

Optimizing the Placement of Chemical and Biological Agent Sensors

Daniel L. Schafer

Thomas Jefferson High School for Science and Technology

Defense Threat Reduction Agency TDOA OR

Mentor: George D. Gunn, PhD

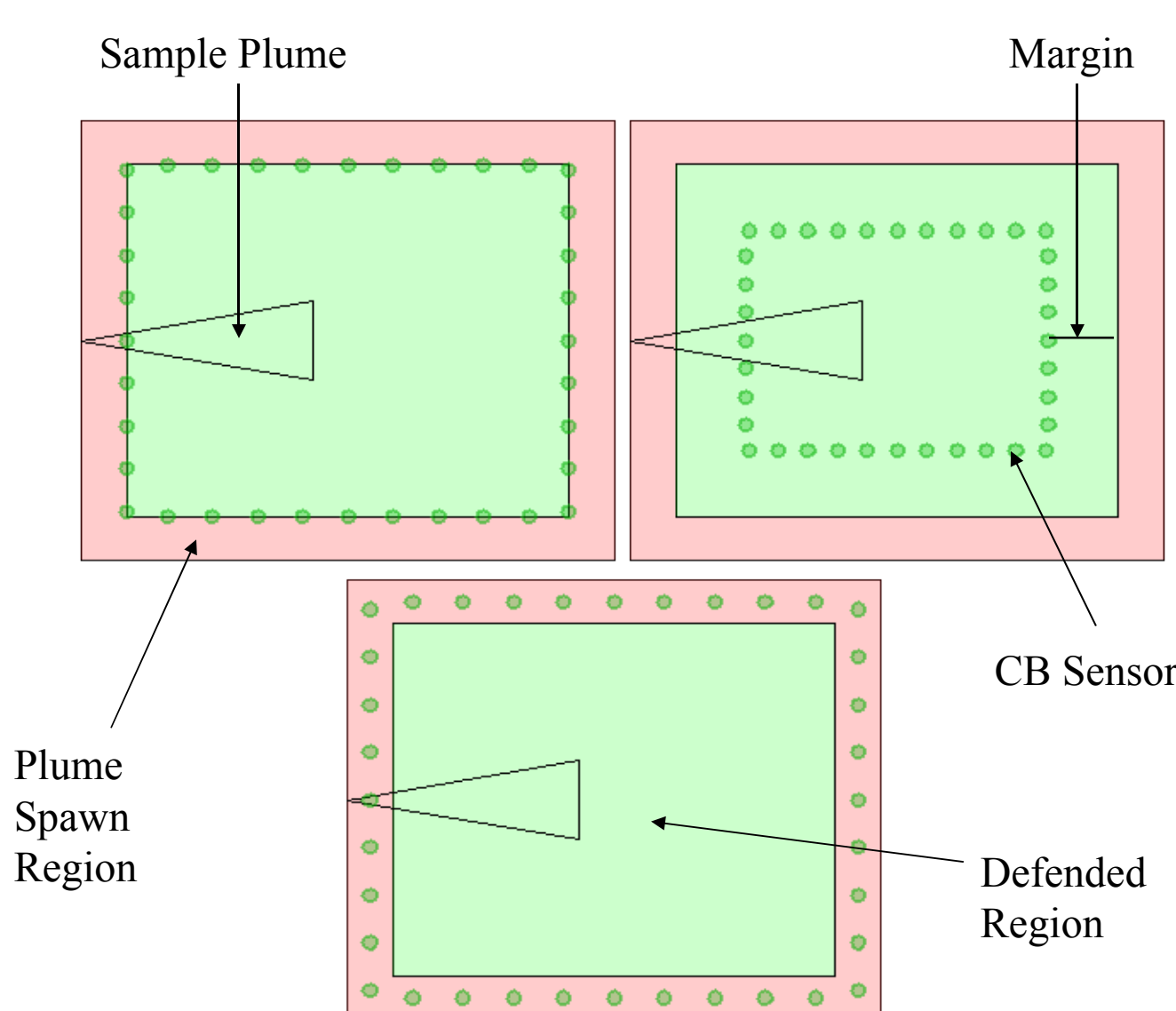
Abstract

A computer program was designed and implemented in Java to optimize the placement of chemical and biological (CB) agent sensors to best protect an installation. This was accomplished through the use of various optimization algorithms, combined with a Monte Carlo simulation method which determined relative utility functions for various sensor arrangements.

Purpose

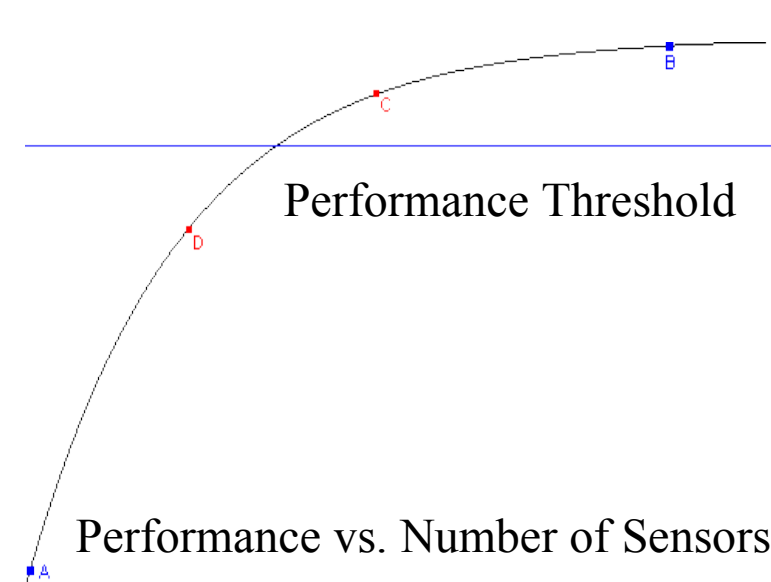
Weapons of Mass Effect pose a threat to any facility, especially military targets. Sensors exist that can detect biological and chemical assaults, allowing the base to either avoid the attack or treat any affected individuals and equipment. It is necessary to determine the most effective sensor placement to ensure that a base is well protected; unfortunately, as many of the tools that simulate such incidents are extremely detailed and complicated, running simulations on them to determine optimal sensor placement can take a prohibitively long amount of time.

The software to quickly simulate many attacks existed in a C++ program, which randomly placed 500 potential biological attacks on a target to determine how effective the sensor arrangement was at detecting the attacks. A new program was developed, which used the basic methodology of the original program to automatically sample different sensor configurations until an optimal one could be determined.



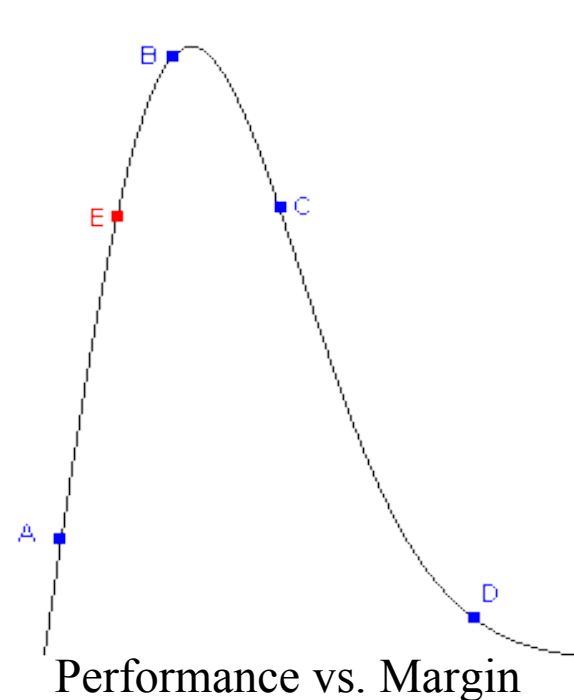
Minimizing Sensors

This method minimizes the number of sensors needed to keep the performance level above a certain threshold, using a Binary Search. (Graph from GCalc.net)

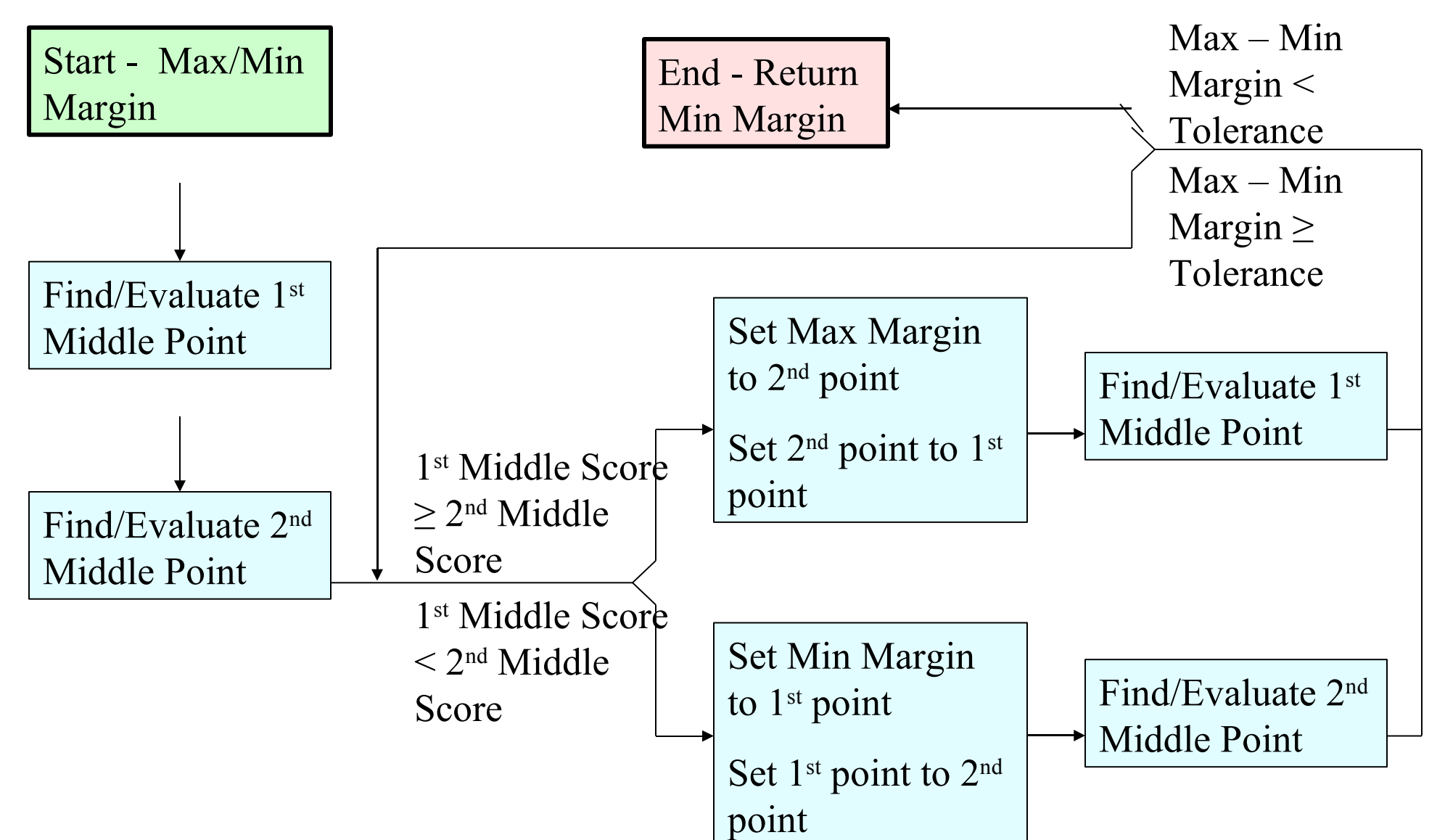
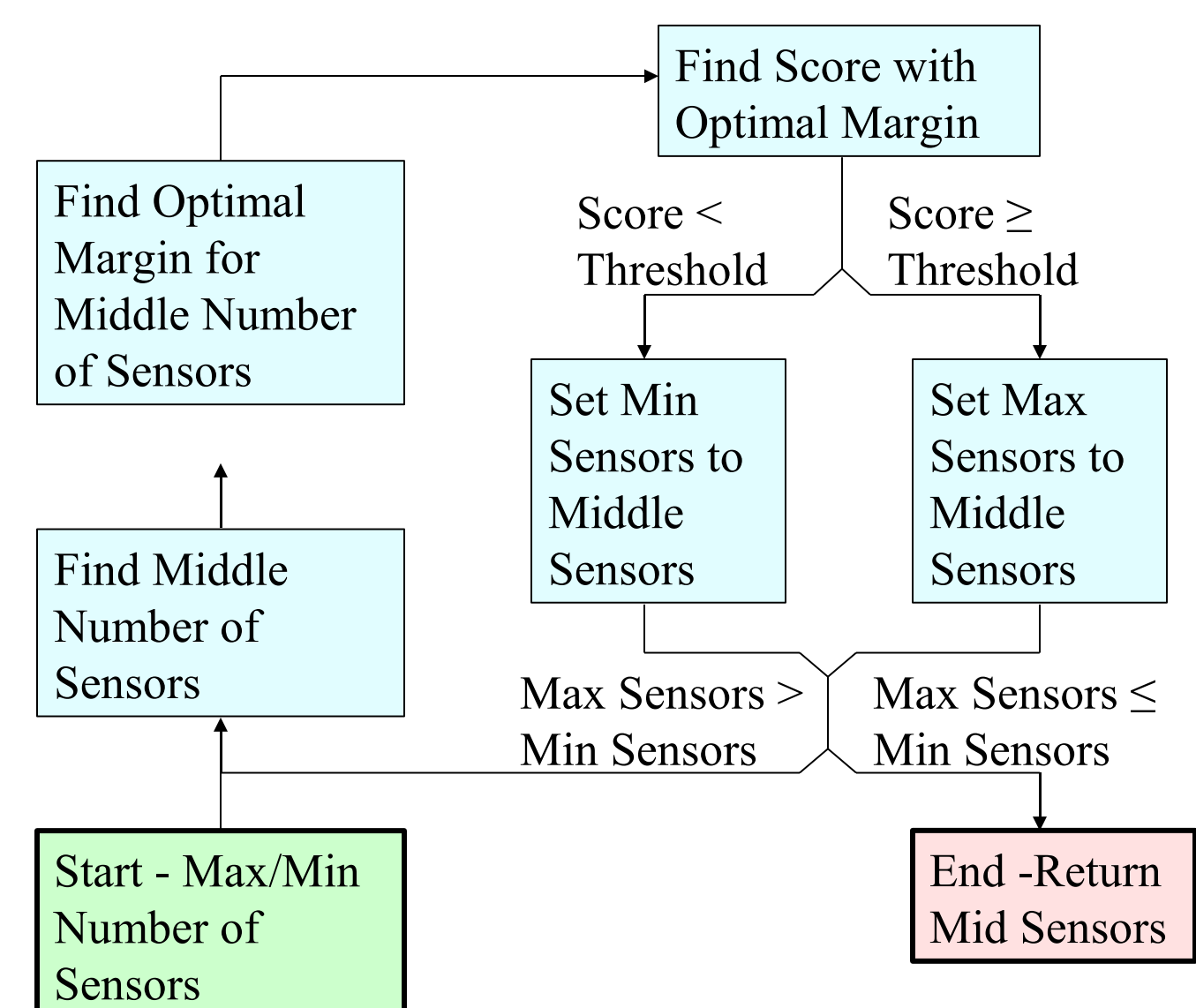


Optimizing Margin

This method finds the margin that maximizes the performance of a sensor grid for a certain number of sensors, using a Golden Section search. (Graph from GCalc.net)



Procedures



Data

Optimization Results (Small Plume)

Scoring Method	Placement	Allocation	
		Large Threshold	Small Threshold
1 One Hit	(7,8) 56 sensors (Uniform) -1.019 margin. Score: 0.96	(6,5) 27 sensors (Dice-5) 0.056 margin. Score: 0.778	(3,5) 15 sensors (Uniform) 1.038 margin. Score: 0.522
2 Multi-Hit	(8,8) 60 sensors (Dice-5) -0.496 margin. Score: 0.744	(7,6) 42 sensors (Uniform) 0.0040 margin. Score: 0.502	(5,7) 35 sensors (Uniform) 0.366 margin. Score: 0.32
3 Area-Weight	(7,9) 60 sensors (Dice-5) 0.238 margin. Score: 0.972	(6,7) 39 sensors (Dice-5) 1.251 margin. Score: 0.802	(5,6) 30 sensors (Uniform) 2.927 margin. Score: 0.638
4 Power Law	(7,8) 56 sensors (Uniform) 1.038 margin. Score: 0.637	(5,7) 35 sensors (Uniform) 1.485 margin. Score: 0.505	18 sensors (Perimeter) 2.152 margin. Score: 0.306

Optimization Results (Large Plume)

Scoring Method	Placement	Allocation	
		Large Threshold	Small Threshold
1 One Hit	59 sensors (Perimeter) -1.104 margin. Score: 0.998	32 sensors (Perimeter) -0.415 margin. Score: 0.984	23 sensors (Perimeter) -1.104 margin. Score: 0.954
2 Multi-Hit	60 sensors (Perimeter) -0.542 margin. Score: 0.955	37 sensors (Perimeter) -0.496 margin. Score: 0.907	26 sensors (Perimeter) -0.629 margin. Score: 0.813
3 Area-Weight	60 sensors (Perimeter) 0.349 margin. Score: 1.0	24 sensors (Perimeter) 0.56 margin. Score: 0.995	16 sensors (Perimeter) 1.596 margin. Score: 0.908
4 Power Law	59 sensors (Perimeter) -0.204 margin. Score: 0.769	28 sensors (Perimeter) 0.877 margin. Score: 0.711	18 sensors (Perimeter) 0.268 margin. Score: 0.602

The above data was generated using default program settings, with the plumes all spawning within 2 km of the 16 km x 19 km base. 2500 trials were done instead of the default 500, to ensure accurate results.

Results and Conclusions

The data gathered through the optimizations showed that the length of the plume relative to the size of the base played a major role in deciding which sensor arrangement would be best. In the initial optimization efforts, uniform and Dice-5 grids were best against a small plume. However, perimeter arrangements were clearly superior for the large plume situations, where the plume would undoubtedly stretch across the entire base. In these cases, the plume was small enough to go undetected when it reaches the first edge, but it was quite large when it reached the second edge, and a perimeter setup had many sensors in the part of the second edge covered.

Data in a plume spawn region analysis showed a similar trend. With the large plumes, a perimeter setup was preferred until the plume spawn region began getting large. Once the plume spawn region reached a large enough size, the plumes were not able to entirely cross the defended area, and the preferred grid changed to uniform. With the small plumes, it is apparent that regardless of the size of the plume spawn region, the plumes are never able to entirely cross the defended area, and thus the small plumes nearly always preferred the uniform or Dice-5 configurations. The one exception was using a non-area-weighted scoring method with a large plume spawn region. Here, many plumes spawned far from the base, and thus were quite large when they reach even the first edge, meaning that they were detected by many perimeter sensors. However, these plumes did not cover much of the base, which explains why they had little effect on the area-weighted algorithm's results.

After analyzing the program's speed and efficiency, it is clear that the design requirements for the program were met. The program is able to complete an entire analysis in a reasonable amount of time, and is also able to perform different types of optimizations depending on the situation. The program is also flexible enough to deal with any sort of input parameters, allowing it to be used on many different bases.