MORTGAGE POOL ALLOCATION BY SIMULATED ANNEALING

Eugene Pinsky
Computer Science Department
Boston University

Paul Fahn and Yechiam Yemini
Center for Advanced Technology
Columbia University

CUCS-052-90

ABSTRACT

Many optimization problems in finance are known to be computationally complex. In this paper we consider one such problem of Mortgage Pool Allocation. In this problem one needs to allocate a set of mortgage-backed securities to fill forward contracts in real-time. The number of securities and contracts is typically large and there are numerous constraints on allocations in the form of rules published by the Public Securities Association (PSA). The problem is to find the most profitable allocation that satisfies all of the PSA's constraints. Simulated annealing is a recently explored probabilistic algorithm, used most commonly to solve simply-formulated combinatorial optimization problems with a small number of constraints. This paper shows successful application of simulated annealing to the problem of mortgage pool allocation: it finds good (highly profitable) solutions very quickly. This result is significant both for the complexity of the problem solved and because it demonstrates the feasibility of simulated annealing for solving real-world financial problems.

Research supported in part by the NY State CAT contract #NYSSTFCU01207901

æ

MORTGAGE POOL ALLOCATION BY SIMULATED ANNEALING

Eugene Pinsky
Computer Science Department
Boston University

Paul Fahn and Yechiam Yemini
Center for Advanced Technology
Columbia University

ABSTRACT

Many optimization problems in finance are known to be computationally complex. In this paper we consider one such problem of Mortgage Pool Allocation. In this problem one needs to allocate a set of mortgage-backed securities to fill forward contracts in real-time. The number of securities and contracts is typically large and there are numerous constraints on allocations in the form of rules published by the Public Securities Association (PSA). The problem is to find the most profitable allocation that satisfies all of the PSA's constraints. Simulated annealing is a recently explored probabilistic algorithm, used most commonly to solve simply-formulated combinatorial optimization problems with a small number of constraints. This paper shows successful application of simulated annealing to the problem of mortgage pool allocation: it finds good (highly profitable) solutions very quickly. This result is significant both for the complexity of the problems solved and because it demonstrates the feasibility of simulated annealing for solving real-world financial problems.

Research supported in part by the NY State CAT contract #NYSSTFCU01207901

1. INTRODUCTION: THE MORTGAGE POOL ALLOCATION PROBLEM

In this paper we consider some of the mathematical problems arising in the trading of "*Mortgage Pass-Through Securities*". These securities are created when mortgages are pooled together and undivided interests in the pool are sold ([19]). These mortgage-backed "Pass-Through Securities" are issued primarily by the Government National Mortgage Association (GNMA), Federal Home Loan Mortgage Corporation (FHLMC), and Federal National Mortgage Association (FNMA).

Traders buy and sell these mortgage-backed securities and forwards/futures. A contract is an agreement between two parties for the sale and delivery of a "pool" or a set of mortgage pools of agreed upon characteristics. In a typical transaction, pool information may not be known at the time of the trade. This is called a TBA (to be announced transaction) in contrast to a trade in which the pool is specified. For example, one may purchase $1 million GNMA securities and receive up to three pools, whose numbers will be announced shortly before the settlement. These mortgage pools are assumed to be either in the inventory, or obtainable through the trading desk.

Contracts have agreed settlement dates. Contracts must be filled on the settlement date so as not to incur the finance costs for "holding" the contract until it is delivered (failure to deliver a security on settlement day is therefore costly to the seller and should be avoided). A contract is filled by mortgage pools that satisfy the requirements of that particular contract, but typically they are of the lowest possible grade in all unspecified categories. To fill a "sell" contract, specific pools of the securities and quantities of these pools must be identified and *allocated*. There is some flexibility in the number of pools and quantity of each pool used to satisfy a sell contract. There is also a flexibility of $\pm 2.499999\%$ variance in the total value of the contract when it is delivered. Rules published by Public Securities Association (PSA) define how the contracts are to be delivered. The assignment of pools by the seller is called an *allocation*.

Most contracts are forward contracts and are allocated on *pool day*. There is one day per month for each type of security. The allocators maintain an inventory of securities from which sell orders may be satisfied. However, only a small amount of pool information is available at the beginning of a pool day and thus only a small number of *sell* contracts can be pre-allocated. As pre-allocation information is communicated among the traders, additional contracts may be allocated from the incoming *buy* contracts. Moreover, it is financially advantageous to postpone the allocation process to as short a time as possible before the pool day deadline. As a result, most of the allocations occur in a rush at the

end of the pool day. The sheer volume of these transactions may prevent the allocator from performing a detailed analysis, resulting in sub-optimal allocations. Moreover, even though it may be advantageous to re-allocate pools during the process due to changing market conditions, the cost of doing this manually may be too high. The problem is then to **design efficient computer algorithms for mortgage pool allocation**.

2. PRELIMINARIES, NOTATION AND THE PSA RULES

In this section we would like to describe the structure of the pools, contracts and PSA rules in more detail and introduce some notation.

A (mortgage) pool is a collection of mortgages assembled by an originator as the basis for a security ([19]). Pools are identified by a number. The original principal amount of a pool as of its issue date is known as its *Par Value* or *Original Face Value*. The outstanding principal balance of the underlying mortgages is known as the *Current Face Value*. This value is computed by multiplying the *Par Value* by a *Conversion Factor*. This *Conversion Factor* parameter is quasi-static: the Bond Buyer newsletter publishes the "Monthly Factor Report" which contains a list of factors for GNMA, FNMA and FHLMC securities. The *Coupon Rate* of a pool is the stated annual percentage rate of interest paid on the underlying mortgages. There are a number of other characteristics such as issue and maturity dates, but we will not consider them here.

Let P_i denote the number of pool i, and let Par_Value(P_i), Cur_Value(P_i), Factor(P_i) and Coupon_Rate(P_i) denote its *Par Value*, *Current Face Value*, *Conversion Factor* and *Coupon Rate* respectively. For example, suppose the i-th pool in the inventory is pool A which is a 10% GNMA pool with the original face value of \$800,000.00 and the factor of 0.99. For this pool $P_i = A$, Coupon_Rate(P_i) = 10.00, Par_Value(P_i) = \$800,000, Factor(P_i) = 0.99, Cur_Value(P_i) = Par_Value(P_i) × Factor(P_i) = \$792,000. Finally, as pools are split, some of them may become too small. Very small pools (< \$25,000) cannot be further used and, therefore, such "pool fragments" should be avoided. They typically cannot be included in deliveries, become a non-liquid asset, and are written off as a loss.

The basic unit of trading is a \$1,000,000 contract. The PSA rules specify the pool composition of the contracts in terms of "millions" and contract fails refer to these "millions". For example, consider a contract whose amount is \$5,550,000. For a legal allocation, we need to allocate each \$1,000,000 contract separately and specify the pools for the remainder \$550,000 contract. The quantity less than \$1,000,000 is called an *odd lot*. A legally

allocated $1,000,000 contract is called a *good million*. Therefore, for this example we need to create five good millions and allocate one odd lot of $550,000. In view of the above, without loss of generality, we assume that all contracts have been split so that we only have contracts whose amount is either a million or a fraction of it. We assume that all such contracts are identified by a number. Let C_j denote the number of contract j and let Market_Price denote the current market price of the contract.

We will find it convenient to introduce the following notation:

Amount(C_j)	nominal amount of contract j
Price(C_j)	agreed-upon price of contract j
Market_Price(C_j)	current market price of contract j
Desired(C_j)	desired amount to be delivered on contract j

As an example, consider a $1,000,000 sell contract C for 10% GNMAs with agreed-upon price of $950,000 (i.e. Amount($C$) = $1,000,000, Price($C$) = $950,000. Suppose that on the settlement day, the market price of this security is $940,000 (Market_Price($C$) = $940,000). In this case, we would like to deliver as much of the security as possible at the old price. This is called a "High Tail Contract", because we want deliver more than the specified amount. Following the PSA guidelines we would optimally deliver a maximum of $1,024,999.0 and thus the profit would be $24,999.99 $\times$ (0.95 − 0.94). On the other hand, suppose that the market price on the settlement day is Market_Price(C) = $980,000. In this case, we would like to deliver as little of the security as possible at the old price. This is called a "Low Tail Contract", because we want to deliver less than the specified amount. Following the PSA guidelines we would optimally deliver $975,000.01 and thus minimize a "loss" of $24,999.99 $\times$ (0.98 − 0.95). The "desired" amount is computed by the following rules:

$$\text{Desired}(C_j) = \begin{cases} \text{Amount}(C_j) & \text{if Price}(C_j) = \text{Market\_Price} \\ 1.02499999 \times \text{Amount}(C_j) & \text{if Price}(C_j) > \text{Market\_Price} \\ 0.97500001 \times \text{Amount}(C_j) & \text{if Price}(C_j) < \text{Market\_Price} \end{cases}$$

The key PSA rules can be summarized as follows:

(1) Pools must have identical coupons and be issued by the same agency.

(2) Sellers may deliver up to $\pm 2.499999\%$ of the value of a contract

(3) For coupons greater than or equal to 12% a maximum of four pools may be used to deliver per $1,000.000. A maximum of three pools may be used to deliver for an odd lot

C_j if \$500, 000 < Amount(C $_j$) < \$1, 000, 000, and a maximum of two pools for an odd lot C_j if 0 < Amount(C $_j$) ≤ \$500, 000.

(4) For coupons less than 12% a maximum of three pools may be used to delivered per \$1, 000. 000. A maximum of two pools may be used to deliver for an odd lot C_j if \$500, 000 < Amount(C $_j$) < \$1, 000, 000, and exactly one pool for an odd lot C_j if 0 < Amount(C $_j$) ≤ \$500, 000.

(5) For contracts filled with more than one pool, no proper subset of those pools may fall within the allowable variance of the contract.

We define Max_Pools_Allowed(C $_j$) to be the maximum number of pools that can be used to satisfy contract C_j. Rules 3 and 4 above define Max_Pools_Allowed(C $_j$) as follows:

$$\text{Max\_Pools\_Allowed}(C_j) = \begin{cases} 1 & \text{if } 0 < \text{Amount}(C_j) \le \$500,000.00 \text{ and Coupon\_Rate} < 12\% \\ 2 & \text{if } \$500,000.00 < \text{Amount}(C_j) < \$1,000,000.00 \\ & \qquad \text{and Coupon\_Rate} < 12\% \\ 3 & \text{if Amount}(C_j) = \$1,000,000.00 \text{ and Coupon\_Rate} < 12\% \\ 2 & \text{if } 0 < \text{Amount}(C_j) \le \$500,000.00 \text{ and Coupon\_Rate} \ge 12\% \\ 3 & \text{if } \$500,000.00 < \text{Amount}(C_j) < \$1,000,000.00 \\ & \qquad \text{and Coupon\_Rate} \ge 12\% \\ 4 & \text{if Amount}(C_j) = \$1,000,000.00 \text{ and Coupon\_Rate} \ge 12\% \end{cases}$$

The PSA rules for GNMA, FHLMC, FNMA are similar (such as rules 1-4). However, at the present time there are additional constraints for the GNMA pools. In splitting GNMA pools one has to split an amount which is a multiple of \$5, 000 in the original face. However, when a pool is created (it is just a polling of the existing mortgages), its face value may not be an exact multiple of \$5, 000. In such a case, the pool has a *tail*: amount which is a fraction of \$5, 000. For any amount X, we can easily define

$$\text{Tail}(X) = X - 5,000.00 \times \lfloor X/5,000.00 \rfloor$$

For example, suppose pool B is created with original face value of \$658, 000. This pool has a tail of Tail(Par_Value(A)) = \$3, 000. The remaining amount (\$655, 000) is a multiple of \$5, 000. If a pool is created with a tail, this tail exists for the lifetime of a pool and simply gets shunted from one broker to another. This tail cannot be split and no more than one tail can be used for any contract. Since for FNMA and FHLMC pools we do not have constraints on the tails, their allocation is easier than that of GNMA pools. We will concentrate, therefore, on the mortgage pool allocation for GNMA securities.

To illustrate these rules and constraints, let us consider a few examples. Assume that we are allocating 10% GNMA mortgage pools to satisfy contract C_j. Assume that we have the following 3 pools in the inventory:

Pool	Original Face	Tail	Factor	Current Face
A	\$800,000.00	\$0.00	0.99	\$792,000.00
B	\$658,000.00	\$3,000.00	0.98	\$644,840.00
C	\$525,480.30	\$480.30	0.97	\$509,715.89

Example 1: Suppose Amount(C_j) = \$500,000.00 and Desired(C_j) = Amount(C_j). For this case Max_Pools_Allowed(C_j) = 1. Consider the following allocation:

Pool	Amount Delivered in Original Face	Amount Delivered in Current Face	
A	\$505,000.00	\$499,950.00	
		\$499,950.00	delivered

This constitutes a good delivery for this odd lot, since only one pool was used (rule #4 is satisfied), the delivered amount (\$499,950.00) is within 2.499999% of the contract amount (\$500,000.00) (rule #2 is satisfied), no small pool fragments are left, since \$295,000 remains in pool A, and no tails were used and/or split.

One could be tempted to try the following allocation:

Pool	Amount Delivered in Original Face	Amount Delivered in Current Face	
C	\$515,480.30	\$500,015.89	
		\$500,015.89	delivered

This allocation gives a closer value to the desired amount (\$500,015.89 instead of \$499,950.00) than the previous one. However, it leaves pool C with a fragment of \$10,000. Such an allocation should be avoided.

On the other hand, the following allocation for the same contract

Pool	Amount Delivered in Original Face	Amount Delivered in Current Face	
A	\$490,000.00	\$485,100.00	
		\$476,713.02	delivered

is illegal because the delivered amount \$476, 713. 02 is not within 2. 499999% of the contract amount. In other words, rule #2 is violated.

Example 2: Suppose Amount(C_j) = \$750, 000. 00. and Desired(C_j) = 1. 02499999 × Amount(C_j) = \$768, 749. 93. For this case Max_Pools_Allowed(C_j) = 2. Consider the following allocation:

Pool	Amount Delivered in Original Face	Amount Delivered in Current Face	
A	\$500. 000. 00	\$495, 000. 00	
B	\$275, 000. 00	\$269, 500. 00	
		\$764, 500. 00	delivered

This constitutes a good delivery for this odd lot, since only two pools were used (rule #4 is satisfied), the delivered amount (\$764, 500. 00) is within 2. 499999% of the contract amount (\$750, 000. 00 - rule #2 is satisfied), no pool fragments were created and the original face of the amounts used is a multiple of \$5, 000. 00. On the other hand, the following allocation for the same contract

Pool	Amount Delivered in Original Face	Amount Delivered in Current Face	
A	\$270, 000. 00	\$267, 300. 00	
B	\$245, 000. 00	\$240, 100. 00	
C	\$265, 300. 00	\$257, 341. 00	
		\$764, 741. 00	delivered

is illegal. Although the delivered amount \$764, 741. 00 is within 2. 499999% of the contract amount (it is in fact very close to the desired amount), more than two pools were used (rule #4 is violated). Moreover, the original tail \$480. 30 for pool *C* is split.

Example 3: Suppose Amount(C_j) = \$1, 000, 000. 00. and Desired(C_j) = 0. 97500001 × Amount(C_j) = \$975, 000. 01. For this case Max_Pools_Allowed(C_j) = 3. Consider the following allocation:

Pool	Amount Delivered in Original Face	Amount Delivered in Current Face
A	\$620, 000. 00	\$613, 800. 00

B	\$203, 000. 00	\$198, 940. 00
C	\$170, 000. 00	\$164, 900. 00
		\$977, 640. 00 delivered

This is an example of a "good million": the allocation satisfies the PSA rules, no pool fragments are generated and, in addition, only one tail (from pool A) is used. On the other hand, the following allocation

Pool	Amount Delivered in Original Face	Amount Delivered in Current Face
B	\$495, 500. 00	\$485, 590. 00
C	\$505, 480. 30	\$490, 315. 89
		\$975, 905. 89 delivered

is illegal. It is illegal because more than one tail is used (i.e. more than one pool is used with the original face amount not a multiple of \$5, 000) and the tail for pool B has been split. Note that in this example, we would like to deliver as little as possible (optimally, just \$975, 000. 01) and therefore, the second allocation would have been better. However, the constraints on the tails prevent us to deliver exactly the desired amount. This example shows that, in general, it may not be possible to deliver the desired amount and thus one needs to choose among many possible allocations.

3. MATHEMATICAL FORMULATION

We are now ready to formulate the problem mathematically. Let $C_1, \ldots, C_R$ be the contracts to be filled and let $P_1, \ldots, P_p$ be the pools. Assume that for each contract j Amount$(C_j) \leq 1,000,000.00$.

Divide all R contracts into three disjoint groups: H ("high"), E ("exact") and L ("low") as follows:

$$H = \{ j \mid \text{Desired}(C_j) = 1.02499999 \times \text{Amount}(C_j) \}$$
$$E = \{ j \mid \text{Desired}(C_j) = \text{Amount}(C_j) \}$$
$$L = \{ j \mid \text{Desired}(C_j) = 0.97500001 \times \text{Amount}(C_j) \}$$

An allocation is the specification of the amount from each pool to satisfy sell contracts $C_1, \ldots, C_R$. It is therefore convenient to represent the allocation as the matrix A where the element a_{ir} is the amount used from mortgage pool i to satisfy contract j. We will

find it convenient to specify the allocation A in the following form(the first column will indicate the amount of money left in the pools after the allocation):

$$A = \left(\begin{array}{c|ccccc} \text{Cur\_Value}(\text{P}_1) - \sum_{r=1}^{R} a_{1r} & a_{11} & \cdots & a_{1j} & \cdots & a_{1R} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \text{Cur\_Value}(\text{P}_i) - \sum_{r=1}^{R} a_{ir} & a_{i1} & \cdots & a_{ij} & \cdots & a_{iR} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \text{Cur\_Value}(\text{P}_p) - \sum_{r=1}^{R} a_{pr} & a_{p1} & \cdots & a_{pj} & \cdots & a_{pR} \end{array} \right)$$

The PSA rules then correspond to constraints on the rows and columns of A. For example, $\sum_{i=1}^{p} a_{ir}$ is the amount delivered to contract r, whereas $P_i + \sum_{r=1}^{R} a_{ir}$ is the original amount in pool i. Let us define $\delta(x) = 1$ if $x > 0$ and $\delta(x) = 0$ if $x = 0$. In particular, $\delta(a_{ij})$ tells us whether pool i is used to satisfy contract j, whereas $\sum_{i=1}^{p} \delta(a_{ij})$ is the number of pools used to satisfy contract r.

The PSA rules can be concisely stated in terms of the matrix A as follows.

$$(1) \quad \sum_{i=1}^{p} \delta(a_{ij}) \leq \text{Max\_Pools\_Allowed}(\text{C}_j) \qquad 1 \leq j \leq R$$

$$(2) \quad \left| \sum_{i=1}^{p} a_{ij} - \text{Amount}(\text{C}_j) \right| \leq 0.02499999 \times \text{Amount}(\text{C}_j) \qquad 1 \leq j \leq R$$

$$(3) \quad a_{ij} = 0 \text{ or } \geq \$25,000 \times \text{Factor}(\text{P}_i) \qquad \text{for all } i, j$$

$$(4) \quad \text{Cur\_Value}(\text{P}_i) - \sum_{r=1}^{R} a_{ir} = 0 \text{ or } \geq \$25,000 \times \text{Factor}(\text{P}_i) \qquad \text{for all } i, j$$

$$(5) \quad \sum_{j=1}^{R} \delta(\text{Tail}(a_{ij}/\text{Factor}(\text{P}_i))) \leq 1 \qquad 1 \leq i \leq p$$

$$(6) \quad \text{Tail}(a_{ij}/\text{Factor}(\text{P}_i)) = 0 \text{ or } \text{Tail}(\text{P}_i/\text{Factor}(\text{P}_i)) \qquad \text{for all } i, j$$

$$(7) \quad a_{i_1 j} + \cdots + a_{i_k j} < 0.975001 \times \text{Amount}(\text{C}_j) \qquad 1 \leq i_1 < \cdots < i_k \leq p$$
$$1 \leq j \leq R, \; k < \sum_{i=1}^{p} \delta(a_{ij})$$

Inequality (1) is the constraint on the number of pools that can be used to satisfy a contract. Inequality (2) implies that one may deliver up to $\pm 2.499999\%$ of the value of a contract. Expression (3) means that each delivered pool amount must have the face value of at least \$25,000. Expression (4) means that either a pool is totally used for delivering a contract or it has at least \$25,000 of the original face value left (i.e. no fragmentation has occurred). Inequality (5) implies that no more than one tail may be used for any contract. Expression (6) implies that a tail cannot be split. Finally, expression (7) implies that no proper subset of pools allocated to a contract may fall within the allowable variance of the

contract.

We are now ready to formulate the pool allocation problem:

MORTGAGE POOL ALLOCATION PROBLEM: Given the contracts $\{C_1, \ldots, C_n\}$ and the pools $\{P_1, \ldots, P_p\}$, compute the allocation matrix A which maximizes

$$\text{Profit}(A) = \sum_{j \in H} \left(\sum_{i=1}^{p} a_{ij} - \text{Amount}(C_j) \right) + \sum_{j \in L} \left(\text{Amount}(C_j) - \sum_{i=1}^{p} a_{ij} \right)$$

where the matrix A satisfies the constraints of the PSA rules.

If we define the cost of an allocation as "cost(A) $= -$profit(A)", then the problem could be stated as follows:

Given the contracts $\{C_1, \ldots, C_n\}$ and the pools $\{P_1, \ldots, P_p\}$, compute the allocation matrix A which minimizes

$$\text{Cost}(A) = - \left[\sum_{j \in H} \left(\sum_{i=1}^{p} a_{ij} - \text{Amount}(C_j) \right) + \sum_{j \in L} \left(\text{Amount}(C_j) - \sum_{i=1}^{p} a_{ij} \right) \right]$$

where the matrix A satisfies the constraints of the PSA rules.

The mortgage pool allocation problem presented above is an example of constrained *combinatorial optimization*. There is an objective "cost" ("profit") function that has to be minimized (maximized). In our case, the problem is to compute the amounts contributed to contracts from all pools, such that the allocation satisfies the PSA rules and is of minimum cost. The space over which we minimize the cost function is discrete and is very large: possible assignments of pools to good millions. The number of possible ways to perform the allocation (the configuration space) is factorially large (despite the restrictions of the PSA rules) and cannot be explored exhaustively. It can be proved ([20]) that the mortgage pool allocation is an instance of well-known combinatorial optimization problems that belong to the class of *NP*-hard problems: there is, probably, no exact algorithm whose worst-case time complexity is polynomial ([2]). In practice, this means that there is no efficient algorithm for mortgage allocation that is guaranteed to give the best possible allocation†. We suggest addressing this problem using the method of simulated annealing.‡

† Such optimization problems are ubiquitous in the financial industry.

‡ An alternative approach, using rule-based systems, is described in [20].

4. SIMULATED ANNEALING

Simulated annealing is a probabilistic algorithm for obtaining good solutions of combinatorial optimization problems† . It was first introduced in the early 1950's by Metropolis ([9]), and then reintroduced in the early 1980's by Kirkpatrick ([6]). It is used for problems in which we wish to find an optimal solution out of a large number of possible solutions; each solution can be assigned a cost, and we seek the solution with minimal cost.

The method of simulated annealing is based on the analogy of cooling and annealing in statistical physics. Annealing refers to a physical process in which a metal or glass is first heated until molten, and then slowly cooled until it reaches its most ordered state, in which it has the lowest energy. At high temperature, the molecules move freely. Under slow cooling, thermal mobility is lost and atoms gradually form a pure crystal, corresponding to the state of minimum energy. The cooling is meticulously controlled according to a predetermined annealing schedule, which specifies the sequence of temperatures, the duration at each temperature, and the temperature at which to stop. If the temperature is lowered too quickly, the physical system will end up in an imperfect state, containing microscopic stress points which could lead the material to crack or fragment. At each temperature T, the metal is allowed to reach *thermal equilibrium*, in which the probability of being in a state with energy E is given by the Boltzmann distribution:

$$\text{Probability}(\mathbf{E}=E) = \frac{1}{Z(T)} \cdot \exp\left(-\frac{E}{T}\right)$$

where $Z(T)$ is the normalization constant. The factor $\exp(-E/T)$ is called the *Boltzmann factor*.

This process of cooling a fluid into a low-energy state (e.g. growing a crystal) was shown to be similar in nature to that of finding an optimal solution of a large combinatorial optimization problem. In simulated annealing, the optimization problem to be solved corresponds to the physical system, and reaching the optimal solution (the one with the lowest cost) corresponds to the system reaching its lowest-energy state. At any point in the simulated annealing procedure, a particular solution to the problem is being considered; this is referred to as the current state of the system.

We introduce a control parameter called the **temperature**. This parameter will govern, along with a probability distribution, the manner in which our system will move

† It is also known as *Statistical Cooling* ([21]), *Probabilistic Hill Climbing* ([14]), *Stochastic Relaxation* ([3]), or *Monte Carlo Annealing* ([4]).

between the various solution states. The temperature parameter corresponds to the actual temperature of the physical system being cooled. The physical system will move readily from one state to another at a high temperature, whereas there will be much less motion at a lower temperature until, in the extreme, there is no motion at all when the temperature is below a freezing point. Likewise in simulated annealing, the system will exhibit a great deal of motion between solution states at high temperature, and will show gradually decreasing motion as the temperature is gradually lowered. Eventually, the temperature is so low that motion from one state to another is almost completely absent; at this point the simulated annealing algorithm halts and the state in which the system is frozen is the solution to the problem. As the temperature is lowered, the system tends to be in states of lower energy, i.e., it considers solutions that are better and better. This movement is shown in **Figure 1**. The solution being considered when the system is finally frozen is, generally, a near-optimal solution.

We note here that simulated annealing has also been fruitfully compared to thermodynamical systems of mechanical physics ([10][11]), and has been successfully applied to hard combinatorial optimization problems, including the traveling salesman problem([6]).

With respect to growing crystals, physical annealing is well understood. One begins raising the fluid to very high temperature. The temperature is slowly lowered until the crystal is formed. When the material begins to freeze, temperature must be reduced very slowly. With respect to combinatorial optimization problem, such as mortgage pool allocation, one starts with an arbitrary initial configuration and chooses some cost ("energy") function reflecting the quality of the allocation. Subsequently, if we have state A (e.g. an allocation of contracts), a new allocation A' is randomly generated from an admissible set of "neighboring" configurations. The "cost" (energy) of the proposed new allocation is subject to the acceptance criterion, based on the value T of control parameter "temperature", and this determines whether the new allocation is accepted. If the cost ("energy") is not increased (the new allocation is at least as good as the current one), we always accept this new allocation A'. On the other hand, if the new allocation A' has higher cost, the acceptance probabilities are distributed according to the Boltzmann factor. The higher the temperature, the more likely are we to accept an allocation with higher cost. The system sometimes goes *uphill* ("escaping from local minima") as well as *downhill*, but the lower the temperature, the less likely is any significant uphill movement.

To implement simulated annealing we need the following:

(1) representation of problem instances and potential solutions $\{S\}$

(2) a cost (or energy) function $E(S)$ for each possible solution state S, which, when mini-

mized, indicates the optimal solution to the underlying optimization problem

(3) a procedure "generate" to generate a new state S' from a current state S

(4) a control parameter T (temperature) and an annealing schedule which specifies the number of random changes to be generated at each value of T (*inner loop criteria* in the algorithm) and how T should be updated ("cooling schedule").

A step in the annealing algorithm proceeds as follows. Let the system be in a state S, which has energy E. Randomly generate a move to another state S' and calculate the energy E' of the new state. If the new state has a lower energy, $E' < E$, accept this move and the system is now in state S'. If, however, the new state has a higher energy than the current state, accept the move with a probability $\exp(-(E-E')/T)$ where T is the current temperature. This probability can be thought of as the relative probability that the system, when left to itself at the current temperature, will be in state S' as opposed to being in state S. In practice, this means that for constant temperature, the smaller the energy jump, the more likely the move will be accepted. For constant energy jump, the higher the temperature, the more likely the move will be accepted. Thus at higher temperatures the system will exhibit greater motion than at lower temperatures.

After a certain number of times of considering these randomly generated moves to new solution states, the temperature is lowered according to a chosen rule. A common rule to lower the temperature is $T = \alpha \cdot T$, where α is a constant ([1][6][15]). After a given number of temperature steps, or when the system appears frozen (motion has ceased), the annealing procedure stops, and the last state is output as the solution to the optimization problem.

Therefore, for the mortgage allocation problem the basic algorithm is as follows:

Simulated Annealing Algorithm

```
compute initial temperature T_0
compute initial allocation A
for each temperature value T
  begin
    while(inner loop criteria is not satisfied)
      begin
        A' =generate(A)
        if (oracle(energy(A),energy(A'),T)=1)
          accept new configuration A'
```

```
        end
      T = update(T);
    end
  print the results
```
Finished!

The acceptance of new allocation A' is governed by the following simple rule:

oracle(energy(A),energy(A'), T):

```
compute y = min{ 1.0, exp(-(energy(A) - energy(A'))/T)}
r = random(0,1);
if (r < y) return(1);
     else return(0);
```

It is instructive to compare simulated annealing with the more conventional problem solving approach of gradient descent. In gradient descent, at each step a move to a new solution state is generated and the cost (energy) calculated. If the new state has a lower cost (energy), then we have found a better solution and the move is accepted. If the new state has a higher cost (energy) the move is rejected since the current solution is better. The algorithm halts when it cannot find any states to move to. The problem with this approach is that it tends, for complex problems, to halt in states which are only local minima, not global minima. Moreover, these local minima are better than all the states which can be reached by a single move, but may be inferior to local minima that can be reached in a few moves.

Simulated annealing overcomes this problem by occasionally accepting a move to a higher (worse) state, depending on the difference in energy and on the current temperature. By moving to a higher state, the system can escape from the trap of local minima, giving it a greater chance of finding a global or near-global minima. This upward movement, which is governed by the probabilistic decision procedure described above, makes simulated annealing tremendously powerful in solving the most complex combinatorial optimization problems.

Therefore, the simulated annealing algorithm is not a "greedy" algorithm in the strict sense, like many iterative algorithms are: it does not always proceed in the direction of lower cost. The quality of the local minimum configurations are often quite inferior with

the global minimum state. Heuristic algorithms often end at these local minimum states because they are greedy: they only accept new states with lower cost. The quality of solutions obtained by simulated annealing depend drastically on the implementation and much experimentation is usually required – this is due to the lack of theoretical results concerning guidelines for optimal properties and control of the algorithm([15]). The initial "temperature", neighboring allocations and rate of "temperature" decrease are the parameters that will affect the speed of the algorithm and the quality of the final solution.

5. IMPLEMENTATION

Let us now discuss the implementation of the algorithm as applied to the mortgage pool allocation. In view of the above discussion, to apply the method we need to specify the following:

(1) Representation of potential allocations $\{A\}$ and the computation of an initial allocation.

(2) The "ENERGY" function $E(A)$ for each possible allocation A.

(3) Computation of an initial temperature.

(4) Computation of an initial allocation.

(5) a procedure "generate" to generate a new allocation A' from a current allocation A.

(6) Annealing Schedule: Stop Criteria, Iterations ("inner loop criteria") and updating the temperature ("cooling schedule").

We have formulated the problem initially as to compute the matrix A so that the cost(A) is minimized. This means that in generating new moves we have two alternatives. The first is to restrict the generation of moves so that each new state satisfies the PSA constraints. The second alternative is to allow moves to illegal allocation, but to introduce a penalty function, which describes the degree to which an allocation falls short of expectation. Potentially, the second approach gives a more powerful algorithm: any changes in the constraints, quality of pools, and other desirable information can be done by simple changes in the penalty function. However, the relative weighting of the penalties is known to have a dramatic effect on the quality of the obtained solution. Too heavy penalties may result in a legal allocation of poor quality. On the other hand, insufficient penalties may result in an illegal allocation. At this point, there are no known definitive standard guidelines on relative weighting of penalty terms and much experimentation is usually required to "tune" the energy function, generation of moves and the "cooling schedule"

([15]). Moreover, since, in practice, the second approach tends to be much slower, we have chosen the first approach: **all allocations considered during the execution of the algorithm are legal**.

We now proceed to describe the implementation of the simulated annealing algorithm for mortgage pool allocation:

• **Representation of potential allocations** $\{A\}$**:** We represent an allocation A as a matrix where the PSA rules translate on certain conditions on its rows and columns, as described in section 3.

• **The "ENERGY" function:** The mortgage pool allocation problem can be described as finding legal allocations of minimum cost (thus maximizing potential profit from using the $\pm 2.499999\%$ variance rule) and choosing one conforming to some desirable constraints (e.g. use of particular pools or avoid using a particular pool, if possible). Therefore, it is convenient to write the energy function as

$$\text{energy}(A) = \text{cost}(A) + \text{pool\_quality}(A)$$

Since the number and amount of contracts and pools is variable, we find it convenient to use "normalized" values for both $\text{cost}(A)$ and $\text{pool\_quality}(A)$, such that $0 \leq \text{cost}(A) \leq 1$ and $0 \leq \text{pool\_quality}(A) \leq 1$. We define such a "normalized" cost of an allocation A as follows:

$$\text{cost}(A) = \sum_{j \in H} \frac{1}{R}\left(1 - \frac{\sum_{i=1}^{p} a_{ij} - \text{Amount}(C_j)}{25,000}\right) + \sum_{j \notin H} \frac{1}{R}\left(1 - \frac{\text{Amount}(C_j) - \sum_{i=1}^{p} a_{ij}}{25,000}\right)$$

Note that $\%2.499999$ variance always results in an amount that is less than $\$25,000$ (recall that for each j we have $\text{Amount}(C_j) \leq 1,000,000$).

With this definition it is easy to show that minimizing cost of A is equivalent to maximizing the profit defined in section 3.

For the pool quality term,

$$\text{pool\_quality}(A) = \sum_{i=1}^{p} \frac{w_i}{p} \cdot \frac{\sum_{r=1}^{R} a_{ir}}{\text{Cur\_Value}(P_i)}$$

where $0 \leq w_i \leq 1$ is the weight given to pool i. The larger the weight w_i, the more "penalty" is given to using pool i. Therefore, for example, if larger pools are more valuable, we will

use larger weights for them. The use of these weights for the pool_quality term allows us to produce an allocation to increase the quality of the inventory†. In this way, the method will try to use or avoid using particular pools, but if it has to use more money in order to allocate, it will do so to prevent a fail.

Note that energy(A) is expressed as a sum of R terms for the contracts (in the "cost" term) and of p terms for the pools (in the "pool quality" term), and therefore, changes in energy can be computed locally: a typical move (described below) involves at most one contract and two pools. Therefore, once we compute the energy of the initial configuration (this requires $O(R \cdot p)$ steps), we can compute the energy change of a new configuration by using values of the two pools and one contract involved. If we keep track of the amounts allocated to each contract and of the amounts used from each pool (this clearly requires extra $R+p$ storage locations), then the energy of the new configuration can be computed from the energy of the previous one in constant time regardless of the number of pools and contracts.

• **Computation of initial temperature:** The method follows directly from the theory of simulated annealing: annealing (cooling) must start when the system is melted (ergodic). The ideal starting temperature is the lowest temperature at which the system is melted. In other words, at the initial temperature virtually all transitions are accepted. Our procedure then is to place the system at an extremely low temperature, essentially frozen, and then quickly heat it until it is melted.

We use the following procedure, an improvement from one described in ([7]): we choose an extremely small number ϵ which is smaller by an order of magnitude than the measurements of the energy of the system, and set the temperature of the system $T = \epsilon$. We now generate a small number of random moves, say 100. The ergodicity is then measured simply as the fraction of these moves that were accepted by the simulated annealing decision procedure. At first this fraction will be low, because no moves to a higher energy state will be accepted when the system is frozen. We then double the temperature and measure the

† To give just some examples of *pool quality*: recently issued pools may be more valuable. Larger pools (larger face values) are more valuable since they are easier to use for splitting purposes and use of variance. The mortgage pools of California became *more valuable* after the 1989 San Francisco earthquake: the speculation may be that people will be more willing to sell their homes and move out. This means that the mortgages in the underlying pools will be paying down very rapidly. There could be then a growing market for California pools, more of a demand and subsequently higher prices for them.

ergodicity again by generating another 100 moves. This step is repeated until the ergodicity exceeds a certain threshold, such as 0.9. At this point, the temperature is so high that 90% of all moves were accepted, and we say that the system is melted. The temperature at this point then becomes the initial temperature for the simulated annealing.

This relatively quick and simple method of computing the initial temperature ensures that (1) annealing will start when the system is indeed ergodic (melted), and (2) the temperature is not much higher than the minimum necessary for ergodicity. We can now be confident that the annealing schedule will move the system efficiently from melted to frozen without excessive time at the high temperature stage‡.

• **Computation of the initial allocation:** We are looking for a fast heuristic. One example is the following:

Step 1: Sort pools by amount
Step 2: Let C_j be the next contract to allocate
and let n = Max_Pools_Allowed(C_j), X = Amount(C_j)/n
Step 3: From the sorted pool list, use up to $n-1$ largest pools
whose current value is less than X
Step 4: Find the n-th pool large enough to split
for the remaining amount
Step 5: Go back to Step 1

Of course, in performing steps 3 and 4 we follow the constraints on the tails, pool fragments, etc. Intuitively, the algorithm tries to delay using large pools and tries to get rid of small pools. As for its running time, we note that at each iteration it involves sorting. For p pools, a simple sorting algorithm (e.g. bubble-sort) takes $O(p^2)$ steps. However, if one uses double linked lists as the underlying data structure, we only need to do sorting at the beginning of the algorithm and then perform local changes. In practice, its running time is negligible for inputs with hundreds of pools and contracts.

‡ We should mention here another widely used method ([25]): If one makes the assumption that the energy is distributed by Gaussian distribution with standard deviation σ, it is argued that an appropriate temperature T should be high enough to accept with probability P_A a configuration whose energy is 3σ the energy of the current configuration. This gives the rule $T = -3\sigma/\log P_A$. To determine σ, one runs the algorithm and accepts all possible transitions.

• **Generating new allocations:** We would like to define a set of transformations (moves) so that starting with some allocation matrix A, we can use these moves to obtain any other valid allocation matrix A'. We have chosen one such general purpose transformation:

$$A = \left(\begin{array}{l|cccc} \text{Cur\_Value}(P_1) - \sum_{r=1}^{R} a_{1r} & a_{11} & \cdots & \cdots & a_{1R} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \text{Cur\_Value}(P_i) - \sum_{r=1}^{R} a_{ir} & a_{i1} & \cdots & a_{ij} & \cdots \; a_{iR} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \text{Cur\_Value}(P_k) - \sum_{r=1}^{R} a_{kr} & a_{k1} & \cdots & a_{kj} & \cdots \; a_{kR} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \text{Cur\_Value}(P_p) - \sum_{r=1}^{R} a_{pr} & a_{p1} & \cdots & \cdots & a_{pR} \end{array}\right) \cdot \begin{array}{c} \\ \\ \\ \\ \\ a \end{array}$$

$$\longrightarrow A' = \left(\begin{array}{l|cccc} \text{Cur\_Value}(P_1) - \sum_{r=1}^{R} a_{1r} & a_{11} & \cdots & \cdots & a_{1R} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \text{Cur\_Value}(P_i) + X - \sum_{r=1}^{R} a_{ir} & a_{i1} & \cdots & a_{ij} + X & \cdots \; a_{iR} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \text{Cur\_Value}(P_k) - X' - \sum_{r=1}^{R} a_{kr} & a_{k1} & \cdots & a_{kj} + X' & \cdots \; a_{kR} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \text{Cur\_Value}(P_p) - \sum_{r=1}^{R} a_{pr} & a_{p1} & \cdots & \cdots & a_{pR} \end{array}\right) \cdot \begin{array}{c} \\ \\ \\ \\ \\ a \end{array}$$

This move can be described as follows: randomly choose contract C_j and two pools i and k. If pool i is used to deliver $a_{ij} > 0$ to contract j, take some amount $X \leq a_{ij}$ and try to allocate that amount from pool k. We usually would not be able to reallocate exactly X because pool k will have, in general, a different factor and tail than pool i. Thus, we allocate an amount X' close to X from pool k to contract C_j.

Note that this move involves at most two pools and one contract. This means that saving/restoring an allocation and generating a new one can be done very efficiently by just modifying one column and two rows. In other words, this step in the simulated annealing algorithm takes constant time, independent of the size of the input.

• **Annealing Schedule:** The stop criteria is usually determined by fixing the number of temperature steps T for which the algorithm is to be executed. As for the "inner loop criteria", the simplest choice is to fix the number of iterations at each temperature by a value depending polynomially on the size of the problem([1] [15]), for example by making

it proportional to $p+R$. Other suggestions are based on the argument that for each value of T, a minimum amount of transitions are accepted ([6]).

As for the "Cooling Schedule" we follow a frequently used rule $T = \alpha \cdot T$, where α is a constant smaller than but close to 1. The typical range of α is $0.8 \leq \alpha \leq 0.95$ ([1] [6] [15]). It is sometimes argued that one should have smaller α in the beginning and final stages of the algorithm. We have fixed $\alpha = 0.9$ throughout the algorithm.

6. EXPERIMENTAL RESULTS

We implemented the simulated annealing model for various sets of sample data. The tests were made on a Sun 4/330. The temperature was decreased according to $T = 0.9T$. Annealing was halted when no move was accepted at one temperature level, up to a maximum of 100 temperature steps. The number of iterations (attempted moves) at each temperature was variable, though an amount directly proportional to the number of pools p plus contracts R seems desirable. The same number of moves are attempted at each temperature level. The pool weights w_i's are all set to zero (i.e. we consider all pools to be equally valuable). Some of the larger sample data sets had more than 500 pools and more than 500 contracts. The contracts represented approximately \$550,000,000. Our method obtained good solutions even for such large allocations in real-time: in less than 2.5 minutes running on an IBM RS/6000 workstation.

For any allocation, we can measure the percentage of variance achieved, i.e., the proportion of profit achieved by delivering, for each contract, an amount larger or smaller than specified dollar value of the contract. 100% of variance then means that the allowed 2.499999% variance was perfectly delivered, in the correct direction, on each contract. In our tests, we measure this percentage before and after annealing. We use the difference in percentage ("improvement") as the measure of how well simulated annealing performed.

Figure 1 demonstrates how the energy of the system is gradually lowered as the system cools. Note the initial jump in the energy at the beginning of the procedure. This is because the initial state of system is not random, but the result of a heuristic allocation algorithm. We can infer that this initial allocation is at a local minimum of the energy function; the first moves at high temperatures tend to "shake up" the system out of this local minimum, thus raising the energy at the beginning of annealing. Note also that there seems to be a phase transition just above $T = 1.0 \times 10^{-3}$. At this point ergodicity is broken and the energy starts to decrease more rapidly.

Simulated annealing proved to be sensitive to the ratio of the total money in the pools to the money needed to fill all the contracts. When this ratio was large, the annealing algorithm had greater flexibility to find better allocations, because more moves were available. When the pools-to-contracts ratio was small, however, there was very little flexibility for adjusting the allocation, and, as a result, the performance by simulated annealing was poor. These results are demonstrated in **Figure 2**. We tested the model on data sets that were similar except for the pools-to- contracts ratio. As the ratio approached 1.0, the improvement found by simulated annealing tends asymptotically towards zero. This deterioration is reasonable since the algorithm depends on having room to maneuver the allocation by the designated moves.

We also varied the number of iterations (attempted moves) at each temperature while holding all other variables constant. The number of iterations are calculated as multiples of $(p+R)$. The results, seen in **Figure 3**, show, as expected, that the more iterations at each temperature, the better performance achieved by simulated annealing. This is because with more iterations, the system better approximates thermal equilibrium at each temperature. The marginal improvement, however, declines with increasing iterations because each increment brings the system less close to equilibrium than the previous increment. Theoretically, after a certain number of iterations per temperature, the system would be in thermal equilibrium, and thus further increasing the iterations would not improve performance at all.

7. CONCLUDING REMARKS

In this paper we have demonstrated that the method of simulated annealing can provide an efficient way to address the problem of mortgage pool allocation. Pool/contract priorities and other desirable characteristics could easily be incorporated by simple changes in the energy function.

Most previous applications of simulated annealing have shown its efficacy for solving classical combinatorial optimization problems with a small number of constraints, such as the traveling salesman problem (TSP). The current application, mortgage pool allocation, differs from these other problems in several ways. There are quite a few constraints in the current problem specified by the PSA rules. They are significantly more complicated than the constraints of TSP; the PSA rules often appear strange and arbitrary, and are difficult to enforce. As a result, the moves in the mortgage pool application are more complicated

than those in applications to standard combinatorial optimization problems, where a set of simple, efficient, and intuitively natural moves may be readily apparent. Lastly, since we wished to demonstrate feasibility of simulated annealing for financial institutions, we needed to have a solution in real-time for large number of contracts and pools. Towards this end, we made whatever compromises were necessary in setting the annealing schedule in order to obtain a solution within a few minutes or less, on a medium-sized workstation. We have thus shown that good solutions are obtainable in real-time using simulated annealing. We hope that these results will encourage use of simulated annealing in commercial settings, as well as provide a foundation for further research applications of simulated annealing.

7. REFERENCES

[1] E. Bonomi, J.-L. Lutton, "The N-city Travelling Salesman Problem: Statistical Mechanics and the Metropolis Algorithm", *SIAM Rev.*, **26**, 1986, pp. 551-568.

[2] M.R. Garey, D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", *Freeman Publishing*, San Francisco, 1979.

[3] S. Geman, D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images", *IEEE Proc. Pattern Analysis and Machine Intelligence*, PAMI-6, 1984, pp. 721-741.

[4] D.W. Jepsen, C.D. Gelatt, "Macro Placement by Monte Carlo Annealing", *Proc. Int. Conf. on Computer Design*, 1983, pp. 495-498.

[5] L.J. Karcher, "Processing Mortgage-Backed Securities", *Simon and Shuster*, 1989.

[6] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, "Optimization by Simulated Annealing", *Science*, **220**, 1983, pp. 671-680.

[7] P.J.M. van Laarhoven, E.H.L. Aarts, "Simulated Annealing: Theory and Applications", *Kluwer Academic*, 1987.

[8] L. Lowell, K.H. Sullivan, "Mortgage Pass-Through Securities", in *The Handbook of Mortgage-Backed Securities*, (F.J. Fabozzi ed.), Probus Publishing, Chicago, Ill., 1988, pp. 69-114.

[9] N. Metropolis, A. Rosenbluth, A. Teller, E. Teller, "Equation of State Calculations by Fast Computing Machines", *J. Chem. Physics*, **21**, 1953, pp. 1087-1092.

[10] M. Mezard, "Spin Glasses and Optimization", in *Heidelberg Colloquium on Glassy Dynamics and Optimization*, J.L. van Hemmen and I. Morgenstern eds., Springer Lecture

Notes in Physics, **25**, 1987, pp. 354-372.

[11] M. Mézard, G. Parisi, M. Virasoro, "Spin Glass Theory and Beyond", World Scientific, 1987.

[12] M.P. McLaughlin, "Simulated Annealing", *Dr. Dobb's Journal*, September 1989, pp. 26-36.

[13] C. Ramsey, J. M. Henderson, "Specified Pools", in *The Handbook of Mortgage-Backed Securities*, (F. J. Fabozzi ed.), Probus Publishing, Chicago, Ill., 1988, pp. 115-129.

[14] F. Romeo, A. L. Sangiovanni-Vincentelli, "Probabilistic Hill Climbing Algorithms: Properties and Applications", *Proc. 1985 Chapel Hill Conf. on VLSI*, 1985, pp. 393-417.

[15] C. Sechen, "VLSI Placement and Global Routing Using Simulated Annealing", *Kluwer Academic*, 1988.

[16] D. Senft, "Mortgages" in *The Handbook of Fixed Income Securities*, (F. J. Fabozzi and I. M. Pollack eds.), Dow-Jones Irwin, 1987, pp. 352-381.

[17] S. R. Schulman, "Forward Prices and Dollar Rolls", in *Advances and Innovations in the Bond and Mortgage Markets*, (F. J. Fabozzi ed.), Probus Publishing, Chicago 1989, pp. 519-526.

[18] M. K. Stamper, K. E. Kennedy, N. Summers, "Trading Mortgage-Backed Securities", in *The Handbook of Mortgage-Backed Securities*, (F. J. Fabozzi ed.), Probus Publishing, Chicago, Ill., 1988, pp. 131-145.

[19] M. Stigum, F. J. Fabozzi, "Mortgage Pass-Through Securities: Ginnie Maes, Fannie Maes and Freddie Macs", in *The Dow Jones-Irwin Guide to Bond and Money Market Investments*, (by M. Stigum and F. J. Fabozzi), Dow-Jones Irwin, 1985, pp. 241-258.

[20] S. Stolfo, L. Woodbury, P. Chan, "The Alexsys Mortgage Pool Allocation System: A Case Study of Speeding Up Rule-Based Systems", *Technical Report*, Computer Science Department, Columbia University, July 1990.

[21] J. A. Storer, A. J. Nicas, J. Becker, "Uniform Circuit Placement" in *VLSI: Algorithms and Architectures*, P. Bertolazzi and F. Luccio eds., Elsevier Publishing, 1985, pp. 255-273.

[22] K. H. Sullivan, B. M. Collins, D. A. Smilow, "Mortgage Pass-Through Securities", in *The Handbook of Fixed Income Securities*, (F. J. Fabozzi and I. M. Pollack eds.), Dow-Jones Irwin, 1987, pp. 382-403.

[23] K. H. Sullivan, "The Mortgage Market", in *The Handbook of Mortgage-Backed Securities*, (F. J. Fabozzi ed.), Probus Publishing, Chicago, Ill., 1988, pp. 53-67.

[24] K. J. Thygerson, C. Brown, "Mortgages and Mortgage-Backed Securities", in *Hand-*

book of Financial Markets: Securities, Options and Futures, 2-nd edition, (F. J. Fabozzi, F. G. Garb eds.), Dow-Jones Irwin, 1986, pp. 378-403.

[25] S. R. White, "Concepts of Scale in Simulated Annealing", *Proc. IEEE Int. Conf. Computer Design*, Port Chester, November 1984, pp. 646-651.

Figure 1. Energy vs. Temperature

Figure 1: The progress of the algorithm is from right to left. As the temperature is lowered, the energy of the system is lessened, corresponding to better solutions to the optimization problem.

Figure 2. Improvement vs. Ratio

Figure 2: As the pools-to-contracts ratio is decreased towards 1.0, the performance of simulated annealing deteriorates. This is because for a small ratio, there is very little room to use the moves for maneuvering and adjusting the allocation. The percentage after the initial allocation was 0.84.

Figure 3. Percent of Profit vs. Number of Iterations

Figure 3: Improvement due to simulated annealing increases as the number of iterations (attempted moves) per temperature increases. Note that the marginal improvement decreases as thermal equilibrium is approached.

Number of Iterations	Final % Change	Change in %
121	0.93091	0.09134
504	0.97144	0.13187
1008	0.98586	0.14629
1512	0.98893	0.14936
2016	0.99121	0.15164
2520	0.99197	0.15240
3024	0.99311	0.15354

Comparing Solution Quality When Varying the Number of Iterations for each T

(ratio = 1.2, initial %of variance =0.83957)

Ratio	Start %	End %	Change
2.00	0.8064	0.9785	0.1721
1.80	0.8218	0.9704	0.1486
1.60	0.8223	0.9555	0.1332
1.30	0.8555	0.9763	0.1208
1.20	0.8396	0.9309	0.0913
1.10	0.8409	0.9150	0.0741
1.04	0.7153	0.7655	0.0502
1.01	0.7009	0.7192	0.0183

Comparing Similar Data Sets with Different Pools-to-Contract Ratio