A Simulated Annealing based approach for solving SplitSearch

Shreya Malani.* Sreenidhi T.* G N Srinivasa Prasanna.*

* *International Institute of Information Technology, Bangalore, India.*

shreya.malani@iiitb.org, sreenidhi.t@iiitb.org, gnsprasanna@iiitb.ac.in

Abstract

Real-time applications that provide location-based services according to the user queries, can in general be modelled as large-sized combinatorial optimization problems. The primary criterion in such commonly used location-based services is to provide an optimal solution from a dynamically-changing solution space, which is exponential in the input size, meeting real-time constraints.

In this paper, we discuss a heuristic approach based on Simulated Annealing technique to address one such location-specific problem of SplitSearch. With a user requesting for 'n' services and there being 'm' providers for each of these 'n' services, Split-Search addresses the selection of the optimal set of providers, with respect to the distance travelled, that can satisfy the requested services. The paper illustrates the improvement in the quality of the solution thus obtained in comparison with that of Greedy approach. A mathematical formulation is also proposed for the problem, in the form of an Integer Linear Program (ILP).

1. Introduction

SplitSearch is a location-based search used to satisfy user's query for multiple services in an optimal way. Objective of the algorithm is to choose a set of locations that provide these requested services and also optimize the distance (or number of nodes) travelled for satisfying the same. For instance, if a user wants to visit a hospital, a bank and a restaurant, SplitSearch algorithm would select node(s) that together satisfy these services with an optimized route. SplitSearch is an intermediary of two NP hard problems and thus itself is a NP hard problem. At one extreme where each node (city) satisfies one specific service requested by the client it is Travelling Salesman Problem (TSP), and at the other extreme where communication cost is zero, it is Set Cover Problem (SCP).

A user having multiple things to do, would generally choose to go to the nearest point satisfying one of his/her requirements, then to the next point nearest from that point and so on, till all the required services are satisfied. This obvious Greedy method of satisfying one's requirements is not necessarily optimal always. In the following sections, we describe the approach based on Simulated Annealing[1] that picks one node from each of the sets satisfying user criteria and iterates each time choosing a better feasible solution, eventually tending towards the optimal path. The ILP and the mathematical model for the problem are explained in further sections. The experimental results of the algorithm thus implemented for Splitsearch on different kinds of test graphs show considerable improvement over the traditional Greedy approach and the details of the same are discussed in the Experimental Results section of the paper.

2. Algorithm

Simulated Annealing (SA) is inspired from the process of Annealing in metals. Annealing[2] is the physical process of first heating the solid state metal at a sufficiently high temperature and cooling it down

very slowly, according to a specific schedule. At each temperature, sufficient time is given for the system to reach a steady state. The lower the temperature, the higher is the time given. If the heating temperature is sufficiently high to ensure random state and the cooling process is slow enough to ensure thermal equilibrium, then the atoms shall place themselves in a pattern that corresponds to the global minimum energy of a perfect crystal.

In Simulated Annealing implementation, the above physical process is imitated by first identifying the correspondence between the metal and the optimization problem and then using the algorithm to simulate each step of annealing. Each basic step of simulation perturbs the value of one of the variables that determine the solution by a small amount. If the new configuration has a favourable cost, the configuration is accepted. If the new configuration has an unfavourable cost, even then it is accepted with a certain probability. SA algorithm repeats the application of the above basic step until no more improvement in the cost function is possible or a stable state is reached by the system. As evident, SA allows hill climbing from local optima. Acceptance of solutions of unfavourable cost on a probabilistic basis ensures that the algorithm does not remain in the valley of local optimum. The SA algorithm is technically a local search algorithm in which there are occasional upward moves that lead to a cost increase and it is expected that these upward moves will help escape from local minima, moving the solution towards the global optimum.

The initial value for the annealing temperature, threshold value of temperature and the type of annealing schedule are specific to the problem being addressed and are set based on experimentation.

In the Simulated Annealing based approach for SplitSearch problem, the probabilistic selection or rejection of a path in every iteration is based on the value generated by a function of current annealing temperature and path length. In every iteration, the new path is obtained by perturbing the path solution of the previous iteration. This is done by picking one node of the path satisfying a specific requirement at random and replacing it with another node satisfying the same requirement. The potential candidate for the replacement node is chosen by sweeping for a node satisfying same requirement in the vicinity of the current node, which when included in the new path does not differ from the previous path length by more than a pre-defined scaling factor, which in our implementation is taken as half. Refer to Section 2.1 for details on perturbation.

```
begin
    if temperature ≥ threshold temperature then
        doAnnealing;
        Tweak the solution obtained in previous
        iteration ;
        and based on random probability, accept
        the solution ;
    else
        doGreedy;
        Accept a tweaked solution only if it is
        better ;
    end
end
```

Algorithm 1: Simulated Annealing

The initial temperature is initialized to a value of 100, in the implementation for SplitSearch. This temperature value is decremented by a factor 1% in every iteration until the solution stabilizes (The solution is considered to be stable if it remains unchanged for fifty successive iterations). The solution at every iteration is accepted based on some probability until the threshold temperature, which is 1.0E-4 in our implementation.

$$difference = \frac{pathLength - minimalLength}{\sum edgeLengths} \quad (1)$$

$$probability = \exp\left(\frac{-difference}{temperature}\right) \quad (2)$$

The probability is based on the difference in the previous and present path lengths with some scaling factor. In the implementation, this scaling factor is taken to be the sum of the lengths of all the edges in the graph as shown in equation (1). Equation (2) shows the probability of accepting a new path. Algorithm 1 shows a pseudo-code for the implementation.

2.1. Perturbation

The implementation starts with a feasible solution obtained from the Greedy approach. In the next iteration, the solution obtained from Greedy algorithm is changed a little by the above explained process of tweaking. A solution formed after tweaking is accepted based on some probability. At initial temperature an unfavourable solution is also accepted but as the temperature decreases only favourable

solutions are accepted. This allows the solution to escape a local minima. The doAnnealing function tweaks the solution in every new iteration and accepts the solution based on some probability. Once the threshold temperature is reached, only a solution which is better than previous iteration is accepted. The solution having minimum path length is stored and updated accordingly after every iteration. Once the solution stabalizes the best solution obtained so far is given as the final solution.

In the implementation, perturbation involves the following steps:

1 Choose a node randomly to alter and note the service this node provides.
2 Find a node in the vicinity of the solution that can replace the node to be altered. The vicinity radius is heuristically decided. If this radius is too large, an unacceptably large variation in the path lengths is observed, whereas a smaller radius will not change the path at all (as it might not find any new node for replacement).
3 After the node is replaced, reorder the nodes to obtain the configuration with the selected nodes that has the shortest path length.

2.2. Integer Linear Program formulation for SplitSearch problem

An outline of the mathematical representation for the problem statement of SplitSearch is as follows:

Problem Statement:
Given a graph G(V,E), with the set of vertices V representing the nodes satisfying various attributes and set of edges E representing the links between the nodes with edge weights as the routing cost. Find the optimal path (distance-wise), that satisfies all the requirements of a user.

Notations Used:

- C_{ij}^k - *Cost(path length) of travelling from i^{th} to j^{th} node, providing k^{th} service at j^{th} node.*
- e_{ij}^k - *0, if edge i to j is not selected, >= 1 otherwise. The value depends on the number of times the edge is traced. e_{ij}^k is by definition 0, if the attribute satisfied by node j is not requested.*
- *Dummy start and end nodes are introduced to satisfy flow equations at start and end nodes. These dummy nodes are not part of the original graph and do not satisfy any of the requested services.*

Objective Function:
Minimize the Cost or the total path length.

$$\text{Min} \qquad \sum_i \sum_j \sum_k \quad C_{ij}^k e_{ij}^k$$

Subject to Constraints:

1 Inflow Constraint:

$$\sum_k \ e_{(dummystart)(start)}^k = 1 \tag{3}$$

There should be inflow in the system from the dummy start node.

2 Outflow Constraint:

$$\sum_j \sum_{k'} \ e_{(j)(dummyend)}^{k'} = 1 \tag{4}$$

There should be outflow from the system to dummy end node that provides the service k', not in the set of requested services.

3 Atleast one outgoing edge from Start node.

$$\sum_j \sum_k \ e_{(start)(j)}^k \geq 1 \tag{5}$$

In the presence of loops, Start node can have more than one outgoing edge.

4 Flow conservation:

$$\sum_i \sum_j \sum_k \ e_{ij}^{k1} = \sum_j \sum_l \sum_k \ e_{jl}^{k2} \tag{6}$$

The sum of all inflow over all services on j^{th} node from any node i should be equal to the sum of all outflow over all services to some other node l.
Note: i, j and l are nodes that do not include dummy nodes.

5 Satisfaction of all requests:

$$\sum_i \sum_j \ e_{ij}^k = v^k \tag{7}$$

where v^k is the total number of visited links that satisfy the k^{th} request.

$$\sum_k \ v^k \geq 1 \tag{8}$$

6 Loop Constraint: We use an Oracle-based cutting plane for removing disconnected self-loops if any, from the solution.

The cutting plane ensures that there is an inflow to or outflow from the cut sets, thereby eliminating the possibility of having disconnected components in the solution. It ensures connectivity between the start node and the nodes satisfying various attributes. Since the cut sets can be exponential in number, they are chosen as needed with the help of an Oracle.

In the case of **Multi-attribute Nodes**, a node may provide multiple services. Thus, visiting one node might satisfy more than one of user's requests. Also, more than one node might be selected providing the same service. Say, for example, a mall is a multiattribute node as it provides an ATM, a theatre as well as restaurants and coffee shops. The constraints in this case are similar to the ones formulated for Single-attribute nodes.

2.3. Example - Tree Topology

Figure 1 shows a sample graph. S is the starting

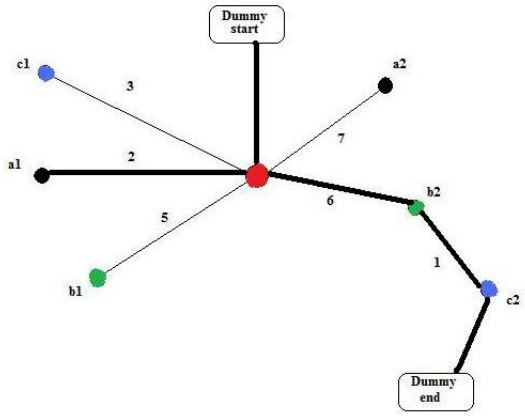

Figure 1: Sample Graph

location. The requested services are 'a', 'b' and 'c'. Nodes a1 and a2 provide same service 'a', nodes b1 and b2 provide service 'b' and nodes c1 and c2 provide service 'c'. The solution generated by the implementation is explained as follows.

- Greedy Solution: S a1 c1 b1 = 15
- SA Solution: S a1 b2 c2 = 11
- LP Solution: S a1 b2 c2 = 11

2.3.1. Simulated Annealing Approach.

1 Find an initial feasible solution using the Greedy approach, which is S a1 c1 b1 with path length 15 for the graph.

2 Simulated Annealing algorithm

i From the previous feasible path randomly choose a node to replace, say c1 in this case.

ii Find a node that can take the place of the node to be replaced (i.e. provides the same service). Only node that provides same service as c1 is c2, which makes the path length as 23 (S a1 c2 b1).

iii Reorder these nodes to get minimum path length. Order S a1 b1 c2 gives path length 21 < 23.

3 At a higher temperature this path is accepted with random probability, even if its length is more than previous computed path length.

4 Repeat steps under (2) for next iteration.

5 Say b1 is replaced by b2, path becomes S a1 b2 c2 with path length 11.

6 This new path in accepted and steps under (2) are repeated till threshold temperature is reached.

7 At temperatures below threshold temperature, a solution is accepted only if it is better than the previous solution.

8 The process repeats till the system stabilizes and the best of solutions obtained, one with path length 11 is returned as the final solution.

2.3.2. LP formulation.

Objective

$$\begin{aligned} \textit{Minimize } & 7*e^0_{s\,a2}+6*e^1 s\,b2+1*e^2_{b2\,c2}+ \\ & 5*e^1_{s\,b1}+2*e^0_{s\,a1}+3*e^2_{s\,c2} \end{aligned} \tag{9}$$

Subject to the constraints:

1. Inflow and Outflow in the network. 'k' is some dummy category not requested by the user.

$$e^k_{dummyStart\ s}=1 \tag{10}$$

$$\begin{aligned} & e^k_{a1\ dummyEnd}+e^k_{a2\ dummyEnd}+e^k_{b1\ dummyEnd} \\ & +e^k_{b2\ dummyEnd}+e^k_{c1\ dummyEnd}+e^k_{c2\ dummyEnd}=1 \end{aligned} \tag{11}$$

2. At least one outgoing edge from start node.

$$e^0_{s\,a1}+e^0_{s\,a2}+e^0_{s\,b1}+e^0_{s\,c1}+e^0_{s\,c2}\geq 1 \tag{12}$$

3. Flow Constraint Equations
For node a1

$$e^0_{s\,a1}=e^k_{a1\,s} \tag{13}$$

For node b2

$$e^1_{s\,b2}+e^1_{c2\,b2}=e^2_{b2\,c2}+e^k_{b2\,s} \tag{14}$$

Similarly for nodes a2 b1 c1 c2.

4. Each service request should be satisfied For Service 0

$$e^0_{s\,a1}+e^0_{s\,a2}\geq 1 \tag{15}$$

$$e^1_{s\,b1} + e^1_{s\,b2} \geq 1 \tag{16}$$

$$e^2_{s\,c1} + e^2_{b2\,c2} \geq 1 \tag{17}$$

5. Loop Constraints

$$e^0_{s\,a1} = 1 \; and \; e^k_{a1\,s} = 1 => \\ e^k_{a1\;dummyend} + e^0_{s\,a2} + e^0_{s\,b1} + e^0_{s\,c1} + e^0_{s\,c2} = 1 \tag{18}$$

3. Experimental Results

The SA based approach for SplitSearch problem has been implemented in Java, currently addressing queries based on Bangalore geo-data. The database currently consists of geospatial information of points of interest belonging to 200 different service providers. The SA based algorithm is found to produce the output path for a SplitSearch problem in few milliseconds for real-life inputs of about 200 nodes and 20 requested services.

The algorithm has been tested on graphs with the parameters scaling upto 10000 nodes and 100 requested services, generated randomly. In all these test runs, SA was found to generate solutions of quality better than or equal to that obtained by Greedy approach.

Figure 2 shows the behaviour of the algorithm as the number of nodes increases, with number of requested services fixed at 20.

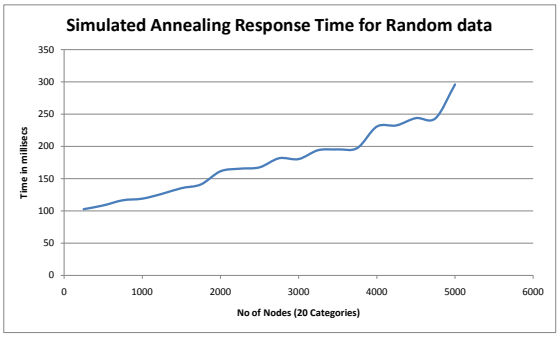

Figure 2: Scalability Graph

	Test Case	Size	Characteristic	Results
1	Uniform Distribution	4 Nodes 4 services	Every service had equal number of providers	Optimal comparable to Brute force
2	Biased Distribution	4 Nodes 3 services	One particular service had more number of providers	Optimal comparable to Brute force and ILP Solution
3	Biased Distribution	4 Nodes 2 services	One particular service had only one provider located far away from all nodes	Optimal comparable to Brute force and ILP Solution
4	Fooling Greedy	4 Nodes 4 services	Local optimum is not the global optimum	Optimal comparable to Brute force
5	Star Topology	5 Nodes 3 services	Every service provider is linked to the start node only	Optimal comparable to Brute force and ILP solution
6	Line Topology	4 Nodes 4 services	The optimum solution involved a zig-zag path	Optimal comparable to Brute force and ILP solution
7	Random Distribution	100 Nodes 10 services	Each service had random number of providers distributed randomly	Better solution than Greedy approach (ILP is intractable)
8	Random Distribution	1000 Nodes 100 services	Each service had random number of providers distributed randomly	Better solution than Greedy approach (ILP is intractable)
9	Random Distribution	500 Nodes 20 services	Each service had random number of providers distributed randomly	Better solution than Greedy approach (ILP is intractable)

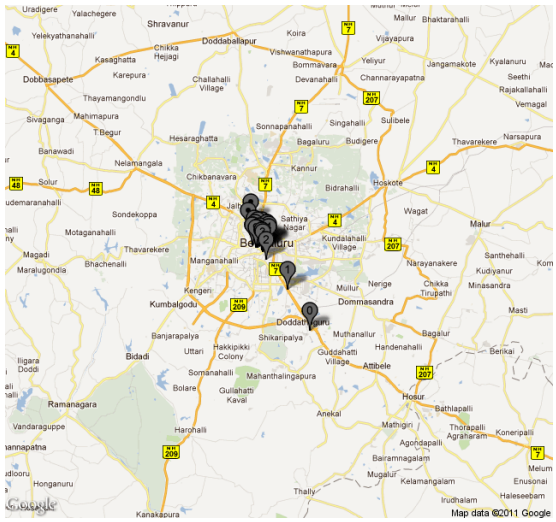

Figure 3: SplitSearch Output Map

The above table shows the scenarios the implementation is tested for(all these tests are for single attribute nodes i.e. each node satisfies one and only one request by the user).

As part of testing the quality of the solution given by the implemented algorithm, the linear program(lp) files for the same test graphs were generated and executed on Cplex.

The solution obtained from SA based approach was comparable to that provided by Cplex on solving the ILP for small-sized problems. For realistic problems with say, 10000 nodes, the ILP requires millions of decision variables making it intractable. However, Simulated Annealing based approach is scalable to handle such inputs.

Figure 3 shows the Output Path for SplitSearch as obtained from SA based approach.The output image shows the final solution comprising of nodes satisfying each of the 20 services requested by the user.

4. Conclusion

The current implementation of SplitSearch using Bangalore geo-database, works in the order of milliseconds for real time inputs of magnitude 200 nodes and 20 requested services. From the experimental data, the performance of Simulated Annealing based approach (using route or node set perturbation), is found to be almost 200% better than Greedy approach. The algorithm provides better quality solutions for optimal path than Greedy algorithm, especially in cases when the optimum collection of nodes is located far from the start node.

Future work involves working on Set Cover based approach for SplitSearch. Traditional Set Cover has no communication cost but SplitSearch includes edge costs as communication cost to find the Minimum Set Cover. Here, every node is a set providing a set of services and considering the path length between two nodes as communication costs, the solution would be set of nodes that satisfy all services requested by the user and has least cost.

Further enhancement to the algorithm is to consider mobile amenities such as bus fleet, ambulance, mobile fruit vendors, etc. Dynamic criteria such as routes based on traffic, load at the nodes providing one or more requested services (balance/queue at the ATM, queue/appointment schedule at the hospital, etc) can also be considered in future, while assessing the quality of the optimum path to be selected.

Appendix: Mathematical Analysis of SA

The most popular model assumed is a (time inhomogeneous) Markov chain. A Markov Chain is a discrete-time stochastic process in which the probability distribution of the state at time t+1 depends on the state at time t and does not depend on the states the chain had passed through on the way to time t. This is exactly what happens in SA, the solution is created from the previous state solution only and does not depend on what the states earlier to the previous state were.

Before going in detailed proof of convergence of SA, here is **a proof of convergence of Markov chains**.[3]

In the study of Markov Chains it is assumed that for all states i and j at all t,
$P(X_{t+1} = j | X_t = i)$ is independent of t.

$$P(X_{t+1} = j | X_t = i) = p_{ij} \tag{19}$$

where p_{ij} is the probability with which the system will go to state j at time t+1 if it is at state i at

time t. These are called transition probabilities.The above equation implies that the probability law relating the next state to the current state does not change over time. However, in case of SA with time the temperature changes and thus the transition probabilities change. At high temperatures some transitions can be accepted (to get out of local minima) but at lower temperatures, the acceptance probabilities for these transitions decrease drastically.

In Markov Chains, given that the process state at time t and t + 1 is i and j respectively, it can be stated that for each i,

$$\Sigma P(X_{t+1} = j | X_t = i) = 1 \quad (20)$$

$$\Sigma p_{ij} = 1 \quad (21)$$

All entries in the transition probability matrix are non-negative and the above equation implies that entries in each row must sum to 1(from some state, state j is reached).

Given if a Markov chain is in state i at time m, then what is the probability that n periods later the Markov chain will be in state j?

$$P(X_{m+n} = j | X_m = i) = P(X_n = j | X_0 = i) = P_{ij}(n) \quad (22)$$

From definition of transition probability $P_{ij}(1) = p_{ij}$.
Now for $P_{ij}(2)$, if the system is now in state i, then for the system to end up in state j two periods from now, it must go from state i to some state k and then go from state k to state j.

$$P_{ij}(2) = \Sigma(p_{ik} * p_{kj}) \quad (23)$$

which is just the scalar product of row i of the P matrix with column j of the P matrix. Hence, P_{ij} (2) is the $ijth$ element of the matrix P^2.

Similarly,

P_{ij} (n) = $ijth$ element of P^n

Also, probability of being in state j at time n is Σ (probability that state is originally i) x (probability of going from i to j in n transitions).
For large n, P^n approaches a matrix with identical rows. This means that after a long time, the Markov chain settles down. And this is independent of the initial state i.

Now for the **Mathematics behind SA**,
The objective is to minimize cost function J defined on a space E. This algorithm creates a Markov chain X_n on E in the following way(Θ is temperature) If X_n is given, then at random choose a neighbour x of X_n,

compute ΔJ = J(x) - J(X_n),
if $\Delta J < \Theta * Z_n$, (where Z_n is exponential variable and J(x) is the cost function) the transition is accepted and X_{n+1} = x, otherwise $X_{n+1} = X_n$.

If the state with the minimum cost visited so far is tracked by the Markov chain, the objective will be achieved.
Assuming the Markov chain is irreducible and aperiodic, then X_n is reversible Markov chain and its invariant probability distribution is given by

$$\pi(i) = \frac{1}{Z_T} exp \frac{-J(i)}{T} \quad (24)$$

where Z_T is normalizing constant and T is current temperature.[4] This shows when T approaches zero the probability distribution is concentrated on global minima of J. This probability distribution is known as Gibbs distribution. The algorithm will converge if limit t tends to infinity, i.e. prob[X(t)] = 1.

Theorem by Hajek
A state i communicates with E at height h if there exists a path that starts from i and ends at some element in E such that the largest value of J along the path is J(i) + h. Let d^* be smallest number such that every i in E communicates with E at height d^*. Then SA converges if limit at t tends to infinity, T(t) = 0 and Σ exp[-d^*/T(t)] = infinity.

The cooling schedule is of the form T(t) = d/log(t) (for SA). So the theorem states SA converges if d≥d^*. The constant d^* is a measure of the difficulty for X(t) to escape local minimum. Consider a local minimum of depth d^*. The SA makes an infinite number of trials to escape from it and the probability of success at each trial is of the order of exp[-d^*/T(t)]. Now as Σ exp[-d^*/T(t)] = infinity, so an infinite number of such trials will be successful.

For a better convergence, consider the cooling schedule. The schedule T(t) = d/log(t) can be approximated as, let t_1 = 1 and $t_{k+1} = t_k$ + exp(kd). Let T(t) = 1/k for $t_k \leq t \leq t_{k+1}$. If the cost function J has unique global minimum the relaxation time is approximated by exp(kd^*). This yields another interpretation of convergence condition d ≥ d^* for the schedule T(t).
If d < d^* then at each temperature 1/k we cannot be sure that X_{n+1} is close to X_n but at d > d^*, the

relaxation time is exp[k(d^*-d)] which implies at k tends to infinity X_{n+1} is close to X_n.

The reason why SA works well -
If a given state is far from optimal then there exists a large number of paths(candidate paths) that lead to another state with lower costs. The probability that the state X(t) escapes local minimum of depth d along any particular path, in single trial, is at most exp(-d/T). On the other hand if the number of candidate paths is very large the probability of escapes is substantial. So constant temperature SA would work in polynomial time if the relaxation time were poly-logarithmic in the size of the state space.

Acknowledgments

We would like to extend our sincere gratitude to our guide Prof. G N Srinivasa Prasanna for providing us with an opportunity to work on the project - A Simulated Annealing based approach for solving SplitSearch. We are heartily thankful to him for his constant encouragement, valuable sug- gestions and the motivation that we got from him. The completion of the paper would not have been possible without the constant feedback, guidance and motivation from our professor.

References

[1] J. M. P. V. S Kirkpatrick, C D Gelatt, "Optimization of simulated annealing," *Science*, vol. 220, no. 4598, 1983.

[2] R. Carr, "Simulated annealing from mathworld a wolfram web resource, created by eric w. weisstein."

[3] W. Winston, "Operations research applications and algorithms," vol. 4th Edition, 2003.

[4] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statistical Science*, vol. 8, no. 1, pp. 10–15, 1993.