Building a Prototype Analog Computer for Exact-1-in-3-SAT (*)

Michael G. Main
Department of Computer Science
University of Colorado at Boulder
Boulder, CO 80309-0430

Technical Report CU-CS-1035-07
October 2007

Abstract

This report describes theoretical work on the design of analog computers to solve discrete combinatorial problem. The approach is to convert a combinatorial problem into a dynamic system that can be simulated by piecewise linear analog circuits. Each instance, *P*, of a the original combinatorial problem corresponds to placing the analog computer in a particular known equilibrium state. If the equilibrium is stable, that implies that *P* has no solution. However, if the equilibrium is unstable, then the analog voltages along that instability (the "downhill" direction of the dynamical system) provide a solution to the problem *P*.

In addition to the theoretical description of the analog approach, this report also provides a detailed analog schematic for one particular combinatorial problem: Exact-1-in-3-SAT (an NP-complete problem). The design is modular, requiring $O(n^2)$ components for an n-variable instance of Exact-1-in-3-SAT. The construction of a prototype of this machine is underway for the 8-variable version of the problem, and this report describes some specific questions that can be answered by the prototype and a larger wire-wrapped 16- and 24-variable versions.

* This work is supported by a grant from the Engineering Excellence Fund of the College of Engineering at the University of Colorado in Boulder.

1. Background

During the age of transistor electronics, just at the birth of integrated circuitry, the design of analog computing components was motivated by analog computations in which voltages represent some physical quantity and the computing components manipulate voltages to carry out various numerical operations such as addition or definite integration. But by the time analog components—specifically the op-amps of the 1960s—had matured enough to implement such operations, it was already clear that digital components had the advantage in both speed and accuracy for the kinds of numerical problems that analog computer designers had hoped to solve. This is underscored by the fact that principle texts on analog computer programming all date from the 1960s [9].

Digital algorithms have their limits, too, particularly in large, well-known combinatorial problems. The NP-complete problems [5] share the trait of having no known polynomial-time algorithms despite decades of intense study. Many other important combinatorial problems, such as prime factorization, graph isomorphism and various pattern matching problems also run up against combinatorial barriers. These problems have motivated recent alternative computation proposals such as DNA computation [1] and quantum computers [3] that may take advantage of natural massive parallelism.

This report describes theoretical work and one detailed design for analog computers that solve combinatorial problems. The approach is promising because it explicitly does not depend on the relative inaccuracy of analog computations, but instead depends only on the ability of an analog computation to fall out of an unstable equilibrium.

The analog work has its origins in a physical machine given by Anastasios Vergis, Kenneth Steiglitz and Bradley Dickinson [13]. Their idea starts with an instance of Boolean Satisfiability [10], and from the specification of the instance they describe how to build a contraption of gears and other mechanical parts. One of the gears in the machine has a crank handle attached to it. Their analysis shows how the ability or inability of the handle to turn provides an answer to the given instance of Boolean Satisfiability. If the handle turns, we infer one answer to the problem; if it fails to turn, we infer a different answer.

An analysis of the gear machine [7] shows that the machine is analogous to a ball that rests in equilibrium in a multi-dimensional dynamic state space. The act of trying to turn the handle puts a downward force on the ball. If the equilibrium is a local minimum in the state space, then this force cannot budge the ball (the handle does not turn). On the other hand, if the equilibrium is not a local minimum, then the hope is that noise will soon jiggle the ball out of the equilibrium so that the downward force accelerates it away from the initial location (and the handle turns).

In a perfect world, though, the handle would never turn. Trying to turn the handle is analogous to exerting a straight down force on the ball that is precisely balanced at the apex of a perfect hill: There is no sideways force to jiggle the ball off the hill. The hope with the gear machine is that noise in the system will kick it into a downhill direction, even when the force is straight down and even when that downhill direction is exponentially small in terms of the total size of the state space. The machine was never

built, so we don't know whether the hope is valid or whether the expected time for that hope to occur is exponentially long.

Building the actual gear machine would be hard to prototype and test, but an analogous electronic circuit, described in this report, is feasible. The analog machine simulates the state of the gear machine by voltages in a circuit. The machine that is given here is to solve the NP-Complete problem, Exact-1-in-3-SAT, which we describe next.

2. Exact-1-in-3-SAT

The NP-complete problem, Exact-1-in-3-SAT [10] is defined here:

> Let V be a set of Boolean variables. A *3-clause* is any set of three of these variables. An *assignment* for V is a function $A:V \rightarrow \{\text{true, false}\}$. For an assignment A and a variable u in V, we say that u is a true variable if $A(u)$ = true; otherwise, we say that u is a false variable.
>
> Exact-1-in-3-SAT: Given a finite set C of 3-clauses in which no two variables appear together in more than one clause, does there exist an assignment such that every 3-clause in C contains exactly one true variable?

For an instance P of Exact-1-in-3-SAT, an assignment that results in exactly one true variable for each 3-clause is called a *valid* assignment. If P has at least one valid assignment, then P is called *satisfiable*; otherwise it is *unsatisfiable*.

3. Defining a Function f_P from an Instance P of Exact-1-in-3-SAT:

Let P be an instance of Exact-1-in-3-SAT with variables $V = \{x_1, x_2, \ldots, x_n\}$ and clauses C. From P, we can define a continuous piecewise quadratic function $f_P:\mathbf{R}^{n+1} \rightarrow \mathbf{R}$ (where $\mathbf{R}$ is the set of real numbers). For this definition, let $\mathbf{x} \in \mathbf{R}^{n+1}$ be a vector of n+1 real numbers. We will use the notation $x_1, x_2, \ldots, x_n$ for the first n components of $\mathbf{x}$ (distinguished from the elements of V by context only). The n+1st component of $\mathbf{x}$ is denoted by w.

3.1. The value of $f_P(\mathbf{x})$ is the sum of these pieces:

a. For each $x_i \in V$, $f_P(\mathbf{x})$ includes the sum of these four terms:

$$0.5w^2 - x_i^2$$
$$+$$
$$\text{if } (x_i > 0) \text{ then } 0.75\, x_i^2 \text{ else } 0$$
$$+$$
$$\text{if } (w - x_i < 0) \text{ then } 0.5(w - x_i)^2 \text{ else } 0$$
$$+$$
$$\text{if } (w + 2x_i < 0) \text{ then } 0.5(w + 2x_i)^2 \text{ else } 0$$

b. For each $\{ x_i, x_j, x_k \} \in C$, $f_P(\mathbf{x})$ includes the term $(x_i + x_j + x_k)^2$
c. $f_P(\mathbf{x})$ includes the single term $-0.2w^2$

The function f_P has a local minimum at the origin if and only if the instance P is unsatisfiable. In fact, if P is unsatisfiable, then $f_P(\mathbf{0})$ is a global minimum; otherwise, any downhill direction away from the origin provides an assignment that satisfies P defined by $A(x_i) = (x_i > 0)$ for all $x_i \in V$.

3.2. There are two dynamic systems that may be useful in determining whether f_P has a local minimum at the origin. The two systems are defined by the following differential equations.

a. Gradient Dynamic: $\mathbf{x'} = -\delta f_P/\delta \mathbf{x}$. Intuitively, the gradient dynamic is a system where the instantaneous velocity of a particle at position **x** is always equal to the direction of steepest descent.

b. Mechanical Dynamic: $\mathbf{x''} = -\delta f_P/\delta \mathbf{x}$. Intuitively, the mechanical dynamic is a system where the instantaneous acceleration of a particle at position **x** is always equal to the direction of steepest descent.

The idea is to build an analog computer in which voltage lines represent the components of a position **x**, and to allow the state of the machine to vary under one of these dynamics. If we start the machine at the origin, and the origin is not a local minimum, then noise in the system may eventually kick the machine into an path away from the origin. The gradient dynamic is intuitively simpler, although in simulations, the additional variation in the mechanical dynamic caused it to move away from the origin more often than the gradient dynamic. In any case, the hardware difference between the gradient and mechanical dynamics is small (replacing n inverting integrators with n inverting adders), so both machines can be built and tested with the same basic design.

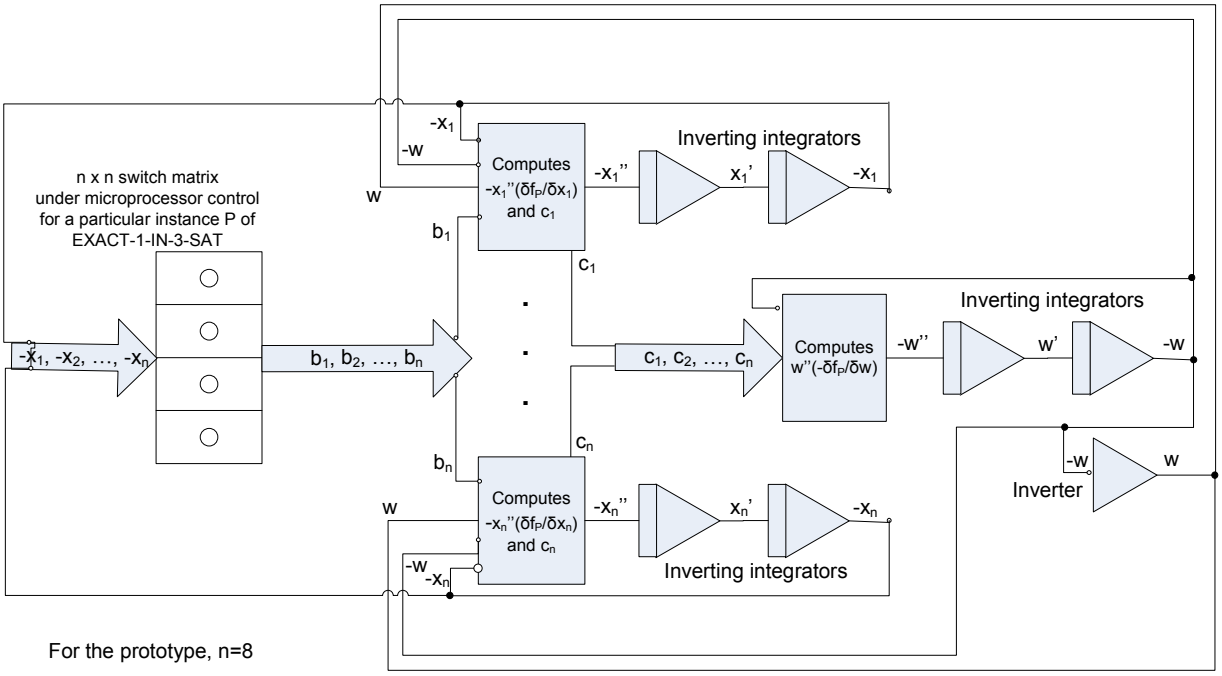
4. Schmatics for the Analog Machine (Mechanical Dynamic)

This section provides schematics for a prototype analog computer that can simulate the mechanical dynamic for any 8-variable instance of Exact-1-in-3-SAT. The design is modular, so that it can scale up to 16-, 24-, 32-… variable instances. The prototype version is programmed for a particular instance *P* via manually set switches. In the larger versions, these switches will be programmed by a microcontroller. The format of the individual computing elements comes from [9].

4.1. Overall Structure

The overall structure of the analog machine in shown in Figure 1. Details of the separate block components are given in Sections 4.2 through 4.4.

Figure 1: Block Diagram for the Mechanical Dynamic Analog Computer



For the prototype, n=8

b_i = sum of those x_j that share a clause with x_i

c_i = contribution of x_i to $\delta f_P/\delta w$ =

w
+
if $(w - x_i < 0)$ then $(w - x_i)$ else 0
+
if $(w + 2x_i < 0)$ then $(w + 2x_i)$ else 0

4.2. Subcircuit for Each Variable x_i

Each variable x_i has circuitry from Figure 1. The inputs to this circuitry are the current values of – x_i, w, $-w$ and b_i. This circuitry, along with inverting integrators that compute x_i' and $-x_i$, is shown in Figure 2. Altogether, it produces outputs of b_i , c_i , $-x_i$ and x_i' and Within the figure, the definitions of b_i , c_i and $-x_i''$ are given as:

$b_i = \sum_{\{x_i, x_j, x_k\} \in C} x_j + x_k$ This is the sum of all the variables that share a clause with x_i.

c_i = the contribution of x_i to $\delta f_P / \delta w$
= w
+
if ($w - x_i < 0$) then ($w - x_i$) else 0
+
if ($w + 2x_i < 0$) then ($w + 2x_i$) else 0

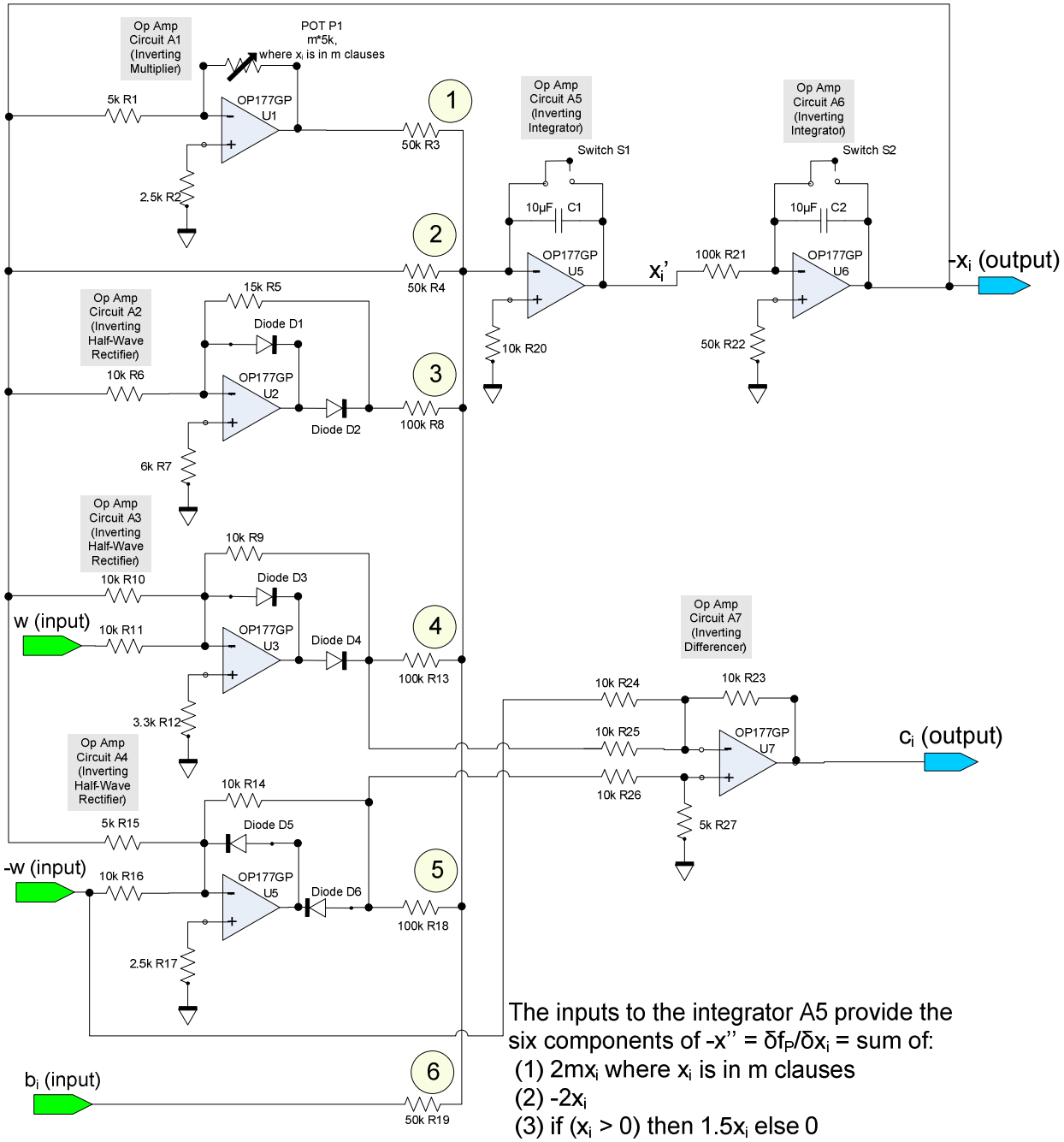
$-x_i'' = \delta f_P / \delta x$
= the sum of the following six pieces:

1. $2m\, x_i$, where m is the number of clauses that contain x_i
2. $-2x_i$
3. if ($x_i > 0$) then $1.5\, x_i$ else 0
4. if ($x_i - w > 0$) then ($x_i - w$) else 0
5. if ($w + 2x_i < 0$) then ($w + 2x_i < 0$) else 0
6. $2b_i$

The primary job of the circuit in Figure 2 is to maintain the value of $-x_i$ via the inverting integrator A6, which continually integrates x_i' . The value of x_i' is itself maintained by inverting integrator A5, which continually integrates $-x_i''$. The inputs to A5 are the components of x_i'' , which are computed by the adders and rectifier circuits (1) through (6). Also, two of the components ((5) and (4)) are combined in a differencing circuit A7 to produce c_i.

The two switches at the top of the inverting integrators must be manually closed to discharge the capacitors before the circuit is run. They are then opened for the actual computation. In addition, the potentiometer P1 must be manually set to $2500m\Omega$, where m is the number of clauses that contain x_i in the instance P.

Figure 2.
Subcircuit for Each Variable x_i



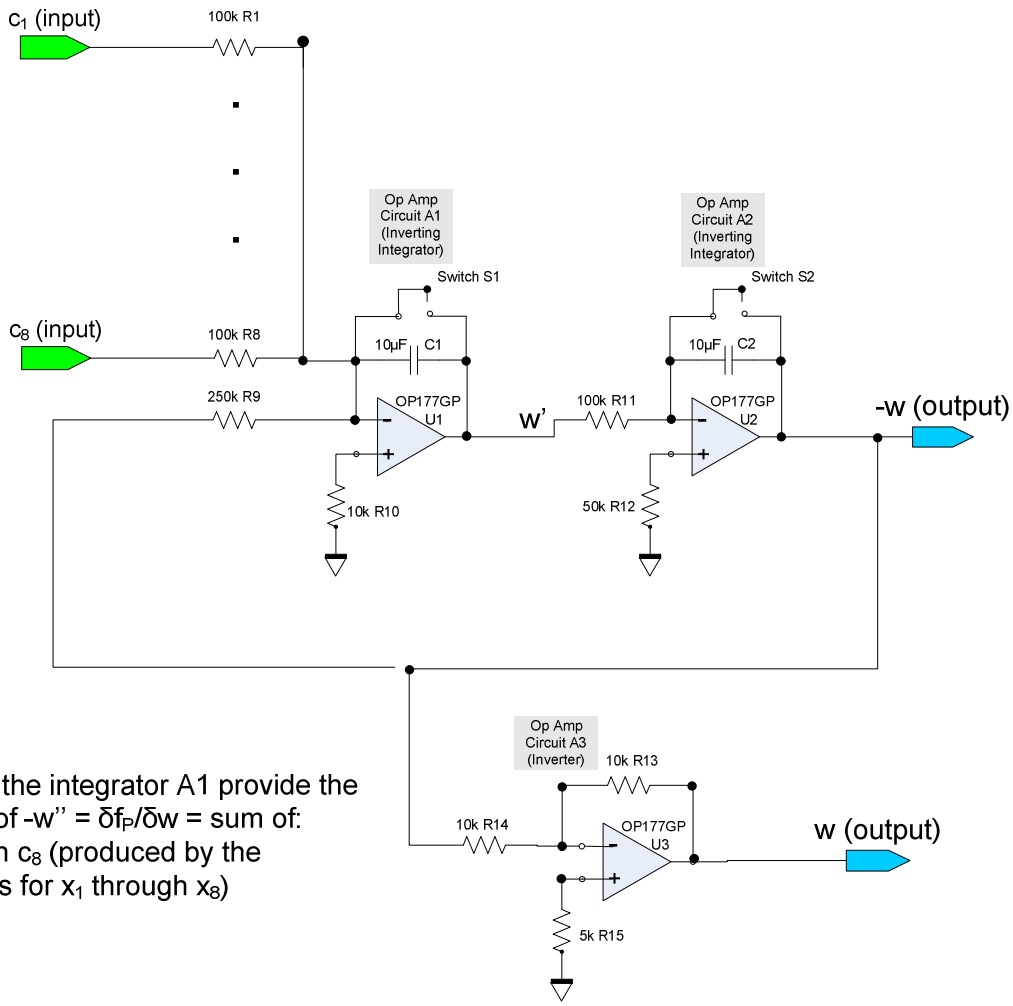
The inputs to the integrator A5 provide the six components of $-x'' = \delta f_P/\delta x_i$ = sum of:

(1) $2mx_i$ where x_i is in m clauses
(2) $-2x_i$
(3) if $(x_i > 0)$ then $1.5x_i$ else 0
(4) if $(x_i - w > 0)$ then $(x_i - w)$ else 0
(5) if $(w + 2x_i < 0)$ then $(w + 2x_i)$ else 0
(6) $2*b_i$ (2*sum of the other variables that appear in a clause with x_i)

4.3. Subcircuit for w

The circuitry to maintain the value of w (and –w) is shown in Figure 3. The use of the two inverting integrators A1 and A2 is similar to the use for the variables x_i, so the output of A1 is *w*' and the output of A2 is *–w*. An inverter is used to also provide an output of *w*.

Figure 3.
Subcircuit for w



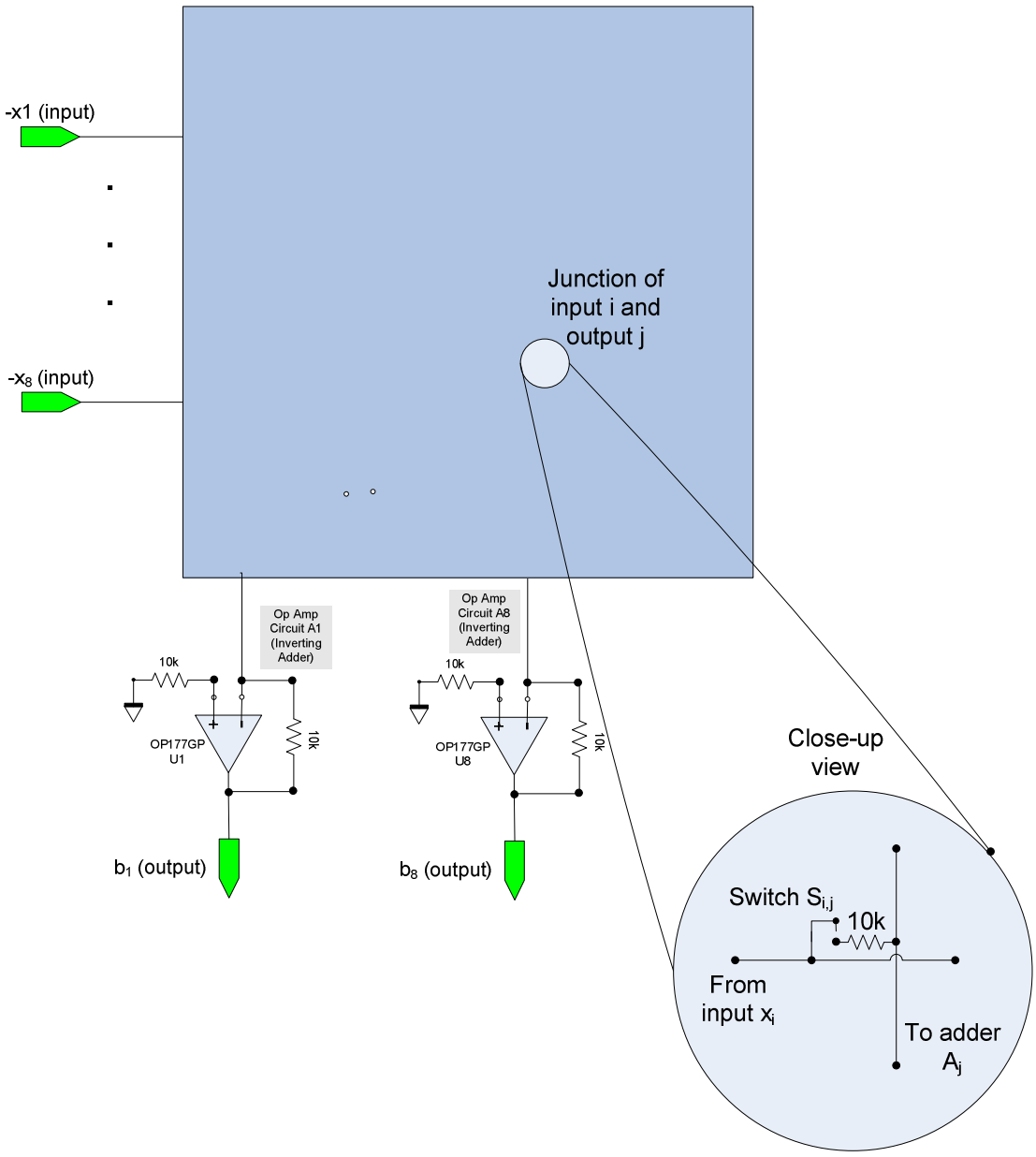
The inputs to the integrator A1 provide the components of $-w'' = \delta f_P/\delta w$ = sum of:
(1) c_1 through c_8 (produced by the subcircuits for x_1 through x_8)
(2) -0.4w

4.4 The Switch Matrix

Figure 4 shows the schematic for the n x n switch matrix from Figure 1 for the prototype case where $n = 8$. The inputs are the n voltages $-x_i$, and the outputs are the n voltages b_j (which is the sum of all the variables that share a clause with x_j.)

The switch matrix contains n x n switches that will be set by hand in the prototype, but will be set by a microcontroller in a larger version. Switch $S_{i,j}$ is closed if x_i shares a clause with x_j; otherwise it is
open.

Figure 4.
Switch Matrix



5. Questions for the Study to Address

These are some specific questions that building the hardware will answer:

5.1. Does the analog computer always stay in a stable equilibrium for an unsatisfiable instance of Exact-1-in-3-SAT?

5.2. Does the analog computer always fall off the equilibrium for a satisfiable instance of Exact-1-in-3-SAT? If so, how quickly? If not, what state is the machine in when it gets stuck?

5.3. How does the time to fall off the equilibrium vary with increasing n. For this question, 16-variable and 24-variable versions of the machine will be built, controlled by a microcontroller processor.

5.4 How do the frequency responses and voltage supplies of the amplifiers affect the speed of falling off an equilibrium?

In addition, the study will continue theoretical work to develop dynamic systems for other combinatorial problems. Of particular interest are computing prime factorizations (a problem with a fast quantum algorithm [11]), large pattern matching problems (relevant, for example, in DNA analysis [4]), determining whether two given graphs are isomorphic (the fundamental I-complete problem [12]), linear programming problems [8] and convex quadratic programming [6].

Bibiography

[1] Leonard M. Adelman. "Molecular Computations of Solutions to Combinatorial Problems," *Science* 266 (11), pp. 1021-1024.

[2] Martin Amos. *Theoretical and Experimental DNA Computation.* Springer-Verlag (2005).

[3] Iona Burda, *An Introduction to Quantum Computation*. Universal Publishers (2005).

[4] Lei Chen, Shiyong Lu and J. Ram. "Compressed pattern matching in DNA sequences," *Computational Systems Bioinformatics Conference*, pp. 62-68 (2004).

[5] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company (1979).

[6] M.K. Kozlov, S.P. Tarasov and L.G. Hacijan. "Polynomial solvability of convex quadratic programming," *Dokl. Akad. Nauk* SSSR 248 (1979), pp. 1049-1051. Translated to English in *Soviet Math Dokl.* 20, pp. 1108-1111.

[7] Michael Main. "Analog Solution of NP-Hard Problems," Department of Computer Science Technical Report, (Jan 5, 1994).

[8] David J. Pannell. *Introduction to Practical Linear Programming.* Wiley (1996).

[9] Michael G. Rekoff, Jr. *Analog Computer Programming.* Charles E. Merrill Books, Inc. (Columbus, Ohio, 1967).

[10] T.J. Schaefer. "The complexity of satisfiability problems," *Proc. 10th Ann. ACM Symposium on Theory of Computing.* Association for Computing Machinery (New York, 1978), pp. 216-226. Note that the Exact-1-in-3-SAT problem remains NP-complete even if negative literals are forbidden and no two variables appear together in more than one clause.

[11] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM J. Sci. Statist. Computing* 26, p. 1484. (1997)

[12] S. Skiena. "Graph Isomorphism," *Implementing Discrete Mathematics: Combinatorial and Graph Theory with Mathematica,* pp. 181-187, Addison-Wesley (1990).

[13] Anastasios Vergis, Kenneth Steiglitz and Bradley Dickenson. "The Complexity of Analog Computation," Mathematics and Computers in Simulation 28, pp. 91-113 (1986).