JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS
Vol. 38, No. 7, July 2015

Engineering Notes

Computational Solutions to Lambert's Problem on Modern Graphics Processing Units

Sam Wagner[*] and Bong Wie[†]
Iowa State University, Ames, Iowa 50010
and
Brian Kaplinger[‡]
Florida Institute of Technology, Melbourne, Florida 32901

DOI: 10.2514/1.G000840

I. Introduction

THE problem of determining an orbit from two position vectors and a specified time of flight, known as Lambert's problem, is one of the most fundamental problems in astrodynamics [1–8]. Historically, solutions were needed to calculate the orbital elements of planets, comets, and other solar system bodies from observations. In the 18th century, the problem of computing a comet's trajectory from observations was a topic of discussion by nearly every eminent astronomer and mathematician. John Henry Lambert (1728–1777) proposed that the orbit depends only on the chord, sum of the two radii, and semimajor axis, which is now known as Lambert's theorem. With the advent of human spaceflight (including robotic spaceflight) and the importance to preliminary orbit determination, solutions to Lambert's problem have continued to be an area of active research interest in recent decades.

Today, solutions to Lambert's problem are used extensively in the orbital targeting problem, specifically in areas such as rendezvous analysis, missile targeting algorithms, and preliminary orbit determination. With the growing complexity of robotic space missions, such as NASA's Cassini, Galileo, Messenger, and OSIRIS REx missions, it is necessary to determine efficient and robust Lambert solution algorithms. In this paper, four modern Lambert solution methods, with any necessary modifications for the CUDA graphics processing unit (GPU) computing architecture requirements, will be examined for efficiency, robustness, and accuracy.

With the advent of modern GPUs, massively parallel computations are now more widely accessible, enabling tens of millions of solutions to Lambert's problem to be evaluated per second. These computational capabilities will allow increasingly complex missions to be designed and flown in the future without the need for expensive supercomputers. Several solution implementations to Lambert's problem have been examined to determine the most efficient and robust algorithms for the purpose of preliminary mission analysis. For this study, all of the algorithms have been developed using Fortran and CUDA Fortran [9].

Various solutions to Lambert's problem can be found throughout literature [5–7,10]. A total of four modern formulations will be tested and evaluated in this paper. The methods evaluated in this paper are the classical universal variable method [5,7,10], Battin's alternate approach to Gauss' original method [6,7], and formulations by Lancaster and Blanchard [2], Gooding [3], and Sun [11]. These methods were chosen because they represent recent advancements in Lambert's problem solutions and are among the most robust solution algorithms.

The search for preliminary mission trajectories often requires Lambert's problem to be solved tens of millions of times. The object of this paper is to determine the most efficient and robust solution implementations. This includes initial guess generations schemes and root-finding methods for methods for the Sun and universal variable methods. Because individual solutions to Lambert's problem are completely independent, the initial orbit determination problem lends itself well to parallel programming, especially when implemented directly on modern GPUs. Before examining the specifics of any particular method, a brief introduction to the problem is presented. The initial and final radius vectors and time-of-flight are given respectively as $\boldsymbol{r}_1$, $\boldsymbol{r}_2$, and Δt, with the two radius vector magnitudes represented as r_1 and r_2. The chord c and semiperimeter s are also needed for each of the solution methods. The chord is simply the distance between the two radius vectors, and the semiperimeter is half the sum of the triangle formed by the chord and radius vectors, defined as follows:

$$c = |\boldsymbol{r}_2 - \boldsymbol{r}_1| = \sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos\theta} \tag{1}$$

$$s = \frac{r_1 + r_2 + c}{2} \tag{2}$$

A method to determine the transfer angle θ without quadrant ambiguity is described next [10]. This method replaces the standard long way and short way terminology that is often used when describing solutions to Lambert's problem [10]. The transfer angle θ for a prograde orbit is determined as follows:

$$\theta = \begin{cases} \cos^{-1}\left(\frac{\boldsymbol{r}_1\cdot\boldsymbol{r}_2}{r_1 r_2}\right) & \text{if } (\boldsymbol{r}_1 \times \boldsymbol{r}_2)_k \geq 0 \\ 360\ \text{deg} - \cos^{-1}\left(\frac{\boldsymbol{r}_1\cdot\boldsymbol{r}_2}{r_1 r_2}\right) & \text{if } (\boldsymbol{r}_1 \times \boldsymbol{r}_2)_k < 0 \end{cases} \tag{3}$$

where the subscript k indicates the out-of-plane component of the cross product. The transfer angle for prograde orbits is calculated in a similar manner [10].

The time to traverse an arc between two points is related to the transfer orbit by Kepler's time equation. From Kepler's equation, Lagrange developed a proof of Lambert's theorem. The proof, which is briefly summarized here, has been the basis of nearly every proposed solution to Lambert's problem. Lagrange's equation removes the orbit eccentricity from Kepler's time-of-flight equation and is only a function of $r_1 + r_2$, c, and the semimajor axis a, as given by

$$\sqrt{\frac{\mu}{a^3}}\Delta t = (\alpha - \beta) - (\sin\alpha - \sin\beta) \tag{4}$$

The two angular parameters α and β are defined in terms of the two physical parameters c and s and the semimajor axis a as follows:

Received 9 July 2014; revision received 31 October 2014; accepted for publication 3 November 2014; published online 28 January 2015.

*Graduate Research Assistant, Department of Aerospace Engineering, Asteroid Deflection Research Center.

†Vance Coffman Endowed Chair Professor, Department of Aerospace Engineering, Asteroid Deflection Research Center.

‡Assistant Professor, Department of Mechanical and Aerospace Engineering.

$$\sin\frac{\alpha}{2} = \pm\sqrt{\frac{s}{2a}} \tag{5}$$

$$\sin\frac{\beta}{2} = \pm\sqrt{\frac{s-c}{2a}} \tag{6}$$

II. Solution Methods to Lambert's Problem

Solutions to Lambert's problem are formulated using Lagrange's time-of-flight equation [Eq. (4)]. In this section, the four Lambert solutions are briefly introduced. Further details for individual solutions to Lambert's problem can be found in the referenced literature. For the case of the universal variable and Sun method, an initial guess scheme for the root-finding routines will be introduced that improve algorithm efficiency and robustness over many initial guess generation schemes commonly found in the literature. Throughout this section, each individual solution follows the notations that are common to the individual solutions in the literature.

A. Battin's Method

The first Lambert solution method considered, commonly known as Battin's method, was first proposed in the 1980s and is perhaps the most mathematically rigorous algorithm [6]. This solution is similar to Gauss's original solution except that it moves the singularity from 180 to 360 deg and dramatically improves convergence when θ is large. Just as Gauss's solution does, Battin's method works for all types of orbits (elliptical, parabolic, and hyperbolic).

Battin's solution algorithm is a successive substitution algorithm, which extensively uses continued fractions to successively substitute the value for $y(x)$ into the definition of the independent variable x. This is done until the change in x is within a specified tolerance. Further details can be found in [6,7]. An outline of the final algorithm is shown next.

1) Inputs are dimensionless time-of-flight parameter T and the Lambert parameter λ.

2) Compute the dimensionless parameters λ, T, ℓ, m, and r_{0p}.

3) From the parabolic time-of-flight T_p, determine x_0.

4) In the following order, calculate η, ξ, h_1, h_2, B, u, and $K(u)$.

5) Compute the solution for y and the updated x.

6) Go to step 3 and repeat until the desired tolerance for the change in x is met or the maximum number of iterations is exceeded.

7) Output the converged semimajor axis a or the initial and final velocities with the Lagrange coefficients.

B. Gooding's Method

The second Lambert solution method considered is known as Gooding's method [3,12]. This method is an extension of Lancaster and Blanchard's unified form of Lambert's theorem developed in the 1960s [2]. Gooding was able to formulate robust initial guesses as well as derivatives for a high-order root-finding routine. The final version of the algorithm developed for this study differs slightly from the algorithms presented in Gooding's papers [3,12] but can provide approximately 100% convergence. This method will also be shown to be one of the most computationally efficient algorithms.

This method relies on a high-order Halley root-finding algorithm [13], which requires up to third-order derivatives. A universal formulation is used that allows the same time equations and the accompanying derivatives T, T', and T'' to be used for both elliptical and hyperbolic orbits. However, when the orbits are near-parabolic, these equations suffer from a loss of accuracy. For the near-parabolic case, a second formulation is used, which is originally given in transcendental form by Lancaster and Blanchard [2] and series form by Gooding [3]. For the final algorithm, both equation forms were tested, and both forms perform approximately the same.

An accurate initial guess generation scheme was developed by Gooding, which uses a bilinear approximation. Initial guess approximations are given for zero-revolution hyperbolic and elliptical orbits as well as multiple revolution cases. Combined with the high-order Halley root-finding method, this makes Gooding method extremely robust and efficient. An outline of the final Gooding solutions method algorithm is shown next.

1) The inputs are the dimensionless time-of-flight T and the Lambert parameter q.

2) Evaluate T_0 when $x = 0$ and determine the initial guess as defined by Gooding [3] and Lancaster and Blanchard [2].

3) If x is close to 1.0, calculate E and K to evaluate T, T', and T'' from the transcendental equations [2] or the series solution [3].

4) Otherwise calculate z, d, y, and E and evaluate T, T', and T'' as given in [3].

5) Update x using Halley's method [13].

6) Go to step 2 and repeat until the desired tolerance for the change in x is met or a maximum number of iterations is exceeded.

7) Output the converged semimajor axis a or the initial and final radius vectors.

C. Sun's Method

The Sun method [11] is among the most robust and efficient Lambert solvers available. The formulation used by Sun is similar to the Gooding algorithm. For this solution, the choice of the Lambert parameter σ, time parameter τ, and independent x variables are comparable to the formulation proposed by Lancaster and Blanchard [2]. However, the final form of the Lagrange time equation differs significantly. This method has been improved upon here by using the high-order Halley root-finding method and by an initial guess generation scheme that is simple and robust.

As with Gooding's solution, Halley's method was used for the root-finding routine in the Sun solution algorithm. Other high-order root-finding methods were tested, such as Laguerre's method [14], but none were able to obtain better performance, in terms of the algorithm's efficiency and robustness.

The initial guess scheme for this algorithm requires computing both the parabolic and minimum energy time equations. The final initial guesses for all orbit types are formulated as follows:

$$x_0 = \begin{cases} 0.5 & \text{if } \tau < \tau_{\mathrm{ME}} \text{ elliptical} \\ 0 & \text{if } \tau \approx \tau_{\mathrm{ME}} \text{ elliptical} \\ -0.5 & \text{if } \tau > \tau_{\mathrm{ME}} \text{ elliptical} \\ 1 & \text{if } \tau \approx \tau_P \text{ parabolic} \\ 3.0 & \text{if } \tau > \tau_P \text{ hyperbolic} \end{cases} \tag{7}$$

With this notation, τ_p and τ_{ME}, correspond to the parabolic and minimum energy time equations, respectively. Using the defined initial guesses and the Halley root-finding algorithm, the final Sun solution algorithm is able to obtain nearly equivalent performance to Gooding's method. The final Sun solution algorithm is outlined next.

1) The inputs are the dimensionless time-of-flight τ and the Lambert parameter σ.

2) From σ, calculate τ_p and τ_{ME} and determine the initial guess as described by Sun.

3) Evaluate τ, τ', and τ'' from [11].

4) Update x using Halley's method.

5) Go to step 2 and repeat until the desired tolerance for the change in x is met or a maximum number of iterations is exceeded.

6) Output the converged semimajor axis, a, or the initial and final velocity vectors.

D. Universal Variable Method

The last method tested in this study is commonly known as the universal variable method. This method has been one of the most extensively studied and published in recent decades [5,7,10,14,15]. The independent variable is chosen so that the final time-of-flight equation applies to all orbits.

As with the Sun algorithm, Halley's root-finding method was determined to be the most robust and efficient root-finding method for the universal variable algorithm.

The universal variable formulation suffers from one major drawback, which is noted here. When the transfer angle θ is less than

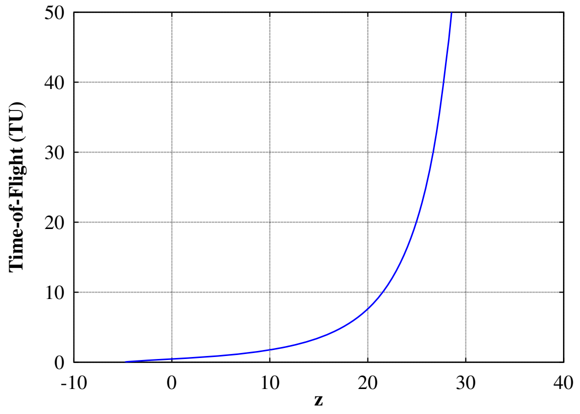

Fig. 1 **Time-of-flight (in scaled canonical units) vs** z**.**

π and the universal variable z becomes a large negative number, the final time-of-flight equation will result in an imaginary solution. These orbits essentially correspond to orbits that are too hyperbolic. This imaginary number in the solution causes the time equation to intersect with the $t = 0$ axis with orbits that are too hyperbolic [5]. Any algorithm implementing this method should check for imaginary numbers in the universal variable time-of-flight equation or variable y. If this situation occurs, the algorithm should be terminated, and an alternative solution to Lambert's problem should be used.

For initial guess generation, the specified time-of-flight Δt can be compared with the parabolic time-of-flight to determine whether the orbit is elliptical, near-parabolic, or hyperbolic. As shown in Fig. 1, the time curve is strictly monotonically increasing, and so derivative-based root-finding methods are well behaved. For zero-revolution elliptical orbits, z has an upper bound of $4\pi^2$. An initial guess of $2\pi^2$ for elliptical orbits has been shown to work well. If the orbit is near-parabolic (z is close to 0), an initial guess of 0 is used, whereas an initial guess of $-\pi/32$ works well for hyperbolic orbits. With the algorithm, shown next, convergence typically occurs in three to five iterations.

1) From the inputs r_1, r_2, θ, and Δt, calculate A.

2) From the parabolic time-of-flight ($z = 0$), determine the initial guess.

3) Evaluate S, S', S'', C, C', C'', x, x', x'', y, y', and y'' to calculate F, F', and F''.

4) Update z with Halley's method.

5) Go to step 3 and repeat until the desired tolerance for the change in z is met or a maximum number of iterations is exceeded.

6) Output the converged semimajor axis or the initial and final velocity vectors.

III. Results

For the four Lambert algorithms, two tests were developed to demonstrate the robustness and efficiency of each algorithm. The first test is designed to demonstrate the robustness and efficiency of each algorithm. For this study, robustness refers to how often an algorithm converges on a solution, and efficiency refers to the computational cost associated with solutions for each solution method. The second test is designed to determine the efficiency of each algorithm, on both CPU and GPU computational architectures. This is done by evaluating an example 25 year search for a direct mission to Mars. This mission scenario is representative of the type of initial orbit determination problem typically performed with Lambert searches. Details of the CPU and GPU workstations used to perform the tests can be found in Table 1. The CPU tests were all run on a standard Dell workstation, whereas the GPU tests were performed on an Amax workstation with an NVIDIA Tesla C2050 GPU. The second test is designed to further test each algorithm's efficiency by determining the number of iterations each algorithm requires to converge for a series of four test scenarios. Although the results of this paper are similar to some of the author's previous work in GPU Lambert computations [16], a performance gain of approximately 50% over the previous work has been realized by improving both the CUDA implementation and the Lambert solution algorithms themselves.

Table 1 **Information on the workstation used in the study**

	Workstation	GPU workstation
Model	Dell T3500 Workstation	Amax ServMax Tesla GPU HPC
Operating system	Windows Vista Professional 64 bit	RHEL v5.4
Processor	Intel Xeon W3520 2.66 GHz	2x Intel Xeon LGA1366 2.26 GHz
Memory	6 GB 1066 MHz DDR3	32 GB 1333 MHz DDR3 4x Tesla C2050
GPU(1)	N/A	448 CUDA cores 1.15 GHz
Theoretical peak double-precision Gflops	42 (CPU)	515 (GPU)

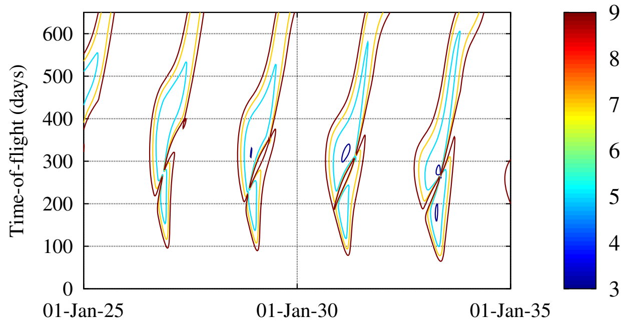

Fig. 2 **50 year porkchop plot of departure** V_∞ **for Mars mission.**

A. Testing the Computational Efficiency and Robustness of Each Algorithm

All of the three search implementations are written in Fortran or CUDA Fortran [9] and have nearly identical implementations. Keeping the search configurations as close as possible ensures that the efficiency of both the CPU and GPU algorithms can be directly compared. The first implementation is standard single-thread Fortran; the second is parallel Fortran with OpenMP [17]; and the third is implemented using PGI's CUDA Fortran [9]. The GPU implementation was written to use a single GPU, in this case an NVIDIA Tesla C2050. OpenMP is an application programming interface design for shared memory multiprocessing on a single machine, whereas CUDA is a parallel programming model design by NVIDIA for use with GPUs. Both of these models are designed to help create algorithms that can use multiple processing cores included in today's CPUs and GPUs.

Solutions to Lambert's problem can be evaluated in parallel because each solution to Lambert's problem is independent. In this case, both standard serial and parallel searches for an Earth-to-Mars transfer orbit are tested. All three versions of the algorithm execute a grid search of launch dates and times of flight, requiring approximately 250 million solutions. This search ensures saturation of the GPU while keeping CPU run times low enough for testing purposes.

The final grid search is for a 25 year period, with launch dates from 2025 to 2050 and a maximum time-of-flight of 1000 days. By using a grid search time step of 0.191 days exactly, 250,002,660 Lambert's problem solutions are required. The resulting porkchop plot, which is a contour plot of the departure V_∞, with contour levels ranging from 3 to 9 km/s, is shown in Fig. 2.

The efficiency of each algorithm can be determined by monitoring the number of times each algorithm fails to converge during the search. The initial orbit determination and orbit optimization problems require robust solutions to Lambert's problem. Therefore, only algorithms that are both extremely robust and efficient are suitable for such problems. From Table 2, it can be seen that the

Table 2 Number of failures for the Earth-to-Mars mission search

Algorithm	Number of failed solutions	Failed %
Battin	0	0.00E + 00
Gooding	0	0.00E + 00
Sun	23	9.20E − 06
Universal variable	80,219	3.20E − 01

Table 3 Run times (in seconds) for each solution test search

Algorithm	Serial Fortran	OpenMP	GPU
Battin	1,482.90	270.52	11.76
Gooding	837.74	159.59	8.40
Sun	845.93	164.93	7.85
Universal variable	840.19	206.23	9.17

Table 4 Algorithm performance increases when compared to serial and parallel Fortran

	Efficiency vs serial Fortran		Efficiency vs OpenMP
Algorithm	OpenMP	GPU	GPU
Battin	5.48	126.14	23.01
Gooding	5.25	99.69	18.99
Sun	5.13	107.70	21.00
Universal variable	4.07	91.65	22.50

Battin, Gooding, and Sun algorithms converge nearly 100% of the time, whereas the universal variable algorithm fails to converge approximately 0.32% of the time. A pure universal variable approach fails more often than this because of a limitation to the problem formulation, which results in solutions with imaginary numbers for orbits that are too hyperbolic. However, this is a known limitation to the universal variable solution [5]. When this situation is encountered, the universal variable algorithm uses the Gooding–Lambert algorithm to determine a solution. For this test, approximately 2.4 million solutions resulted in orbits that were too hyperbolic, or approximately 0.9% of solutions. Although this is a relatively small percentage, it is five orders of magnitude higher than the failure percentage of the Sun algorithm, which had the next-highest failure rate. The Sun algorithm failed a total of 23 times, all of which were solutions with nearly exactly parabolic orbits, corresponding to solutions that have an x value very close to 1.0. This failure suggests that, as with Gooding's method, near-parabolic orbits would improve with an alternative formulation for the time equation and the accompanying derivatives. Of the four algorithms, only Battin's and Gooding's algorithms converged for 100% of solutions.

B. Graphical Processing Unit Performance

The CPU and GPU run times for each version of the search algorithm for all four Lambert algorithms are shown in Table 3. For the serial implementation, Battin's method had the longest run times, at approximately 1500 s, whereas the other three algorithms all had run times of approximately 840 s.

Differences in the four solution algorithms become evident in both OpenMP and CUDA Fortran parallel implementations of the search program. For the OpenMP algorithms, Battin's method still produced the longest total run time at 270 s. The universal variable method shows the least speed-up, with a run time of approximately 200 s. With the OpenMP parallel search, both Gooding's and Sun's methods have run times of approximately 160 s. This represents a relative increase in speed of approximately 25% increase when compared to the universal variable method. The Gooding and Sun algorithms converge within a very predictable number of iterations, typically no more than three and four iterations, respectively, whereas the universal variable method can take anywhere between two and nine iterations. This large variation in the number of required iterations imposes some limitations on the speed increases possible with parallel algorithms. The low speed increase of the universal variable method is likely due to the large range of the number of required iterations, which will be discussed in further detail with the second test. This large variation in number of required iterations causes processor scheduling issues, which force the search program to wait for the slowest solution in a set given to the processor.

The kernel developed for the CUDA search algorithm will suffer from some of the same processor scheduling issues as the OpenMP parallel algorithm. This effect is minimized by adjusting the block size and number of threads on which the kernel is executed to ensure maximum performance of the algorithm. The final CUDA Fortran algorithms were able to process the entire 250 million Lambert solutions in just a few seconds. All four algorithms had closer performance on the GPU than on either CPU implementation. As with the CPU search implementations, Battin's algorithm had the longest average run times at 11.76 s. Although the Gooding algorithm was slightly faster than the Sun method on both CPU versions, it was approximately 6.5% slower on the GPU. With a run rime of 9.17 s, the universal variable method was 14% slower than the Sun algorithm, whereas the Battin algorithm was 33% slower.

A summary of each algorithm's relative performance increase for the three grid search programs can be found in Table 4. In general, the GPU search algorithms were approximately 100 times faster than standard Fortran, and 20 times faster than the parallel OpenMP versions. Battin's algorithm saw the largest performance increase when compared with both the serial and OpenMP at 126 and 23 times, respectively. When compared with standard Fortran, the Gooding, Sun, and universal variable CUDA versions had approximate speed increases of 100, 107, and 92 times, respectively.

As indicated by Table 1, the Tesla C2050 GPU has a peak theoretical performance of 515 Gflops, whereas the CPU for the workstation used has a peak theoretical performance of 42 Gflops (for four cores). If both algorithms are able to fully use peak performances, the GPU implementation should be approximately 12 times more efficient than the OpenMP CPU implementation. Typical increases of the GPU implementations of approximately 20 times, when compared to the OpenMP versions, indicates that the CUDA implementation is more efficient at using the parallel processing capabilities of the GPU than the OpenMP implementation is at using the full parallel capabilities of the CPU. If both could use 100% efficiency, the CUDA algorithm performance increase should only be approximately 12 times when compared with OpenMP.

Ultimately, the most important performance aspect of any version of the algorithm is how many solutions per second can be obtained. This is true for any initial orbit determination or mission optimization algorithm. On the GPU, Sun's method was the best algorithm, with a maximum of nearly 32 million solutions per second. Gooding's method was able to achieve approximately 30 million solutions per second, whereas the Battin and universal variable algorithms obtained 21 and 27 million solutions per second, respectively. It should also be noted that the GPU results do not include the time it takes to transfer any data to and from the GPU. However, even if transfer times are included, which typically takes no more than a few second per data set, the GPU algorithms are still able to achieve significantly better performance than their equivalent CPU version. On the CPU, Gooding's method has the best performance at approximately 1.6 million solutions per second.

C. Demonstrating the Efficiency of Each Algorithm

The second test has been developed to test the efficiency of each test by monitoring the number of iterations each algorithm takes to converge for a range of orbit types. Because universal variable algorithm is not a function of the Lambert parameter (λ, q, or σ), parameters directly from Lambert's theorem must be used for the tests (radii, transfer angle, and transfer time).

To compare the efficiency of each of the four algorithms, a series of tests were developed to simulate the types of missions for which

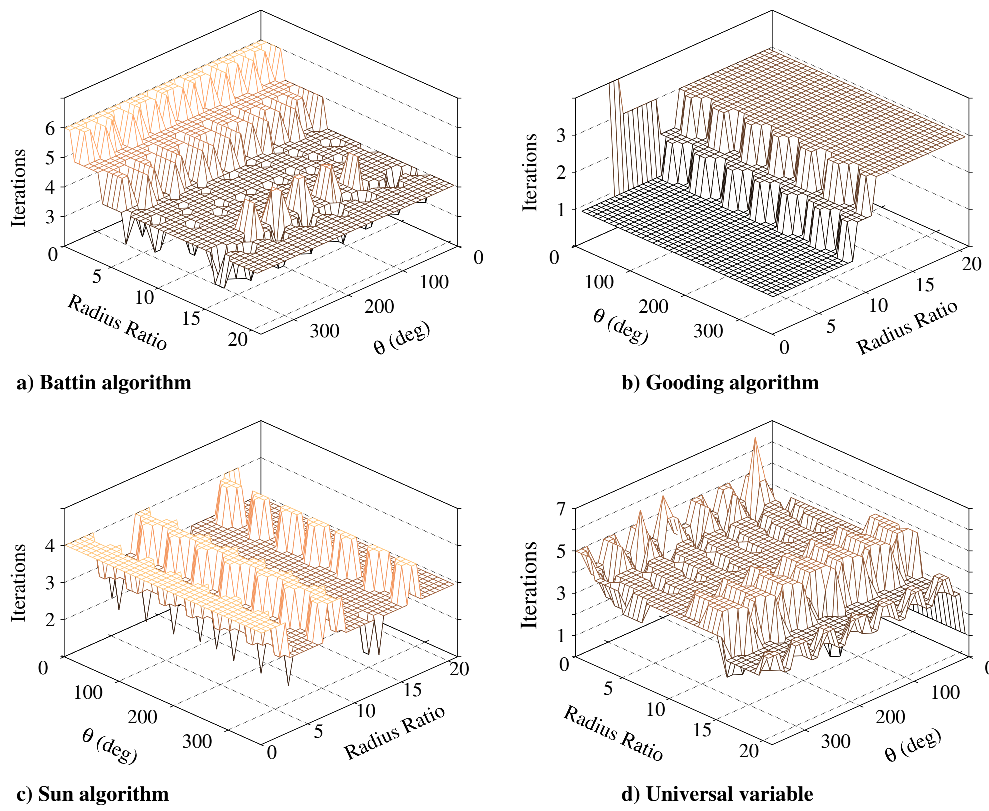

Fig. 3 Comparison of various Lambert algorithms for the results with a Δt of 5 years.

Lambert's problem is typically solved. The tests are designed to include all orbit types, from very hyperbolic to highly elliptical orbits. In each test case, an orbit around the sun is assumed with r_2 held constant at 1 AU (1 AU $= 149,599,650$ km). The initial radius r_1 is then varied from 0.01 to 10,000 AU while the transfer angle θ is simultaneously varied from 0 to 360 deg. Each test is then distinguished by the required transfer times of 0.1, 5, 25, and 100 years, respectively. For sun-centered orbits, the sun gravitational parameter μ has a value of $132,712,440,018\ \mathrm{km}^2/\mathrm{s}^3$.

Each algorithm can then be compared by examining the number of iterations versus radius ratio r_1/r_2 and transfer angle θ. A comparison of the number of iterations required for each algorithm

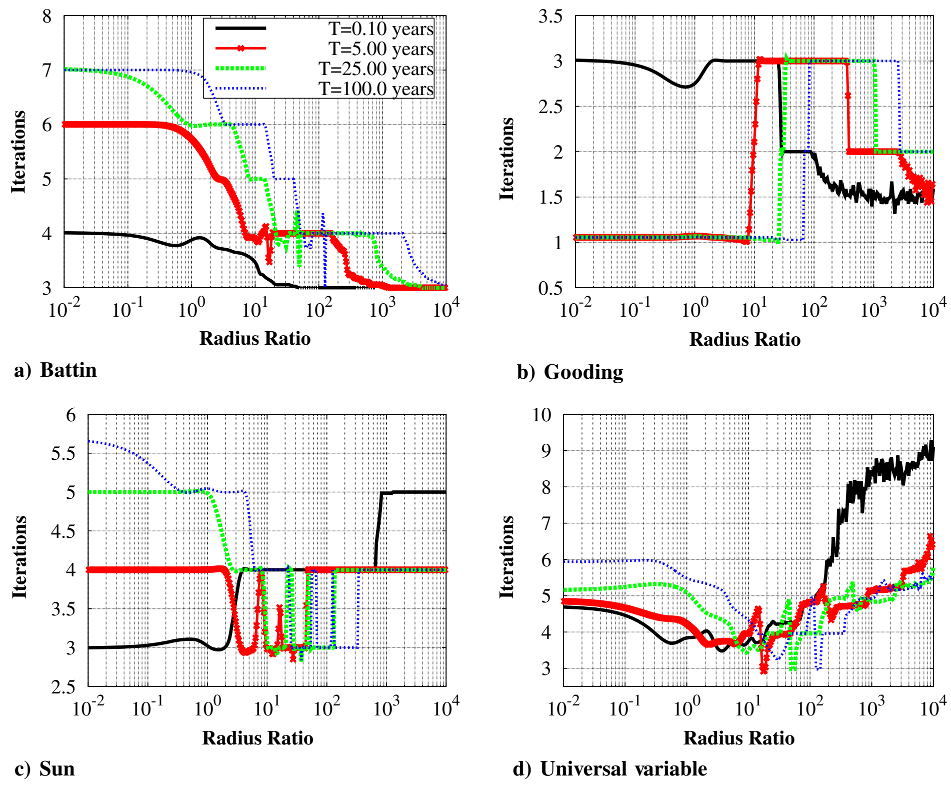

Fig. 4 Comparison of various Lambert algorithms for the each of the four transfer times.

Table 5 Average iterations required for each test case

Δt, years	Battin	Gooding	Sun	Universal variable
0.1	3.39	2.48	3.68	5.23
5	4.75	1.66	3.84	4.62
25	5.35	1.62	4.31	4.80
100	5.67	1.56	4.58	5.17

for the third test case, with a Δt of 5 years, is shown in Fig. 3. From these figures, it is apparent that the number of iterations required for each of the four algorithms varies very little with radius ratio. The same general trend is true for the other three test cases.

With this knowledge, it is convenient to compare the number of iterations required versus the radius ratio. This is accomplished by averaging the number of iterations for each radius ratio over the entire range of transfer angles. The results for all four test cases are shown in Fig. 4, which is a semilog plot with the radius ratio represented on the log scale. Several conclusions can be drawn for each solution method to Lambert's problem by examining this figure.

The Battin method is most efficient for short transfer times. As the plots show, the required number of iterations typically increases as the time-of-flight is increased. For each test case, the number of iterations required also decreases as the radius ratio increases. For the worst-case scenario (large transfer times and small radius ratios), the algorithm takes up to seven iterations to converge. For all other orbit combinations, convergence typically occurs within three to five iterations. The average number of iterations required for each test case is shown in Table 5.

When comparing the number of required iterations, the Gooding algorithm has the most predictable results. Except for the case when the transfer angle was exactly 0 or 360 deg, as illustrated in Fig. 3, convergence always occurs within three iterations. This high convergence rate is due to the extremely accurate initial guess using the bilinear initial guess approximations. Unlike Battin's method, this algorithm requires fewer iterations as the time of flight is increased. However, as the radius ratio increases, the number of iterations for the Gooding algorithm also increases. Out of the four Lambert algorithms, Gooding's algorithm is the most well behaved, resulting in extremely predictable performance.

Like Gooding's method, Sun's algorithm is well behaved. When solutions have large transfers times and small radius ratios, the algorithm typically converges within no more than four iterations. Except in cases where the initial guess is extremely close to the solution, this algorithm takes no more than three iterations to converge. In the worst-case scenario, this method takes no more than six iterations to converge on a solution.

The universal variable solution algorithm consistently requires an average of approximately five iterations to determine a solution for each of the transfer times. This method is well behaved, except when the transfer times are short and the radius ratio is greater than 10. This method can take up to nine iterations to converge when the transfer is very hyperbolic (low transfer time and large radius ratio), which is a known limitation of this method. In most cases, convergence typically occurs within six iterations.

IV. Conclusions

The Battin, Gooding, and Sun algorithms are able to converge for nearly 100% of all orbit combinations. Both the Gooding and Sun algorithms have approximately the same performance, whereas the Battin algorithm is consistently slower. The universal variable method is conceptually the easiest method to understand and is the most commonly discussed method in the literature, but it lacks the robustness of the other three methods. If the efficiency of the algorithm is the most important factor, both Gooding's and Sun's methods, as implemented here, are likely the best to use.

The performance for each Lambert algorithm, when run on the graphics processing unit (GPU), has been compared. Each Lambert solution algorithm is able to compute tens of millions of solutions per second, with Sun's method having the best performance at nearly 32 million solutions per second. This represents an increase in performance of two orders of magnitude when using GPUs over standard CPU algorithms. Although even the grid search CPU run times are not high when compared with common high-performance computing algorithms, mission optimization algorithms often require run times ranging from hours to multiple days. By offloading computations for solutions to Lambert's problem to GPU(s) and sufficiently parallelizing optimization algorithms, performance increases of up to two orders of magnitude can be realized when compared with standard CPU algorithms. This will allow mission designers to quickly compute complex trajectories when evaluating potential mission architectures.

Acknowledgments

This research has been supported in part by NASA's Iowa Space Grant Consortium and a NASA Innovative Advanced Concepts Phase 2 project.

References

[1] Hall, A., "On a Theorem of Lambert's," *Analyst*, Vol. 6, No. 6, 1879, pp. 171–173. doi:10.2307/2636268

[2] Lancaster, E. R., and Blanchard, R. C., "A Unified Form of Lambert's Theorem," NASA TN-D-5368, Sept. 1969.

[3] Gooding, R. H., "On the Solution of Lambert's Orbital Boundary-Value Problem," *Royal Aerospace Establishment*, NASA TND-5368, Greenbelt, MD, April 1988.

[4] Loechler, L., "An Elegant Lambert Algorithm for Multiple Revolution Orbits," M.S. Thesis, Massachusetts Inst. of Technology, Cambridge, MA, May 1988.

[5] Bate, R., Mueller, D., and White, J., "Orbit Determination from Two Positions and Time," *Fundamentals of Astrodynamics*, 1st ed., Dover, New York, 1971, Chap. 5.

[6] Battin, R., "Solving Lambert's Problem," *An Introduction to the Mathematics and Methods of Astrodynamics*, AIAA Education Series, AIAA, Reston, VA, 1999, Chap. 7.

[7] Vallado, D., "Initial Orbit Determination," *Fundamentals of Astrodynamics and Applications*, 2nd ed., Microcosm, El Segundo, CA, 2004, Chap. 7.

[8] Arora, N., and Russell, R., "GPU Accelerated Multiple Revolution Lambert Solver for Fast Mission Design," *AAS/AIAA Astrodynamics Specialist Conference*, AAS Paper 2010-198, Georgia Inst. of Technology, Atlanta, Feb. 2010.

[9] "CUDA Fortran Programming Guide and Reference," Portran Group, Beaverton, OR, 2014.

[10] Curtis, H., "Preliminary Orbit Determination," *Orbital Mechanics for Engineering Students*, 1st ed., Elsevier, Oxford, 2005, Chap. 5.

[11] Sun, F. T., "On the Minimum Time Trajectory and Multiple Solutions of Lambert's Problem," *AAS/AIAA Astrodynamics Specialist Conference*, AAS Paper 1979-164, National Tsing Hau Univ., Taiwan, June 1979.

[12] Gooding, R. H., *A Procedure for the Solution of Lambert's Orbital Boundary-Value Problem*, Kluwer Academic, Norwell, MA, Jan. 1990, pp. 145–165.

[13] Halley, E., "Methodus Nova Accurata et Facilis Inveniendi Radices Æqationum Quarumcumque Generaliter, Sine Praevia Reductione. Per Edm. Halley," *Philosophical Transactions (1683–1775)*, Vol. 18, Nos. 207–214, 1694, pp. 136–148.

[14] Prussing, J. E., and Conway, B. A., "Position in Orbit as a Function of Time," *Orbital Mechanics*, Oxford Univ. Press, Oxford, 1993, Chap. 2.

[15] Chobotov, V. A., "Position and Velocity as a Function of Time," *Orbital Mechanics*, 3rd ed., AIAA Education Series, AIAA, Reston, VA, 2002, Chap. 4.

[16] Wagner, S., and Wie, B., "GPU Accelerated Lambert Solution Methods for the Orbital Targeting Problem," *21st AAS/AIAA Space Flight Mechanics Meeting*, AAS Paper 2011-263, Iowa State Univ. of Science and Technology, Ames, IA, Feb. 2011.

[17] OpenMP, Software Package, Ver. 3.0, Lawrence Livermore National Lab., Livermore, CA, 2002.