules do not even promise that! So in the DAO, we follow our five schemes with a deterministic hillclimber that guarantees a local minimum at some additional expense.

This iterated next-descent hillclimbing algorithm [Ack87b] considers the students one-by-one, either in arbitrary order from a random start (**mr**) or in seniority order (**m**), depending on the configuration, and tries placing each student in every other available bed looking for lower scores. Any bed switch that improves the overall grade is accepted, and the student index is saved. From there we continue until we reach the last student and last bed choice. If we had any improvements, we start over until we have looped to the student index last changed. We end up with the best grade such that there is no better state one move away — a local minimum. The landscape of the results of a complete DAO run (page 17) shows no single move improves the overall grade.

This minimization step can be configured to take place once on the best solution ever seen, or after each round. The ramifications are discussed in Section 4.3.7.

3.4 Discussion: Adaptive Schedules

Why do our adaptive approaches seem to work well? What makes the standard deviation of grade differences a good barometer for detecting apparent equilibrium?

From experience, we've seen that running briefly at high temperatures and longer at lower temperatures produces good results, and this is sensible because, equilibrium is approached more quickly at high temperatures. The overall structure of the solution is shaped at high temperatures, while at low temperatures the fine details of the solution are resolved [EFC98]. After many moves, approaching equilibrium, we want to cool the temperature, making it more difficult to make a bad move.

The standard deviation indicates the spread in the grade changes of the moves being attempted or accepted. Since one standard deviation includes about 60% of all the grade changes, larger standard deviations indicate motion, while smaller ones reflect similar grade changes, perhaps an indication of 'settling'. In the attempt-based scheme, which includes the zero grade differences in its history and calculations, a zero standard deviation could also be an indication of settling.

The only cooling schedule that can potentially 'reheat' the temperature is **sa4**. The temperature is directly proportional to the average standard deviations of the changes in cost. Since the standard deviation can be larger or smaller, the temperature change is non-monotonic. Our approach is similar to the RFC method described in [EFC98], "Reheating is done when the temperature drops below the phase transition (the point of maximum specific heat) and there has been no decrease in cost for a specified number of iterations,... Reheating increases the temperature above the phase transition to produce enough of a change in the configuration to allow it to explore other minima when the temperature is again reduced." Although we borrowed the notion of a relationship between *specific heat*[5] and temperature, we use the square root of the variance of cost changes as a basis for setting the temperature. Due to the reheating possibilities, running times for **sa4** are less predictable than with the other two dynamic schemes as seen in Figures 3.1 through 3.3.

[5] "Specific heat is a measure of the variance of cost (or energy) values of states at a given temperature" [EFC98].

3.5 Handling Ties

It is easy to overlook the question of what to do about 'lateral' ties, moves that leave the score unchanged, yet change the state. The decision can have significant consequences. Unexpectedly, it turns out that simulated annealing, the Metropolis criterion, random lateral descent hillclimbing (**rldhc**), random descent hillclimbing (**rdhc**), and iterated next-descent hillclimbing (**m**) each handles ties differently. The probabilities of accepting differing quality moves by these methods are shown in Table 3.2.

Method	**Downhill**	**Uphill**	**Lateral**
sa	sigmoid	sigmoid	sigmoid (= 0.5)
metropolis	1.0	sigmoid	0.5 or 1.0
rldhc	1.0	0.0	0.5 or 1.0
rdhc	1.0	0.0	0.0
m	1.0	0.0	0.0

Table 3.2: Probability of Accepting Moves by Methods

If the landscape around the current state is a 'mesa', it could take many lateral moves before progress can be found. While never taking lateral moves is a feature for iterated next-descent hillclimbing in avoiding possibly never terminating, we may never see the edge of the mesa while searching.

Our simulated annealing approaches symmetrically invoke the sigmoid regardless of the sign and magnitude of the change, yielding a 50-50 chance of accepting lateral moves regardless of temperature.

The Metropolis Algorithm, a well-known simulated annealing algorithm, differs from our uniformly sigmoid-based approach in that it always takes better moves. Its acceptance criterion for lateral moves, however, was left unspecified in the original 1953 journal article [MRR+53]:

> We then calculate the change in energy of the system ΔE, which is caused by the move. If $\Delta E < 0$, i.e., if the move would bring the system to a state of lower energy, we allow the move and put the particle in the new position. If $\Delta E > 0$, we allow the move with probability $\exp(-\Delta E/kT)$; i.e., we take a random number ϵ_2 between 0 and 1, and if $\epsilon_2 < \exp(-\Delta E/kT)$, we move the particle to its new position. If $\epsilon_2 > \exp(-\Delta E/kT)$, we return it to its old position.

Subsequent work has differed on their treatment of the $\Delta E = 0$ case, varying from always taking lateral moves (e.g. [vLA87, KGV83, JAM89, AKvL97, EFC98]) to taking them half the time ([vLA87, Ack87a]).

Between DAO versions v1.10 and v1.14, we added the Metropolis criterion, and an adjustable acceptance probability of lateral moves for random descent hillclimbing and Metropolis to test the impact of differing acceptance probabilities on the final outcome, in our problem domain. These results are reported in Sections 4.3.5 and 4.3.6 (starting on page 62). For comparisons with our simulated annealing schemes, presented in Section 4.3.7 (page 64), we used the sigmoid probability of accepting ties for both random lateral descent hillclimbing and the Metropolis algorithm.

Chapter 4

Results: Fall 1998 – Fall 2000

The DAO has been the principal method of dorm assignments over the last three semesters at UNM, without replacing the previous system entirely. We scored on both targeted areas of improvement: Usability and resource utilization.

Both experimental and production use of the DAO has provided many insights into the dorm assignment problem and the simulated annealing solution. At version v1.15, the artifact exists as both a practical tool to assign beds, and an academic tool to investigate simulated annealing. Presented in this chapter are case studies and tests spanning three fall semesters.

4.1 The Artifact

Motivated and constrained by the limitations of existing system, we have attempted to provide the Housing staff with less frustrating ways to view and manipulate the data to achieve better and faster assignments.

4.1.1 Choosing Constraint Coefficients

Providing an understandable way to adjust the objective function is essential to finding good coefficients and is key to its usability. In the HMS assignment system, four rules, like this, are required to put students of nearer ages together:

```
18 years +/- 1 year    64
(not empty(applicant.birthdate) and not empty(occupant.birthdate))
and between((date()-applicant.birthdate),6205,6934) and
between((date()occupant.birthdate),6205,6934)
```

Such rules, though relatively general, are hard to understand and modify. In the DAO, by contrast, the same constraint is expressed as two configuration parameters:

```
ERR_AGE_DIFF = 3
ACCEPT_AGE_DIFF = 2
```

Seniority vs Smoking Question
A student with seven semesters seniority did not get into any of his requested halls because his smoking status caused too many red flags to go up! Ranville was able to lower the smoking error enough to get him into a single room in his first choice non-smoking dorm (with the understanding he would not be able to smoke in his room), without causing havoc among the other less senior smokers.

For the most part, at a constant temperature, larger coefficients create a more rugged landscape, while smaller ones make a more even terrain. The smaller coefficients become more important as the temperature is lowered. Except for dorm and roommate preferences, the rules apply evenly among all the residents. Mixed weighted and unweighted constraints complicate the job of balancing the values chosen, but serves to express the seniority and reservation order advantages.

Mystery Unraveled
A student requesting a single room was assigned a single room in a hall they didn't want, instead of a double room in their third choice hall, even though some lower priority students did get in. Other similar cases were found. Trying the obvious fix — increasing the third choice hall error — still left some students misplaced.   *Further investigation uncovered the real explanation: The DAO was constrained because those lower ranked students' other hall choices were already filled, except for special living option rooms.*   *Summary reports generated by the DAO revealed that many of those special living option rooms were ultimately left vacant. When Ranville removed the excess special living designations and reran the DAO, the problem was solved.*

More quickly than ever before, Housing can generate a complete set of reports to describe the current state of the HMS database. The ten DAO reports provide clues for adjusting constraint coefficients and improving assignments. Dorm_View allows the user to further refine the reports by hall, and visualize problem areas.

4.1.2 Between Two Systems

The existing HMS system is used for many other purposes beyond just dorm room assignments. So DAO must co-exist with, rather than replace HMS, and this co-existence imposes some overhead.

Using the new approach requires: Executing six queries to export the data out of HMS, logging into a Unix system, running the DAO, and importing the data back into the master system (page 12).

An HMS assignment run, using a special set of rules (Appendix A) that say 'just take our suggestions', actually performs the assignments. Instead of updating HMS tables directly, we use this round about procedure to avoid voiding the 'warranty'. This last step, which runs on a local copy of HMS to avoid network overhead, takes about 30 seconds per student on a 350MHz workstation. As mentioned in Section 1.1.1, the HMS roommate logic is not rule-based, and thus we cannot alter it by changing the HMS rules. However, we discovered that emptying the roommate database on the local copy rendered the overpowering HMS roommate logic harmless.

Finally, five occupant related files are uploaded to the master copy to complete the assignment process. These final steps in the process are streamlined with the use of batch files selected through window system shortcuts.

A benefit of this approach is that nothing in the master system changes until the data is imported and the master database need not be locked for the entire duration of the assignment process. The DAO can easily run during normal business hours. Besides running in less time, without weekend restrictions, results become available in a timelier manner.

4.1.3 Batch Management

Each DAO run, consisting of one to many rounds, is uniquely identified by a batch number. All the data associated with a run is organized in a directory named by the batch. To ease archiving, batches are further organized under a term directory. Each batch directory has subdirectories for REPORTS, SAVEDATA, GRAPHS, with the most generally useful files (summary, exception, runlog and the HMS import) at the top level.

4.1.4 Checkpoint Restart

To minimize runtime loss due to a system crash, network loss, or electrical outage, DAO has the ability to restart a run from checkpoint files written after the last completed round. The data is saved in five files: A small control file containing the latest summary information, two 50K byte files saving the current and best student occupancy data, and two smaller files for the current and best waiting lists. For operating systems supporting renaming operations, we use the OLD-NEW-LST suffixes, otherwise we use generational suffixes. The former maintains only the last two checkpoints, while the latter depends upon the number of rounds.

An internal version number, manually assigned to the objective function, indicates whether the global grade can be verified during a checkpoint restart — any change to the objective function, including user-designated coefficients, invalidates, and therefore, bypasses the verification.

The checkpoint files are also used by Dorm_View to display the results of a previous batch.

4.1.5 Security Issues

Given the sensitive nature of the data to be communicated between the two systems, security was a concern.

The **ncpmount** Linux utility enabled us to mount our Novell file server volumes, access the data files and output the results from the Linux machine directly to the Novell server, avoiding potentially insecure ftp access. Both machines were located in the same office. **Tcp_wrappers** and **ssh** secured access to our Linux box from other locations.

4.1.6 Efficiency Considerations

Several authors (e.g. [MS91, vLA87, AKvL97, EFC98]) concur that a risk with simulated annealing is its extensive running times. The potential for cycles exists for the same reason SA does so well — its willingness to take uphill moves. By refining neighborhoods, simulated annealing has become a technique used to reduce random walk behavior [Mac99].

Of the DAO's three main processing stages (Table 4.1), our concerns for efficiency lie in the middle:

Read and parse the input data and configuration files to create internal data structures;
Optimize student bed assignments with CPU-intensive SA algorithm on memory-resident data;
Output assignments & reports.

Table 4.1: DAO Process Flow

Linear time grading functions and memory-resident data structures support the need for a tight inner optimization loop to keep running times manageable. Profiling version v0.83 of the DAO[1], revealed inefficiencies in the *age_consensus* and *music_consensus* grading functions, and these were subsequently linearized. Available beds and moveable students are accessed directly through indexed arrays providing $O(1)$ access with no search or filtering required. For efficiency, the objective function is calculated most often by quickly applying changes in the cost resulting from a move, reserving complete calculations of all the students for the phase interval as an invariant insurance, as discussed in Section 2.1.2 (page 20).

Given our acceptable running times and results, we opted not to maintain a memory of some recently visited points (tabu solutions) and incur the extra overhead to

[1]The UNIX utility **gprof** was included as a Makefile option.

recognize repeating them to help prevent cycling [HTdW97]. To diversify the search and avoid unexplored neighborhoods we choose each move randomly. Furthermore, we reset the random seed at the start of each round, and optionally start from the same initial configuration at each round, or continue with the results from the last round. With the size of our problem and a random selection method, there is little chance of picking a point and the same subsequent point, repeatedly. During the minimization step, the iterated next-descent hillclimber avoids cycles by never accepting lateral moves.

As a CPU-bound process, linear speed up may be attainable as processor speeds double about every eighteen months. This was evident with the 350MHz Pentium II, and 700MHz Pentium III dedicated machines used in our tests. The average processing speed was approximately nine thousand attempts per second on the slower processors, and seventeen thousand attempts per second on the faster machine. The post-round minimization steps were a quarter-time slower at 6,700 attempts per second on the 350MHz computers, and 12,480 attempts per second on the 700MHz machine. Though it has yet to be investigated, we hypothesize the difference was due to a higher proportion of unaccepted moves during the minimization step, requiring the original state to be restored.

4.2 Case Studies

Case Studies, though not scientific, help provide context for the work. Discussed here are some significant lessons learned over the course of DAO's development, from Fall 1998 through Fall 2000.

4.2.1 Fall 1998, versions v0.01 - v0.39

Preliminary test results in Fall 1998 were convincing enough to take our solution to Housing Management:

- An overall grade improvement of approximately 67% appeared achievable, with almost 50% more students receiving a perfect score;
- Roommate requests across adjoining suites were supported;
- Preferences and roommate requests of returning students staying in their same room were considered;
- Reduced running time — running in hours rather than days;
- Without weekend restrictions, the Housing staff was not prohibited from using the HMS system;
- With an understandable way to adjust the grading function, hall preferences could be made more important than roommate requests, and age differences could be minimized.

The final data run through the same objective function as all our subsequent tests, ended with a global score of 199,091,532 with 438 perfect grades, and 81.4% satisfaction. This was based on multiple weekly iterations, new and returning students in separate initial runs, manual roommate corrections for returning students, and blocked halls and desireable rooms to stage placements. With a score almost three times that of our test results, room for improvement was apparent.

4.2.2 Fall 1999, versions v0.40 - v0.85

At this time, Ranville was willing to duplicate her efforts and use both systems to test the worthiness of our new approach. There was some concern that seniority, especially lottery numbers, would not work to management satisfaction — our method was not as easy to explain as the previous assignment algorithm. All the reports, provided in seniority order, have shown simulated annealing produced assignments that imitate the deterministically-ordered seniority model well enough that this potential deal-breaker disappeared as an issue.

To minimize bad assignments, Ranville and her staff have incorporated the parsing and data checking provided by the DAO into the pre-assignment data entry process.[2]

In response to our concern that multiple runs might yield less than optimal results, Ranville agreed to try new and returning students together for the first run, and every two weeks thereafter. This was a big win as the majority of the applications would be in the initial batch.

Ranville still needed the ability to block rooms manually. Minimizing this need became the inspiration to create a constraint to effectively upgrade students automatically. Although automatic upgrading was available, there was no way to distinguish moveable students from the final manual placements Ranville made day-to-day while talking to parents, students and coaches. Automatic upgrades would have to wait.

Unfortunately for science, this first use of DAO proved so successful that Ranville quit the side-by-side testing and made the decision to use our assignment program exclusively instead. We learned some important lessons this first summer of production:

[2] **Flex** and **Bison** UNIX utilities are used to parse each of the input data files, providing extensive data checking.

1. HMS roommate logic could ignore our suggested placements;
2. HMS had to be completely shutdown before copying its files;
3. Setting the $partial\_fill$ error too high at the beginning of the summer when there were plenty of spaces gave us unexplainable placements;
4. Though presumably helpful to the HMS assignment process, filling in choices for students who left their preferences blank on their application forms, took away some degrees of freedom from the DAO;
5. Automatically setting all non-smokers as *objectors* removed the possibility of giving those who actually made the designation on their application extra consideration placing them in non-smoking dorms (discussed on page 6).
6. We needed a way to distinguish moveable and non-moveable assignments for automatic upgrading to work.

Running the final assignments after manual upgrading through the objective function, we found Ranville achieved a global grade of 171,948,075 where 608 students had a perfect grade, and there was 87.3% satisfaction where most students were placed in one of their requested hall. Anecdotally, the Area Coordinators noted there were fewer room change requests.

4.2.3 Spring 2000, versions v0.90 - v1.03

For the spring semester, the first question was: With only two hundred new student placements was it worth using the new system? According to Ranville, the answer was an overwhelming 'yes'.

New requirements surfaced for the second half of the academic year, such as the *first_timer* contraint to put new students together; blocking 'doubles-as-singles' so returning students can keep their double room to themselves; and keeping specific beds empty in rooms where the occupants agreed to be hosts to visiting students.

4.2.4 Fall 2000, versions v1.04 - v1.15

The Dorm_View GUI and the automatic upgrading capability were in place. For the first time, Ranville had the tools to make manual assignments with more complete knowledge of the consequences of each move. The FINAL_OCC attribute added to HMS provides her the means to effortlessly identify who can be moved. Dorm_View supports this working model of manual upgrades by providing detailed results in terms of the objective function before the changes are committed.

Ranville remarked how quickly she was able to modify the initial results of the DAO with Dorm_View (Figure 4.1) — she made 38 changes in a few hours. She used the *blocked/unblocked* room lens, and the *final/moveable/new/moved* student lens. Gliding over the rooms with the mouse she could also see the gender of the room. She chose to view all the dorms in one window to facilitate the Find Student function, and the waiting list in a separate window. The scrolling messages documented the moves she made so she could regain state between interruptions.

After two weeks, Ranville had improved the overall results since her first post-DAO attempts, giving 82 additional students one of their three requested hall choices. Analyzing her changes, it appears the *first_timer* and *mixed_study_late* constraints could have been smaller, and the *better_bed* constraint used. Keeping in mind that it is easier to modify a few bad bed assignments than to change coefficients that have a global impact, the output will occasionally require the human touch to break the rules.

We learned a critical lesson with these first few manual upgrades: Students moved to different beds in the same hall also had to be 'de-bedded', otherwise HMS would search for another place if the suggested bed is occupied. The de-bedding process took eleven key strokes per student in HMS, so too many beds changes would make auto-upgrading unusable in the current environment.[3]

Subsequent test runs produced over 650 same hall bed changes to improve 25 hall assignments; we do not know how many were necessary or just because there was no reason not to. In version v1.12, we gave it a reason slightly less dramatic than never accepting a lateral move — the *different_bed* constraint, designed to delicately tag moving bed positions in the same hall. Same hall bed changes dropped by a factor of six in test runs.

With only a few cancellations automatic upgrading was not used this semester. A smaller resident population and more group matches was credited for the small number of cancellations. Though the automatic upgrading feature and the *different_bed* constraint were not tried, Ranville found power in the interactive Dorm_View tool to upgrade students manually, faster and better.[4]

Three subsequent DAO runs placed 100, 90, and 69 more new students — only 16.4% of the 1,578 students in the initial run. These runs were made with the more desireable dorms blocked, forcing the system to place the late-comers into the unpopular dorms. This legacy blocking practice provides control and is well-understood.

At the end of August, based on 1715 residents, the results showed: A total score of 100,417,531 with 94.0% satisfaction, and 829 students with perfect grades.

[3]The HMS auto-assignment batch could have been backed-out and another configuration imported and re-assigned in about five hours, but by then too much time had been invested in manual upgrades that would have been lost by reversing the batch.

[4]During 'Move-In-Week', 'self-upgrading' replaced auto-upgrading as evidenced by a poster announcing: DOUBLES AS SINGLES — MOVE-IN SPECIAL — Rooms in Coronado and Santa Clara — *Available for Immediate Occupancy* (six male rooms & six female rooms).

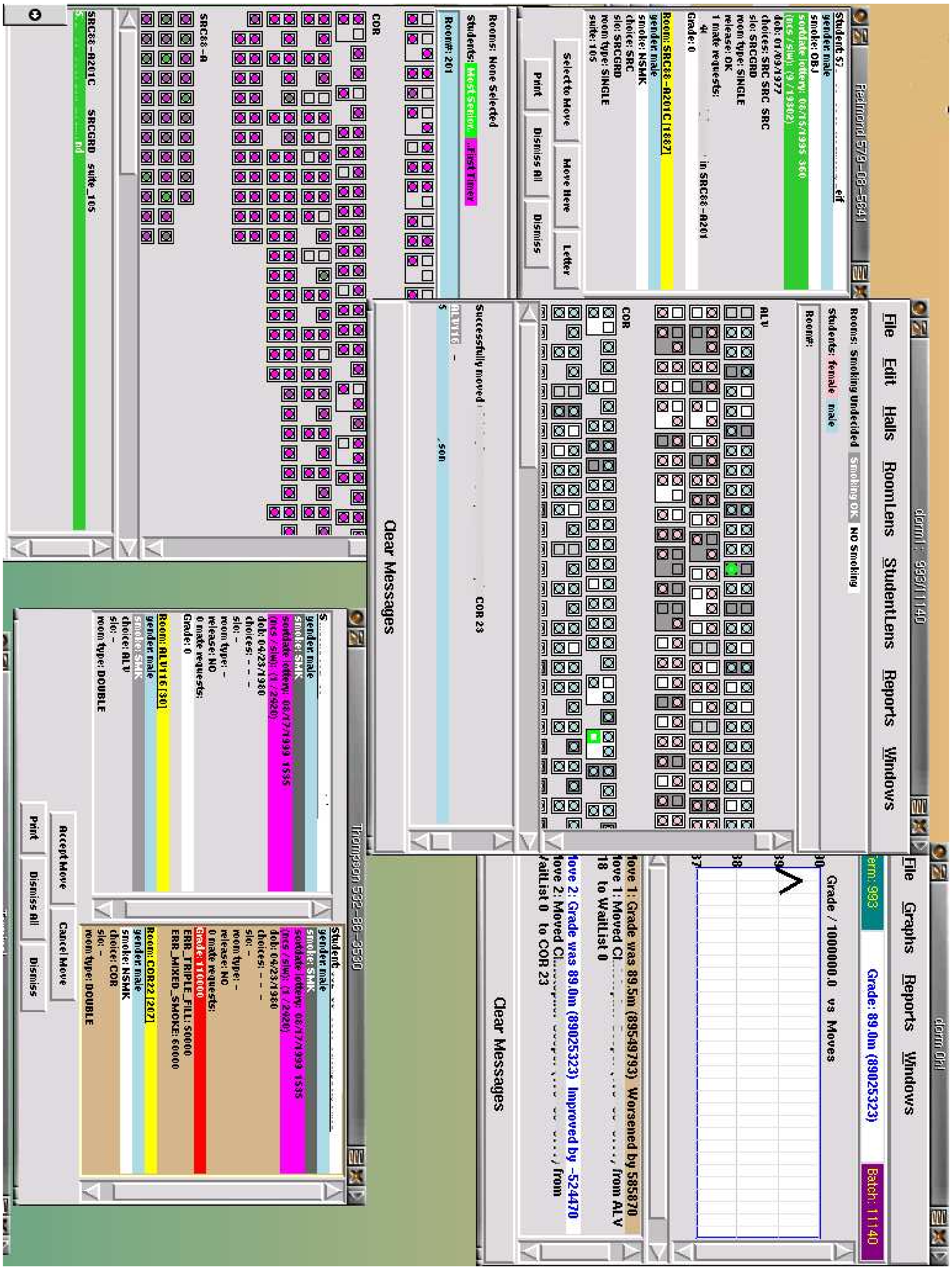

Figure 4.1: Dorm_View Screenshot

4.3 Experimental Results

Complementing the case studies, we performed experiments to better understand the power of randomization and our cooling schedules as referenced in Table 4.2.

Abbreviation	Description
sa0	algorithmic
sa0/m	algorithmic, + min step
sa1	table-driven
sa1/m	table-driven, + min step
sa2	attempt-based adaptive
sa2/m	attempt-based adaptive, + min step
sa3	accept-based adaptive
sa3/m	accept-based adaptive, + min step
sa4	reheating accept-based adaptive
sa4/m	reheating accept-based adaptive, + min step
sigmet1	Metropolis and table-driven schedule (half laterals)
sigmet1/m	Metropolis and table-driven schedule, + min step
sigmet4	Metropolis and reheating schedule (half laterals)
sigmet4/m	Metropolis and reheating schedule, + min step
met1	Metropolis and table-driven schedule (full laterals)
met1/m	Metropolis and table-driven schedule, + min step
met4	Metropolis and reheating schedule (full laterals)
met4/m	Metropolis and reheating schedule, + min step
rdhc	random descent hillclimbing (no laterals)
rdhc/m	random descent hillclimbing, + min step
rldhc	random lateral-descent hillclimbing (half laterals)
rldhc/m	random lateral-descent hillclimbing, + min step
rldhc2	random lateral-descent hillclimbing (full laterals)
rldhc2/m	random lateral-descent hillclimbing, + min step
mr	minimize using interated next-descent hillclimbing (random start in arbitrary order)
m	minimize using interated next-descent hillclimbing (in priority order)
r	randomized

Table 4.2: Key to Methods

4.3.1 Characteristics of the Data

For our experiments, we used the Fall datasets from 1998, 1999 and 2000. The Spring dataset was not used as it would have required changes in the objective function to be accurate.

Fall Yr	**# students**	**# beds**
2000	1578	2050
1999	1871	2064
1998	1930	2085

The 1998 dataset had not benefitted from DAO's extensive data integrity checking, unlike the later years. Essentially all the data entry errors were eliminated in the 2000 dataset, and less than a dozen problems with contradictory music tastes and birthdate errors remained in the 1999 dataset. There were 457 empty preferences for room type and first choice halls, and 23 first hall choice and special living option mismatches in 1998. Smoking students were the source of most of the contradictory preferences each year as they requested non-smoking halls, non-smoking special living options and non-smoking roommates:

Fall	**#**	**Requests**		
Year	**Smokers**	**% NS Hall**	**% NS SLO**	**# NS Mates**
2000	126	35.0	04.8	144
1999	173	33.0	33.0	80
1998	142	27.5	08.5	68

4.3.2 Methodology

Using the three datasets, we ran a series of 32 runs, using DAO versions v1.10 to v1.15, varying only the cooling schedules and the dataset. Each round within a term's dataset began with the same arbitrary placement of all students into available beds, followed by a randomization step[5]. The objective function penalties remained constant, and each round used a different random number seed. All students were moveable and no beds were blocked. A minimization step in priority order followed each round.

Using non-parametric ranking methods which make no assumptions about the distribution of the samples, we calculate the *mean rank*[6] of various approaches listed in Table 4.2 based on the ranks of their measurements rather than the measurements themselves.

Ranking the data from lowest to highest scores, we use the Kruskal-Wallis test, often called an "analysis of variance by ranks", for groups with three or more sample populations. For tied ranks, the rank assigned is the mean of the ranks they would have been assigned had they not been tied.

In each case, the null hypothesis asserts the populations are the same. If the test statistic is greater than $\alpha = .05$ (and in most cases .01), we reject the null hypothesis, and conclude the samples come from different populations.

To determine between which of the samples significant differences occur, multiple comparison tests of the rank means are made against the $Q_{\alpha,k}$ test statistic [Zar99].

The following sections present our specific investigations and observations.

[5]For RANDOMIZING_PHASE times (default 100,000), we randomly select a student and a bed and switch them with a 50% probability, like running at a high temperature.

[6]The mean rank allows for varying sample sizes: A group's rank sum is divided by its sample size.

Scheme	Avg Score	Std Dev	Avg Best	Std Dev	Avg Attempts
sa0	49.084	0.270	49.082	0.276	54.106
sa1	52.170	0.887	49.814	0.303	6.340
sa2	49.276	0.324	49.174	0.209	12.095
sa3	50.062	0.620	49.284	0.258	8.171
sa4	49.069	0.308	49.055	0.288	19.528
sigmet4	49.051	0.235	49.032	0.236	18.605
sigmet1	52.022	0.857	49.741	0.366	6.162
rldhc	49.121	0.244	49.115	0.248	42.526
mr	667.506	10.266	667.506	10.266	$\tilde{4}0.000$
m	671.013	7.197	671.013	7.197	$\tilde{4}0.000$
r	933.03	11.232	933.03	11.232	.100

Table 4.3: Statistics (in millions) from Fall 2000 Tests

Scheme	Avg Score	Std Dev	Avg Best	Std Dev	Avg Attempts
sa0	88.685	0.188	88.683	0.194	43.376
sa1	92.640	0.872	89.306	0.238	8.379
sa2	88.858	0.292	88.777	0.226	17.735
sa3	89.281	0.498	88.722	0.189	13.019
sa4	88.629	0.243	88.620	0.229	27.608
sigmet4	88.570	0.214	88.562	0.213	26.884
sigmet1	92.998	1.015	89.362	0.256	8.482
rldhc	88.750	0.221	88.743	0.218	43.361
mr	867.751	8.405	867.751	8.405	$\tilde{4}0.000$
m	863.249	7.132	863.249	7.132	$\tilde{4}0.000$
r	1156.37	35.924	1156.37	35.924	.100

Table 4.4: Statistics (in millions) from Fall 1999 Tests

Scheme	Avg Score	Std Dev	Avg Best	Std Dev	Avg Attempts
sa0	71.336	0.831	71.329	0.839	44.162
sa1	75.310	1.080	72.049	0.881	9.182
sa2	71.078	0.520	71.062	0.514	19.830
sa3	71.606	0.895	71.268	0.760	14.021
sa4	71.193	0.805	71.173	0.803	31.728
sigmet4	71.077	0.740	71.071	0.734	31.255
sigmet1	74.810	1.209	71.788	0.807	9.102
rldhc	70.992	0.544	70.987	0.533	43.772
mr	836.634	5.462	836.634	5.462	$\tilde{4}0.000$
m	838.684	5.437	838.684	5.437	$\tilde{4}0.000$
r	1125.95	13.412	1125.95	13.412	.100

Table 4.5: Statistics (in millions) from Fall 1998 Tests

4.3.3 Random Optimization

Are our random optimization methods better than resource-matched deterministic hillclimbing?

For our deterministic entry, starting with randomized configurations, using a null entry table-driven scheme, we took the best of five iterated next-descent hillclimbing rounds (approximately 40 million total attempts), in both random (**mr**) and priority order (**m**) as described in Section 3.3. The random and priority orders of iteration did not register any difference.

Pairwise comparisons consistently showed the iterated next-descent hillclimbing rounds were different than all the randomized methods tested, except the unminimized table-driven schemes (**sa1** & **sigmet1**) in all terms, minimized table-driven schemes (**sa1/m** & **sigmet1/m**) in 1999, and the unminimized accept-based adaptive scheme (**sa3**) in Fall 2000.

On average, the iterated next-descent hillclimbing scores (**m** & **mr**) were an order of magnitude greater than all the others[7], with wider standard deviations (Tables 4.3 through 4.5), forcing us to conclude our randomized methods have a distinct advantage over simple hillclimbing.

4.3.4 Taking Uphill Moves

Given a random algorithm, does varying non-zero probabilities of accepting an uphill move work better than never taking an uphill move?

[7]Iterated next-descent hillclimbing did, however, improve the score of an average random state (**r**) by 25%.

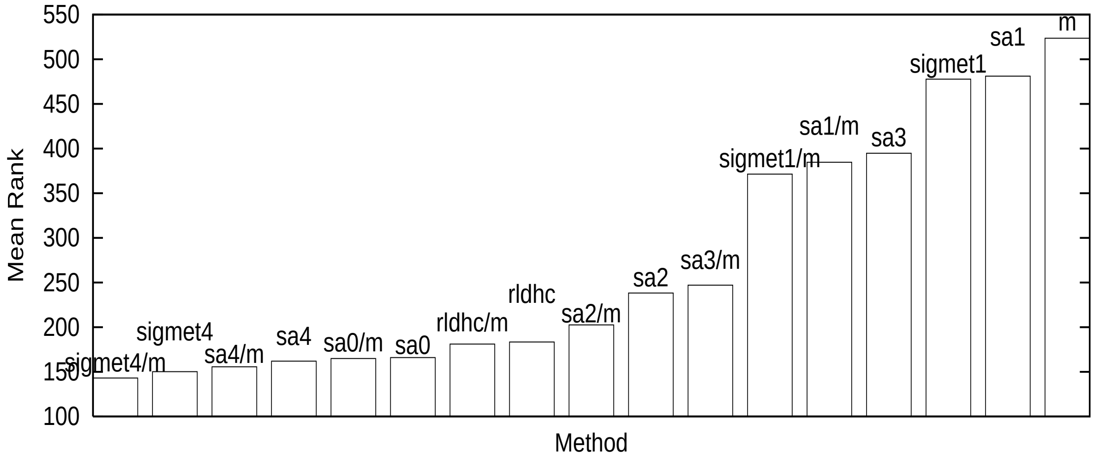

Figure 4.2: Mean Rank By Method — Fall 2000

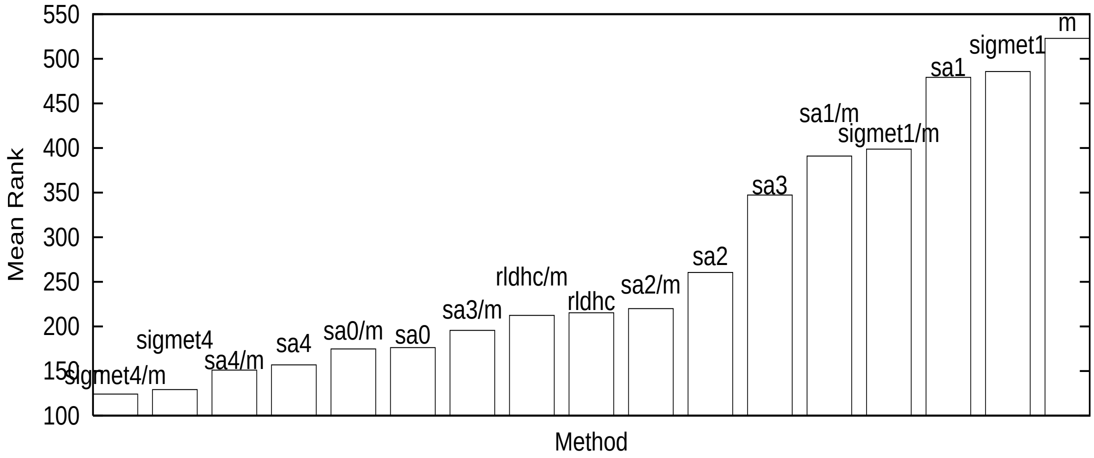

Figure 4.3: Mean Rank By Method — Fall 1999

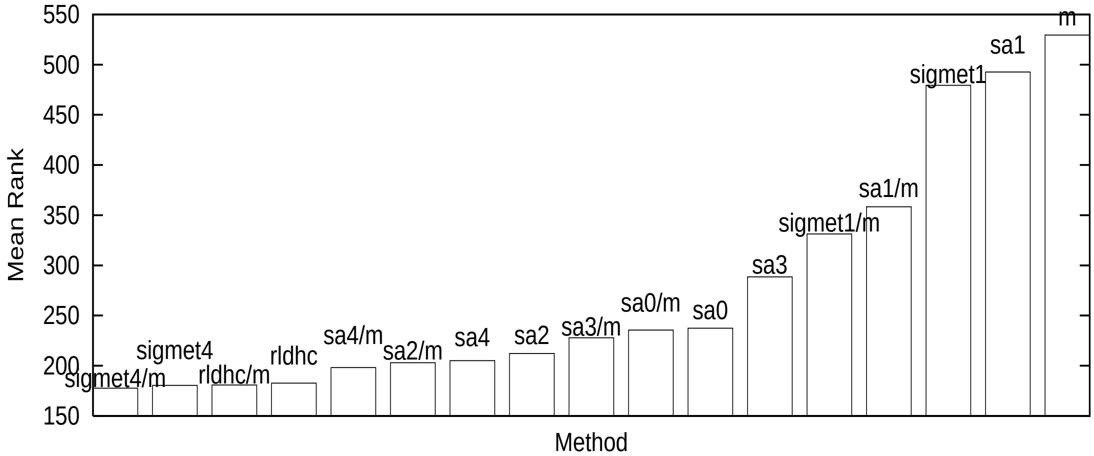

Figure 4.4: Mean Rank By Method — Fall 1998

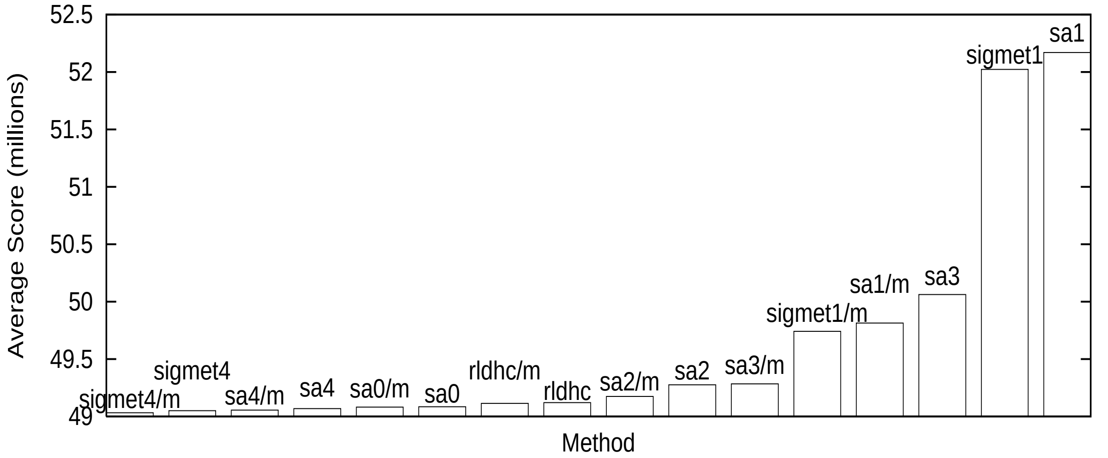

Figure 4.5: Average Score By Method — Fall 2000

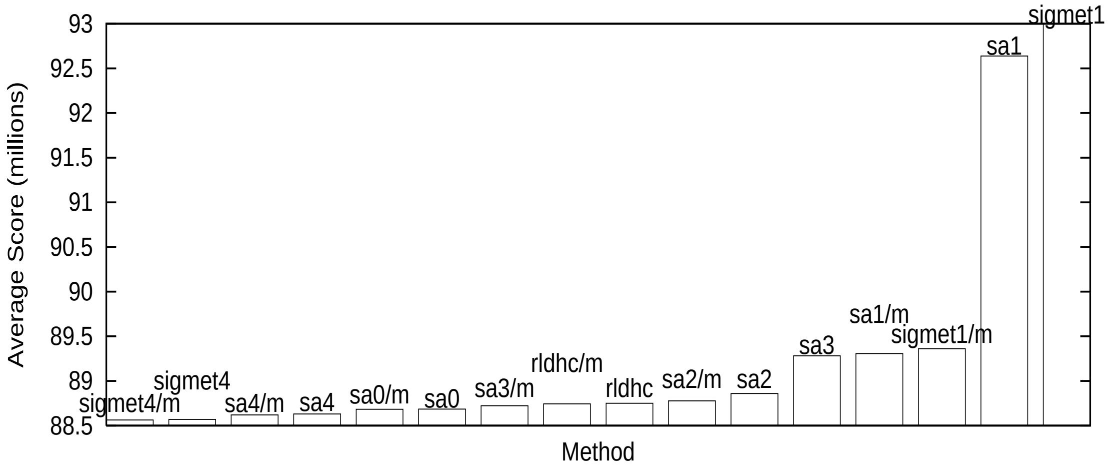

Figure 4.6: Average Score By Method — Fall 1999

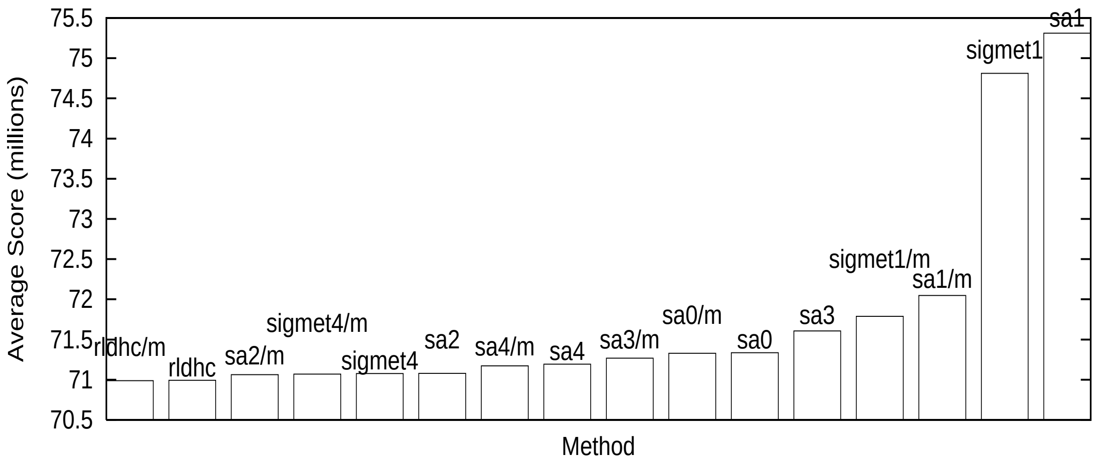

Figure 4.7: Average Score By Method — Fall 1998

Here, in some sense, we are comparing hillclimbing and simulated annealing. To answer this question we use the same 50% lateral move probability, and 40 million hillclimbing attempts followed by a minimization step.

Figures 4.2 through 4.4 show us the answer is 'not necessarily'. For example, hillclimbing (**rldhc**) without minimization was significantly different than **sa1/m** and **sigmet1/m** in every dataset: Hillclimbing can beat some simulated annealing schedules. Furthermore, we cannot say our best ranked simulated annealing schedule was statistically different than these hillclimbing results. In fact, in the Fall 1998 dataset — the only dataset where the mean rank order (Figure 4.4) did not match the average score sorted order (Figure 4.7) — the **rldhc** average score was the lowest.

Sometimes, with a good annealing schedule, we can escape local minimal traps that catch hillclimbers, and arrive closer to the global optimal solution, though how close we cannot say. As we see in the next section, accepting lateral moves is essential to finding a good solution in our domain. With a sufficient number of attempts, random lateral descent hillclimbing (**rldhc**) produced results comparable to simulated annealing.

4.3.5 Importance of Lateral Moves on a Mesa

Does accepting lateral moves impact the search for a minimal solution?

To investigate the significance of lateral moves, we ran three random descent hillclimbing methods, based on 40 million attempts, and varying the probability of accepting lateral moves from never (**rdhc**), to half (**rldhc**), and then to all the time (**rldhc**2). We also ran the Metropolis algorithm with half and all lateral moves accepted based on the table-driven (**sigmet1** & **met1**) and reheating (**sigmet4** & **met4**) cooling schedules.

	%	Fall 2000		Fall 1999		Fall 1998	
Scheme	Ties	Avg Score	Std Dev	Avg Score	Std Dev	Avg Score	Std Dev
rdhc	0	50.077	0.297	89.100	1.863	71.707	0.687
rldhc	50	49.121	0.244	88.750	0.221	70.992	0.544
rldhc2	100	49.076	0.394	88.730	0.229	71.061	0.540
mr	0	667.506	10.266	867.751	8.214	836.634	5.462
m	0	671.013	7.197	863.249	7.405	838.684	5.431
sigmet4	50	49.051	0.235	88.570	0.214	71.077	0.740
met4	100	49.272	0.426	88.726	0.378	71.057	0.525
sigmet1	50	52.022	0.857	92.998	1.015	74.810	1.209
met1	100	52.006	1.121	92.626	0.926	74.619	0.900

Table 4.6: Lateral Test Statistics (in millions)

Across the Fall 2000 and 1999 datasets (Table 4.6), we found random lateral descent hillclimbing, accepting all or half of the ties, differed significantly from random descent hillclimbing (**rdhc**) and iterated next-descent hillclimbing (**m** & **mr**), both never taking lateral moves and producing less desireable results. This suggests our problem domain has 'mesa' characteristics — large regions of unchanging value.

Varying the probabilities of accepting lateral moves with the Metropolis algorithm showed no significant difference for either cooling schedule when tested on all three datasets (Table 4.6). With a mesa landscape, accepting lateral moves is essential to finding better solutions — whether the best probability is 100% or 50% depends upon the dataset.

4.3.6 Varying Probabilities for Better Moves

Should better moves always be accepted?

By avoiding the sigmoid calculation in the case of better and lateral moves, the Metropolis criterion saves some CPU cycles. On the other hand, the symmetric sigmoid algorithm provides a consistent logic independent of the quality of the move.

Using the reheating and the table-driven schemes, as described in Section 4.3.5, we compared the Metropolis criterion to our sigmoid-based approach, both accepting half the lateral moves, searching for significant differences in the final results.

Among the three datasets, based on 32 runs each, we found no significant difference among the reheating schemes, either before or after minimization. The table-driven schemes after minimization were significantly different than before minimization, but there were no reported differences within each group.

We have no statistically sound evidence that varying the probability for better moves differs significantly from the Metropolis criterion. Both approaches result in the same asymptotic distribution, since most considered moves are uphill.

4.3.7 Reaching a Local Minimum

Is the minimization step important to the final result?

The use of iterative descent hillclimbing (**m**) on the results from our random methods not only guaranteed local minima — a marketing advantage — it also improved the results of the methods, often to the point of indistinction.

The following statements regarding the differences between the schemes could only be made before the minimization step took place. With the Fall 1998 data, **sa3** was significantly different than **sigmet1** with a .01 confidence level. In Fall 1999, **sigmet4**, **sa4** and **sa0** were strongly different than **sa3** with .01 confidence, and **sigmet4** differed from **sa2** with .05 confidence. For Fall 2000, **sigmet4**, **sa4** and

rldhc were statistically different than **sa3** with .01 confidence, and **sa2** with a .05 confidence level.

Most of the differences between the schemes and the table-driven schemes remained after minimization. In Fall 2000, all the schemes except **sa3** (i.e. **sigmet4**, **sa4**, **sa0**, **rldhc**, **sa2**) differed from **sa1** with .01 confidence; **sa3** changed from no apparent difference with the table-driven approaches, to strong differences with .01 confidence after minimization. With the Fall 1999 dataset, all the schemes remained strongly different than **sa1** and **sigmet1**. In Fall 1998, all the schemes were initially strongly different than the table-driven schemes, but **sa3/m** and **sa0/m** lost this distinction with **sigmet1/m**. The table-driven approaches were the least volatile to the post-round minimization process, and ranked poorly.

sa3 was most volatile following the minimization step (see page 60). While there was no change in the rank order of the schemes with the Fall 2000 dataset, in Fall 1999 (Figure 4.3), **sa3/m** moved ahead of **rldhc/m** and **sa2/m**, and for Fall 1998 (Figure 4.4), **sa3/m** moved ahead of **sa0/m**.

Disappearing differences among the schemes, after minimization, implies the lowest score before minimization is not necessarily the solution that reaches the lowest result. This realization threw a monkey-wrench into the original idea of performing the minimization step only on the best pre-minimized round. Based on these results, DAO version v1.10 was changed to allow per round minimization at the expense of increased running time.

To detect the possible situation where an overall lower grade occurring during a round might be lost by accepting an uphill move, we added the ability to keep track of the best grades after each accepted improving move during a round, and verify this was no better than the final unminimized result. The option of maintaining this lowest per accept score as the global best grade was added in DAO version v1.15.

Out of eight trials each across all three datasets, we saw this case occur once with **sa3** in the 1998 dataset, and twice with **sa4** in the other two datasets. This small sample provided support both for ignoring and remembering the lowest score ever seen. In one case, minimization after each round lost the lowest result which would have minimized even further, in the other it saved it because the ending grade minimized lower than the within-round lowest grade would have. We did not see the case where the lowest round score before minimization was less than the minimized ending score.

The minimization step plays an important role in that it guarantees a local minimum solution. The amount of work performed by the minimization step varies. One might have guessed that a long minimization step implies a poor scheme, but the results show otherwise. For example, **sa3** typically takes many iterations to reach a local minimum, while **sa0** often finds a local minimum by itself, yet both achieve comparable scores.

4.3.8 Comparing the Schemes: Scores and Attempts

So which scheme works best?

Without stronger statistics, score alone is too highly data-dependent to yield a good characterization of scheme quality. If not score, then perhaps rate of improvement could be useful. On average, **sa1** was the most efficient approach, producing the most improvement per attempt. It was also the fastest, and as it turns out, consistently the worst performer, on all three datasets.

Turning to the 'Score vs Total Attempts' scatter plots (Figures 4.8 through 4.10) for assistance, ideally we would like to see data in the bottom left-hand corner of the graph and few results in the upper right. Instead, we see **sa1** produced some of the

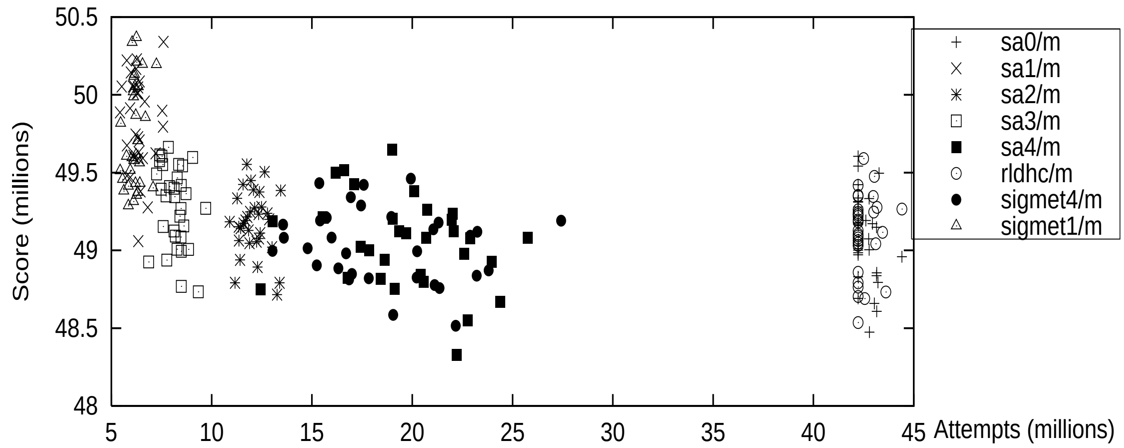

Figure 4.8: Score vs Total Attempts by Scheme — Fall 2000

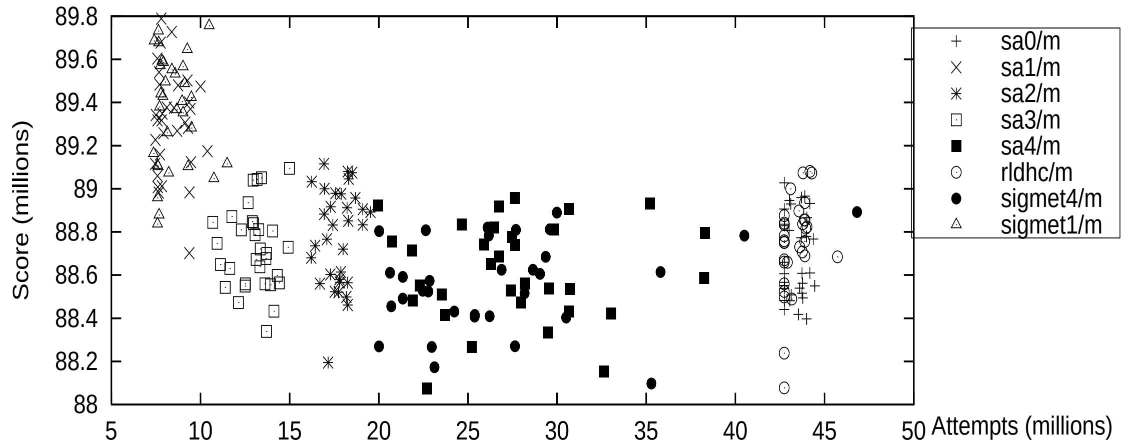

Figure 4.9: Score vs Total Attempts by Scheme — Fall 1999

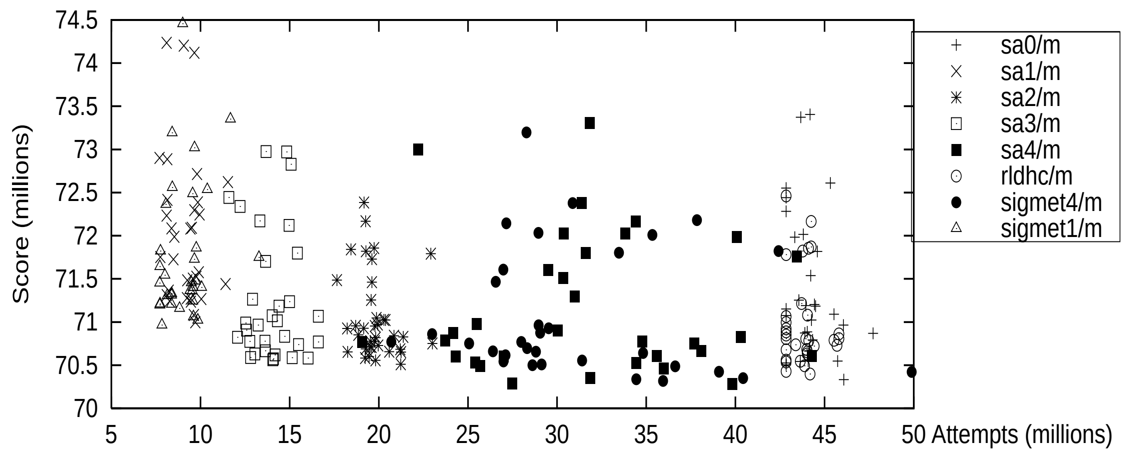

Figure 4.10: Score vs Total Attempts by Scheme — Fall 1998

highest scores, while the results of the other schemes are in the same score range. A few of the **sa4** rounds came out the lowest.

In terms of attempts, we see **sa0** and **rldhc**[8] take the most attempts, **sa1** and **sa3** the least, leaving **sa2** and **sa4** in the middle.[9] Furthermore, **sa4** had the widest variability in total attempts, while **sa2** and **sa3** were more predictable. Overall, the dynamic adaptive schedules gave us the best performance.

With so much variability in scheme performance, and sensitivity to changes in the objective function and the data, rather than trying to pick one, we use all the schemes in rotation, and multiple runs (as time permits), to sample the different approaches for the best results.

[8]The random lateral-descent hillclimbing took as long as the algorithmic scheme intentionally to give it the fairest opportunity to find a good configuration. As hillclimbing was not our focus, we offer no data regarding its performance in fewer attempts.

[9]The Metropolis variations of the table-driven and reheating schemes behaved similarly to our simulated annealing versions in number of attempts.

Chapter 5

Conclusions

Results in the previous chapter show that simulated annealing can effectively match students and dorm rooms according to student preferences and the policy of the dorm system. DAO's randomized global approach produces faster, better matches.

While DAO version v1.15 is undeniably a good beginning, looking ahead to its fourth consecutive semester of service, we conclude with some ideas for future enhancements, from the nearer-term to the more speculative:

Student Designated Priorities Naming their top three preferences, students could designate their own priorities, instead of relying on the current one-size-fits-all approach. This would eliminate staff having to guess between hall, roommate and room type on a case-by-case basis, and likely to lead to much happier students and staff.

New Hall A new apartment-style dorm, scheduled for occupancy Fall 2001, needs to be added to the DORMS.INI configuration file following the pattern of the Student Residence Center buildings. A two or three letter acronym for the building will be elected and added to the HMS database, but no code changes in the DAO are expected. The application form must be reprinted with the new hall as a choice, providing an opportunity to make other improvements to the form.

New Room Type According to Area Coordinator Jenna Sultemeier, demand was so overwhelming for the larger rooms, to accommodate the growing number of new university students having never shared a room with a sibling at home, a new *Super Single* room type may be offered Fall 2001.[1] Designating a new room type for these rooms in the database, and modifying the application form to reflect the new choice, can implement this change without modifying the code.

Online Applications Web-based residence forms could relieve some of the data entry burden as well as standardize the data. As borne out by a previous class group project, such an undertaking would also raise new issues, such as security, duplicate applications, and credit card payments. Again, interfacing with the existing legacy system continues to constrain new development.

Tighter Coupling Eliminating the exporting process from HMS to the DAO by accessing the database directly could provide a more seamless integration of our technology with the entire housing management system. Memory resident data structures would still be created for an efficient main annealing loop. Without an API from CBORD, however, this approach is difficult to implement and maintain.

Formatted Reports Outputting the reports in HTML, for example, would make the information easier to read with a browser. This idea also applies to the user documentation.

[1] Sultemeier, J. Personal communication, September 12, 2000.

Two-Move Local Minimum The DAO currently guarantees that within one move, there is no better place for anyone, but it makes no promise about two moves. Though expensive in runtime, all pairs of moves could be checked. The current engine would require new data structures to maintain a move history to restore state between attempts.

Parallel Processing If DAO run times became excessive, parallel processing could be used to gather results of individual rounds in less overall elapsed time, allowing the best of the best to rise to the top more quickly. Initially configuration parameters for constraint values, dorm definitions, blocked beds, annealing options and data files would be distributed. On a per round basis, each slave process reports its total score whenever available, and the master checks if it is a new record score, and signals the slave to save the configuration if so, otherwise the master signals another round or the end. Since slave processes do not have to synchronize with each other, the varying times of the schemes are less of a problem.

Realtime Visualization To visualize the assignment process in the GUI, accepted moves could appear in realtime as blinking beds with graphs indicating the overall grade, temperature, and other statistics, changing over time. Of course the impact of screen redrawing would dramatically slow the optimized main inner loop, but it could be valuable to help users better understand the processing.

Judgment and Reasoning Based on the concepts in [AB83], wouldn't it be nice, if you could just ask the DAO to explain itself? Answering the question, "why did you do that instead of this?" reduces the explanation task to one of accounting for the differences between a pair of moves instead of a 'verbose' enumeration of all possible bed assignments. Preferably only the relevant data would be presented. Using judgment to form an interpretation of the environment with respect to a goal, say a group request, an analysis could explain why the roommates did not get together. Using some form of reasoning, it could propose avenues to make a move happen. Given the ability to recognize significant differences in the value of concepts, it might call our attention to empty bed situations, a gender inequity, or a smoking stalemate. Currently this is all done by two people using the GUI and DAO's extensive reports.

Further Generalization Using this system at another university to do the same function would require analysis of and probably modifications to the objective function, input formats, report designs and other customizations. Implementing an objective function definition language for the DAO, rather than necessarily requiring low-level reprogramming in C, would facilitate such technology transfer. A more generic rewrite of our existing SA engine as a member of a search library with interface requirements for the objective function, might also be useful in applying global optimization techniques to other problem areas.

More improvements could certainly be added, the DAO is still a work in progress.

Appendices

A HMS Rules for Assignment with the DAO **74**

B The DAO Objective Function **75**

Appendix A

HMS Rules for Assignment with the DAO

HMS assignment expressions to place a student in the room being tried only if it matches the DAO's *suggestion* imported into the *Force Occ* attribute field:

```
    MATCH FORCE OCC +262144

alltrim(rooms.bldg_id)+alltrim(rooms.room_no) ==
alltrim(getattrib(applicant.id+gcterm,"Force Occ"))

    NOMATCH FORCE OCC -999

alltrim(rooms.bldg_id)+alltrim(rooms.room_no) <>
alltrim(getattrib(applicant.id+gcterm,"Force Occ"))
```

HMS will not place a student in the suggested room if someone is already there. If HMS has roommate matches to make, these rules may not work as expected.

Appendix B

The DAO Objective Function

The Objective Function[1], f, embodies all the contraints used to match students and dorm room beds. The domain is all the students in their beds.

The details of the two types of errors: *Room-student* and *student-consensus*, are as follows:[2]

[1]This description corresponds to the last time it was modified in DAO version v1.12, when the *different_bed* constraint was added.

[2]Default values are zero unless specified (this policy was adopted in version v0.84).

1. ROOM-STUDENT RELATED CONSTRAINTS: **For each student:**

NAME	DESCRIPTION	SCORE
Bed	not in a bed	1 * ERR_NOBED
Gender[1]	male in a female room, or female in a male room	1 * ERR_WRONG_SEX
Smoke	non-smoker in a smoking room, or smoker in a non-smoking room; objector in a smoking room, or objector in a smoking-either room	1 * ERR_WRONG_SMOKE * SMOKE_OBJ_WEIGHT
Dorm Prefs[2]	specific room/bed, building, or better bed request not granted Special Living Option (SLO) 1^{st} Hall choice not granted 2^{nd} Hall choice not granted 3^{rd} Hall choice not granted Room-type not granted	w * ERR_SPECIFIC_ROOM[4] w * ERR_SPECIAL_LIVING[4] w * ERR_DORM_CHOICE1[4] w * ERR_DORM_CHOICE2[4] w * ERR_DORM_CHOICE3[4] w * ERR_ROOM_PREF[4]
Underage	the dorm has age restrictions[5] and the student's age is less than UNDER_THIS_AGE, or missing uses MIN_AGE_DEFAULT	# years under age * ERR_UNDERAGE
New Resident	the dorm has new restrictions[5] and the student has been a resident NEW_RESIDENCY_SEMESTERS	(NEW_RESIDENCY_SEMESTERS - student[ncs] + 1) * ERR_NEW_RESIDENT[3]
Un-requested SLO	the dorm has special living options but the student did not request a slo[6], or dorm has special living options different than the student's requested slo[6]	1 * ERR_UNREQUESTED_SLO
Better Bed	student is not in a better bed [7]	student[ncs] * ERR_BETTER_BED[3]
Different Hall	student is not in the first hall, as initialized by ASSIGN.LST[8]	1 * ERR_DIFFERENT_HALL
Different Bed	student is in the same first hall, but a different bed, as initialized by ASSIGN.LST[8]	1 * ERR_DIFFERENT_BED
Worse Hall	student is not in the first hall or better hall according to their prefs, as initialized by ASSIGN.LST[8]	1 * ERR_WORSE_HALL

2. STUDENT-CONSENSUS CONSTRAINTS: **For each room:**

NAME	DESCRIPTION	SCORE
Gender[9]+	not completely one gender	# students * min[$gender$] * ERR_MIXED_SEX
Smoke[9]+	not completely smokers or non-smokers (smokers penalized); if any smoke objector is present	# smokers * ERR_MIXED_SMOKE * SMOKE_OBJ_WEIGHT
Music	for each student's objection clashing with another student's preference (both penalized)	2 * # clashes * ERR_MIXED_MUSIC
Study Late	not completely late studiers, or not, lateness specifiers suffer	# late studiers * min[$lateyes/no$] * ERR_MIXED_STUDY_LATE
Age Difference	for each student in the room with a dob, max[Δage] between everyone unacceptable	each student's max[Δage] > ACCEPT_AGE_DIFF * ERR_AGE_DIFF
Group[9]+	for each student in the room without requested roommates[10]	w * # requested mates not present * ERR_GROUP_INCOMPLETE[4]
Suite	each adjoining room's Group, Gender, and Smoke constraints are re-evaluated upon a move for the intermediate global grade calculations	
Partial Fill [11]	room is partially filled or empty, not including blocks	min[$emptybeds, filledbeds$] * # students * ERR_PARTIAL_FILL
Triple Fill	double room is packed with three students	# students * ERR_TRIPLE_FILL
First Timer	room has first time residents and overriding renewals	# first-time students * (old[ncs]$-$firsts[ncs])$*$ ERR_MIXED_FIRSTTIMER[3]

NOTES:

1. Gender became a 'hard constraint' in DAO version v0.93, guaranteeing the randomly-chosen bed will always be the appropriate gender, without requiring a large constraint value to ensure it.

2. Each student is allowed three hall choices, one special living option, one specific room request, and a room type.

3. Where *ncs* is the number of consecutive semesters of dorm residency.

4. Where w is the weight based on the seniority (the number of concurrent semesters residing in the dorms) times SENIORITY_WEIGHT, plus any lottery benefit, l.

$$w = (\text{senior} * \text{SENIORITY\_WEIGHT}) + l$$

   Lottery benefit is based on the LOTTERY_BENEFIT: The tenths of a semester's seniority weight the first lottery number is worth. The last lottery number is worth no extra priority. The Lottery number in the middle is worth half as much as the first, the rest are distributed evenly in between. Lottery numbers are tracked separately for new and returning students. To maintain distinct seniority levels, the LOTTERY_BENEFIT should be less than one full semester's weight (9 at most). The lower this benefit, the better the overall assignments will be since first-come order is just random noise to good matches.

5. The DORMS.INI configuration file designates the dorms with new resident and underage restrictions.

6. For non-smokers, the DVNS special living option is a big exception because it refers to the entire non-smoking building of DeVargas (version v1.00).

7. The DORMS.INI configuration file designates the dorms with better beds and the beds that are better.

8. The ASSIGN.LST input file contains students' current assignments in HMS.

9. Applies to rooms where the specific designation is *either* (not including virtuals). The + indicates the constraint applies across adjoining suites.

10. A *bestfriend* is the first roommate requested when a double room is also requested. A bestfriend must be present in the same room, an adjoining room is not sufficient.

11. Minimizes the number of partially filled rooms and reduces the consolidation process. This error compensates for the smaller music, age or other penalties, but if too large, can disregard hall choices.

References

[AB83] David H. Ackley and Hans J. Berliner. The QBKG System: Knowledge Representation for Producing and Explaining Judgments. Technical Report CMU-CS-83-116, Carnegie-Mellon University, 1983.

[Ack87a] David H. Ackley. An Empirical Study of Bit Vector Function Optimization. In Lawrence Davis, editor, *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann Publishers, 1987.

[Ack87b] David H. Ackley. *Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers, Boston, MA, 1987.

[AKvL97] E.H. Aarts, J. Korst, and P.J. van Laarhoven. Simulated Annealing. In E. H. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 4. John Wiley and Sons, 1997.

[Ame] Ameritherm, Inc. *Annealing*.
On web at http://www.ameritherm.com/annlovrv.html. Accessed October 31, 2000.

[BGNZ96] H.E. Bomeijn, D.L. Graesser, S. Neogi, and Z.B. Zabinsky. Simulated Annealing for Mixed Integer/Continuous Global Optimization, 1996. On web at http://ideas.uqam.ca/ideas/data/Papers/fthtinber96-38.html. Accessed October 31, 2000.

[Col94] The Concise Columbia Electronic Encyclopedia, Third Edition. *Annealing*, 1994. On web at http://www.encyclopedia.com. Accessed October 31, 2000.

[CSE96] Computational Science Education Project CSEP. *Mathematical Optimization*, 1991, 1992, 1993, 1994, 1995, 1996. On web at http://csep1.phy.ornl.gov/CSEP/MO/NODE28.html. Accessed October 31, 2000.

[DM84] K. Roscoe Davis and Patrick G. Mc Keown. *Quantitative Models for Management*. Kent Publishing Company, Boston, Massachusetts, 1984.

[DS00] Jack Dongarra and Francis Sullivan. The Top 10 Algorithms. *Computing in Science & Engineering*, 2(1), Jan/Feb 2000.

[EFC98] Saleh Elmohamed, Geoffrey Fox, and Paul Coddington. A Comparison of Annealing Techniques for Academic Course Scheduling. In *Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling*, pages 146–166, Syracuse, NY, USA, Apr. 4 1998. Practice and Theory of Automated Timetabling.

[Enc00a] Microsoft Encarta Online Encyclopedia 2000. *Metallography*, 1997 - 2000. On web at http://encarta.msn.com. Microsoft Corp. All Rights Reserved. Accessed October 31, 2000.

[Enc00b] Microsoft Encarta Online Encyclopedia 2000. *Metalwork*, 1997 - 2000. On web at http://encarta.msn.com. Microsoft Corp. All Rights Reserved. Accessed October 31, 2000.

[Gib76] Jean Dickinson Gibbons. *Nonparametric Methods for Quantitative Analysis (Second Edition)*. American Sciences Press, Inc, Columbus, Ohio, 1976. Republished in 1985.

[HTdW97] Alain Hertz, Eric Taillard, and Dominique de Werra. Tabu Search. In E. H. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 5. John Wiley and Sons, 1997.

[JAM89] David S. Johnson, Cecilia R. Aragon, and Lyle A. Mc Geoch. Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning. *Operations Research*, 37(6):865–892, Nov-Dec 1989.

[JAM91] David S. Johnson, Cecilia R. Aragon, and Lyle A. Mc Geoch. Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. *Operations Research*, 39(3):378–406, May-June 1991.

[KGV82] S. Kirkpatrick, C. Gelatt Jr., and M. Vecchi. Optimization by Simulated Annealing. Technical Report RC9355, IBM Research Report, 1982.

[KGV83] S. Kirkpatrick, C. Gelatt Jr., and M. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):498–516, May 1983.

[Lee94] F.H. Allisen Lee. Parallel Simulated Annealing on a Message-Passing Multi-Computer, 1994.

[Mac99] D.J.C. MacKay. Introduction to Monte Carlo Methods. In Michael I. Jordan, editor, *Learning in Graphical Models*. MIT Press, 1999.

[MRR+53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21(6), June 1953.

[MS91] B.M.E. Moret and H.D. Shapiro. *Algorithms from P to NP, Volume I.* Benjamin/Cummings Publishing Company, Redwood City, CA, 1991.

[Pic87] M. Piccioni. Combined Multistart-Annealing Algorithm for Continuous Global Optimization, 1987.
On web at http://www.isr.umd.edu/TechReports/ISR/1987/TR_87-45/TR_87-45.phtml. Accessed October 31, 2000.

[RG86] R.E. Randelman and G.S. Grest. N-City Traveling Salesman Problem - Optimization by Simulated Annealings. *J. of Stat. Phys.*, 45:885–890, 1986.

[Rib] Rita Almeida Ribeiro. *Fuzzy Mathematical Programming.* Presented at UNM EECE Seminar. May 15, 2000. Albuquerque, NM.

[Sha] Bassan Z. Shakhashiri. *Science is Fun in the Lab of Shakhashiri.* University of Wisconsin-Madison Chemistry.
On web at http://www.scifun.chem.wisc.edu/WOP/RandomWalk.html. Accessed October 31, 2000.

[vLA87] P.M.J. van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications.* Dordrecht Reidel Publishing Company, Dordrecht, Holland, 1987. Republished in 1989 by Kluwer Academic.

[Zar99] Jerrold H. Zar. *Biostatistical Analysis – Fourth Edition.* Prentice Hall, Upper Saddle River, New Jersey, 1999.

the changing red heat
anneals to a cool calm stay —
local minimum